

Hortonworks Data Platform

Accessing Cloud Data

(December 17, 2019)

Hortonworks Data Platform: Accessing Cloud Data

Copyright © 2012-2019 Cloudera, Inc. All rights reserved.

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.

Licensed under the Apache License, Version 2.0 (the "License"); you may not use this file except in compliance with the License. You may obtain a copy of the License at

<http://www.apache.org/licenses/LICENSE-2.0>

Unless required by applicable law or agreed to in writing, software distributed under the License is distributed on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied. See the License for the specific language governing permissions and limitations under the License.

Table of Contents

1. About This Guide	1
2. The Cloud Storage Connectors	2
3. Working with Amazon S3	5
3.1. Limitations of Amazon S3	5
3.2. Configuring Access to S3	6
3.2.1. Using Instance Metadata to Authenticate	6
3.2.2. Using Configuration Properties to Authenticate	7
3.2.3. Using Environment Variables to Authenticate	8
3.2.4. Embedding Credentials in the URL to Authenticate (Deprecated)	8
3.3. Defining Authentication Providers	9
3.3.1. Using Temporary Session Credentials	10
3.3.2. Using Anonymous Login	10
3.3.3. Protecting S3 Credentials with Credential Providers	11
3.4. IAM Role Permissions for Working with S3	13
3.5. Referencing S3 Data in Applications	14
3.6. Configuring Per-Bucket Settings	14
3.6.1. Configuring Per-Bucket Settings to Access Data Around the World	16
3.7. Using S3Guard for Consistent S3 Metadata	18
3.7.1. Introduction to S3Guard	18
3.7.2. Configuring S3Guard	20
3.7.3. Monitoring and Maintaining S3Guard	26
3.7.4. Disabling S3Guard and Destroying a S3Guard Database	26
3.7.5. Pruning Old Data from S3Guard Tables	27
3.7.6. Importing a Bucket into S3Guard	27
3.7.7. Verifying that S3Guard is Enabled on a Bucket	27
3.7.8. Using the S3Guard CLI	28
3.7.9. S3Guard: Operational Issues	28
3.7.10. S3Guard: Known Issues	29
3.8. Safely Writing to S3 Through the S3A Committers	30
3.8.1. Introducing the S3A Committers	30
3.8.2. Enabling the Directory Committer in Hadoop	32
3.8.3. Configuring Directories for Intermediate Data	33
3.8.4. Using the Directory Committer in MapReduce	33
3.8.5. Enabling the Directory Committer in Spark	33
3.8.6. Verifying That an S3A Committer Was Used	34
3.8.7. Cleaning up After Failed Jobs	35
3.8.8. Using the S3Guard Command to List and Delete Uploads	35
3.8.9. Advanced Committer Configuration	36
3.8.10. Securing the S3A Committers	37
3.8.11. The S3A Committers and Third-Party Object Stores	37
3.8.12. Limitations of the S3A Committers	38
3.8.13. Troubleshooting the S3A Committers	38
3.9. Security Model and Operations on S3	39
3.10. S3A and Checksums (Advanced Feature)	40
3.11. A List of S3A Configuration Properties	41
3.12. Encrypting Data on S3	52
3.12.1. SSE-S3: Amazon S3-Managed Encryption Keys	53
3.12.2. SSE-KMS: Amazon S3-KMS Managed Encryption Keys	53

3.12.3. SSE-C: Server-Side Encryption with Customer-Provided Encryption Keys	55
3.12.4. Configuring Encryption for Specific Buckets	55
3.12.5. Mandating Encryption for an S3 Bucket	56
3.12.6. Performance Impact of Encryption	57
3.13. Improving Performance for S3A	58
3.13.1. Working with Local S3 Buckets	58
3.13.2. Configuring and Tuning S3A Block Upload	58
3.13.3. Optimizing S3A read performance for different file types	61
3.13.4. Improving Load-Balancing Behavior for S3	63
3.13.5. S3 Performance Checklist	63
3.14. Working with Third-party S3-compatible Object Stores	64
3.15. Troubleshooting S3	64
3.15.1. Authentication Failures	65
3.15.2. Classpath Related Errors	67
3.15.3. Connectivity Problems	67
3.15.4. Errors During Delete or Rename of Files	70
3.15.5. Errors Related to Visible S3 Inconsistency	71
3.15.6. Troubleshooting Encryption	71
4. Working with ADLS	74
4.1. Configuring Access to ADLS	74
4.1.1. Configure Access by Using Client Credential	74
4.1.2. Configure Access by Using Token-Based Authentication	76
4.2. Protecting the Azure Credentials for ADLS with Credential Providers	77
4.3. Referencing ADLS in URLs	78
4.4. Configuring User and Group Representation	78
4.5. ADLS Proxy Setup	78
5. Working with WASB	79
5.1. Configuring Access to WASB	79
5.2. Protecting the Azure Credentials for WASB with Credential Providers	79
5.3. Protecting the Azure Credentials for WASB within an Encrypted File	80
5.4. Referencing WASB in URLs	81
5.5. Configuring Page Blob Support	81
5.6. Configuring Atomic Folder Rename	82
5.7. Configuring Support for Append API	82
5.8. Configuring Multithread Support	83
5.9. Configuring WASB Secure Mode	83
5.10. Configuring Authorization Support in WASB	84
6. Working with Google Cloud Storage	86
6.1. Configuring Access to Google Cloud Storage	86
6.1.1. Create a GCP Service Account	86
6.1.2. Create a Custom Role	87
6.1.3. Modify GCS Bucket Permissions	89
6.1.4. Configure Access to GCS from Your Cluster	90
6.2. Additional Configuration Options for GCS	91
7. Accessing Cloud Data in Hive	93
7.1. Hive and S3: The Need for S3Guard	93
7.2. Exposing Cloud Data as Hive Tables	93
7.3. Populating Partition-Related Information	94
7.4. Analyzing Tables	95
7.5. Improving Hive Performance with Cloud Object Stores	95

8. Accessing Cloud Data in Spark	97
8.1. Using S3 as a Safe and Fast Destination of Work	98
8.2. Improving Spark Performance with Cloud Storage	98
8.2.1. Improving ORC and Parquet Read Performance	99
8.2.2. Accelerating S3 Read Performance	99
8.2.3. Accelerating Azure Read Performance	100
8.2.4. Putting it All Together: spark-defaults.conf	100
9. Copying Cloud Data with Hadoop	101
9.1. Running FS Shell Commands	101
9.1.1. Commands That May Be Slower with Cloud Object Storage	102
9.1.2. Unsupported Filesystem Operations	103
9.1.3. Deleting Files on Cloud Object Stores	104
9.1.4. Overwriting Objects on Amazon S3	104
9.1.5. Timestamps on Cloud Object Stores	104
9.2. Copying Data with DistCp	105
9.2.1. Using DistCp with S3	106
9.2.2. Using DistCp with Azure ADLS and WASB	107
9.2.3. DistCp and Proxy Settings	108
9.2.4. Improving DistCp Performance	108

List of Figures

2.1. HDP Cloud Storage Connector Architecture 3

List of Tables

2.1. Cloud Storage Connectors	3
3.1. Authentication Options for Different Deployment Scenarios	6
3.2. S3Guard Configuration Parameters	24
3.3. S3A Fast Upload Configuration Options	59
3.4. S3A Upload Tuning Options	61
6.1. Overview of Configuring Access to Google Cloud Storage	86
6.2. Optional Properties Related to Google Cloud Storage	92
7.1. Improving General Performance	95
7.2. Accelerating ORC Reads in Hive	95
7.3. Accelerating ETL Jobs	96

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), Microsoft Azure (ADLS, WASB), and Google Cloud Storage (GCS) (Technical Preview)

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), Microsoft Azure, and Google Cloud Platform (GCP). 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.

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.

2. The Cloud Storage Connectors

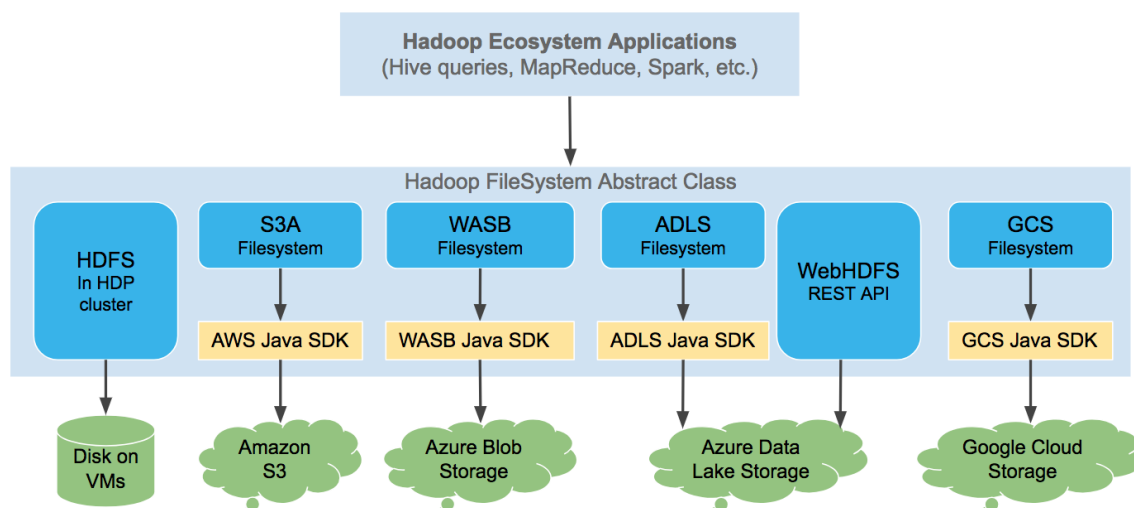
When deploying HDP clusters on cloud IaaS, you can take advantage of the native integration with the object storage services available on each of the cloud platforms: **Amazon S3 on AWS, ADLS and WASB on Azure, and GCS on Google Cloud**. This integration is via cloud storage connectors included with HDP. Their primary function is to help you connect to, access, and work with data the cloud storage services.

The cloud connectors allow you to access and work with data stored in Amazon S3, Azure ADLS and Azure WASB storage services, and Google Cloud Storage, 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.
- Back up Hadoop clusters using `distcp`.

The cloud object store 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 AWS Java SDK to access the web service, and provides Hadoop applications with a filesystem view of the buckets. Applications can 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 data. Applications can 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.

Similarly, the ADLS connector implements the Hadoop filesystem interface using the ADLS Java SDK to access the web service. Applications can manipulate data stored in ADLS with the URL starting with the `adl://` prefix.

Google Cloud Storage (GCS) is an object store. The GCS connector implements the Hadoop filesystem interface using GCS Java SDK to access the web service, and provides Hadoop applications with a filesystem view of the buckets. Applications can manipulate data stored in Google Cloud Storage buckets with an URL starting with the `gcs://` 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.	<code>s3a://</code>
Azure Data Lake Store (ADLS)	The ADLS connector enables reading and writing files stored in the ADLS file system.	<code>adl://</code>
Windows Azure Storage Blob (WASB)	The WASB connector enables reading and writing both block blobs and page blobs from and to WASB object store.	<code>wasb://</code>

Cloud Storage Service	Connector Description	URL Prefix
Google Cloud Storage (GCS)	The GCS connector supports access to Google Cloud Storage.	gcs://

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, WASB, or GCS use a fully qualified URL for that connector.

3. Working with Amazon S3

The [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 be restricted to different users or [IAM roles](#).
- Data stored in an Amazon S3 bucket is billed based on the size of data how long it is stored, and on operations accessing this data. 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 downloaded from an Amazon S3 bucket to any host over the internet is also 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. 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 non-atomic.
- 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. To mitigate the inconsistency issues, you can configure S3Guard. To learn more, refer to [Using S3Guard for Consistent S3 Metadata \[18\]](#).

- 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 Access to 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 dangerous and deprecated. 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](#).

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.

Also see [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 (Deprecated)

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.

3.3. 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
<code>org.apache.hadoop.fs.s3a.SimpleAWSCredentialsProvider</code>	Standard credential support through configuration properties. It does not support in-URL authentication.
<code>org.apache.hadoop.fs.s3a.TemporaryAWSCredentialsProvider</code>	Session authentication
<code>org.apache.hadoop.fs.s3a.AnonymousAWSCredentialsProvider</code>	Anonymous login. Use for accessing public data without providing any credentials at all.

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

Classname	Description
<code>com.amazonaws.auth.EnvironmentVariableCredentialsProvider</code>	AWS Environment Variables
<code>com.amazonaws.auth.InstanceProfileCredentialsProvider</code>	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.

3.3.1. 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.

3.3.2. 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.

3.3.3. 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 \[11\]](#)
2. [Configuring the Hadoop Security Credential Provider Path Property \[12\]](#)

In addition, if you are using per-bucket credentials, refer to [Customizing Per-Bucket Secrets Held in Credential Files \[13\]](#).

3.3.3.1. 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.

3.3.3.2. 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/

```

3.3.3.3. 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 \[7\]](#)

[Credential Provider API](#)

3.4. IAM Role Permissions for Working with S3

AWS IAM roles can be granted a broad set of permissions, including options such as "write only", "delete forbidden", "listing and aborting multipart uploads". These permissions can be explicitly granted to paths under the base store.

The S3A connector only supports a simplistic model of access: buckets may be read-only, or the caller has full access. Any set of permissions between these is likely to cause filesystem operations to fail partway through. For example, attempting to rename data from a path to which the caller only has a read access to one with write access might copy some of the files and then fail, leaving the source directory unchanged, and the destination

directory with a partial copy of the files. As another example, the S3A committers need the ability to list multipart uploads (`s3:ListBucketMultipartUploads`), and abort them (`s3:AbortMultipartUpload`).

Here then, are the basic permissions required for read-only and read-write access to S3 through the S3A connector.

Permissions required for read-only access to an S3 bucket

```
s3:Get*
s3:ListBucket
```

Permissions required for read/write access to an S3 bucket

```
s3:Get*
s3:Delete*
s3:Put*
s3:ListBucket
s3:ListBucketMultipartUploads
s3:AbortMultipartUpload
```

Futher permissions are required for [S3Guard](#) and for working with files encrypted with [SSE-KMS](#).

3.5. Referencing S3 Data in Applications

You can reference data in Amazon S3 using a 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
```

3.6. 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 \[7\]](#)

[Customizing Per-Bucket Secrets Held in Credential Files \[13\]](#)

3.6.1. 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 `yourcore-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.

3.7. Using S3Guard for Consistent S3 Metadata

S3Guard mitigates the issues related to S3's eventual consistency on listings by using a table on Amazon DynamoDB as a consistent metadata store. This guarantees a consistent view of data stored in S3. In addition, S3Guard *may* improve query performance by reducing the number of times S3 needs to be contacted, —as DynamoDB is significantly faster.

3.7.1. Introduction to S3Guard

Amazon S3 is an *object store*, not a filesystem. There are no directories, only objects. The S3A connector lets Hadoop, Hive and Spark applications see files in a directory tree, but really they are working on the objects underneath, by listing them and working on each one one-by-one.

Some of the operations which filesystems support are actually absent, with rename being the key one. The S3A connector mimics file or directory rename, by copying each file then deleting the original, which takes about 6-10 megabytes/second.

The S3 Object Store is "eventually consistent": when a file is deleted or overwritten it can take time for that change to propagate across all servers replicating the data. As a result, newly deleted files can still be visible, while queries of updated files can return the old version.

How long is "eventually"? There's no official number; the paper "[Benchmarking Eventual Consistency](#)" has shown it can vary by time of day, and be ten seconds or more -sometimes much more.

A critical problem is listing inconsistency: when a query is made of S3 to list all objects under a specific path, that listing can be out of date. This means that those operation on files under a "directory" mimicked by listing and acting on all objects underneath it are at risk of not seeing the complete list of files. Newly created files are at most risk.

This may affect the following operations on S3 data:

- When **listing files**, newly created objects may not be listed immediately and deleted objects may continue to be listed – which means that your input for data processing may be incorrect. In Hive, Spark, or MapReduce, this could lead to erroneous results. In the worst case, it could potentially lead to data loss at the time of data movement.
- When **renaming directories**, the listing may be incomplete or out of date, so the rename operation loses files. This is very dangerous as MapReduce, Hive, Spark and Tez all rely on rename to commit the output of workers to the final output of the job. If data is lost, the output is incorrect –something which may not be immediately obvious.
- When **deleting directories**, the listing may be inconsistent, so not all objects are deleted. If another job writes data to the same directory path, the old data may still be present.
- During an **ETL workflow**, in a sequence of multiple jobs that form the workflow, the next job is launched soon after the previous job has been completed. Applications such as Oozie rely on marker files to trigger the subsequent workflows. Any delay in the visibility of these files can lead to delays in the subsequent workflows.
- During **existence-guarded path operations**, if a deleted file which has the same name as a target path appears in a listing, some actions may unexpectedly fail due to the target path being present – even though the file has already been deleted.

This eventually consistent behavior of S3 can cause seemingly unpredictable results from queries made against it, limiting the practical utility of the S3A connector for use cases where data gets modified.

S3Guard mitigates the issues related to S3's eventual consistency on listings by using a table on Amazon DynamoDB as a consistent metadata store. This guarantees a consistent view of data stored in S3. In addition, S3Guard *may* improve query performance by reducing the number of times S3 needs to be contacted, –as DynamoDB is significantly faster.

How S3Guard Works

S3Guard is a feature in the Hadoop S3A connector which uses Amazon's DynamoDB to cache information about created and deleted files, "The S3Guard Database".

When an application using the S3A Connector with S3Guard enabled manipulates objects in S3, such as creating, deleting or "renaming" them, the S3Guard database is updated.

Newly created/copied files are added to the table, while deleted files have "tombstone markers" added to indicate that they have been deleted.

When a directory is listed on S3, the information from the S3Guard database is used to update the listing with this information: new files are added while those files with tombstone markers are not included in a listing.

As a result, the listing is up to date with all operations performed on the directory by all clients using S3Guard.

When a file is opened for reading, or any existence check made, the S3Guard database is checked first. If an entry is found in the database, that is used as the response—omitting all checks of S3 itself. This includes tombstone markers, which are used as evidence that a file does not exist. The caller is given the current state of the object (existence/nonexistence, size and type), without S3 being queried at all. This can be significantly faster than interacting with S3.

With this design, directory listings are kept consistent across a sequence of operations, even across multiple servers—indeed, across multiple Hadoop clusters.

What S3Guard Cannot Do

Guarantee access to updated data. While it keeps the listing consistent, attempts to read newly updated data may still return the old value. The only way to guarantee that newly-created data is immediately visible is to use different filenames when writing the new data.

Mimic the "directory rename is a single atomic transaction" behavior of a filesystem like HDFS. Directory renames are still slow and visible while in progress. This means that if the operations fail partway through, the source and destination paths may contain a mix (including some duplicate) copies of data files.

3.7.2. Configuring S3Guard

To configure S3Guard, perform the following tasks:

1. [Preparing the S3 Bucket \[20\]](#)
2. [Choosing a DynamoDB Table and IO Capacity \[21\]](#)
3. [Create DynamoDB access policy](#) in the IAM console on your AWS account.
4. You can optionally [restrict access to S3Guard tables](#)
5. [Configure S3Guard](#) in the Ambari web UI.
6. [Create the S3Guard Table in DynamoDB \[25\]](#)

3.7.2.1. Preparing the S3 Bucket

S3Guard can protect any read/write or read-only S3 bucket. To begin "guarding" a bucket, the first step is to create that bucket if it does not already exist.

Existing buckets can use S3Guard: after creation the data can be imported, though this is not necessary.

Except in when S3Guard is configured to operate in “authoritative mode” (which we do not recommend), clients using S3Guard can work with the same S3 bucket as applications not using S3Guard. Those clients not using S3Guard do not get the consistency, but their use can continue unchanged.

3.7.2.2. Choosing a DynamoDB Table and IO Capacity

Amazon's DynamoDB service bills its customers for allocated writes per second ("Write Capacity"), reads per second ("Read Capacity") and storage, which is billed by the gigabyte.

S3Guard uses DynamoDB read capacity when it is listing directories or querying file status; write capacity when it is updating the database after querying S3 or when a client using S3Guard creates, updates or deletes objects in S3. The more allocated capacity, the more throughput clients using S3Guard can get when interacting with the S3Guard database, but the higher the monthly costs—even when the database is not being used.

S3Guard only stores filenames and other metadata in its tables: for each directory only the name is stored; for a file the size and timestamp are preserved; storage costs should be low. It is the amount of provisioned IO capacity which can be the main expense for using S3Guard.

There are three strategies to manage IO capacity:

1. Dedicate a single table to a bucket; choose read and write capacity based on expected use. This isolates the table for both capacity and security. When a bucket is not being used, you are still billed for the allocated capacity.
2. Share a single table with all buckets in a single AWS region; choose read and write capacity based on expected average use across all buckets. This balances capacity across buckets. However, it gives all clients the ability to this the contents of all buckets in the table.
3. Either of the two previous strategies, but using [DynamoDB Auto Scaling](#) to manage the capacity. This can keep costs down when idle. yet allow it to scale up to handle higher load.

If using allocated capacity, we would advise using a shared table; when using auto scaling, the main disadvantage of a single table is lost, and because of the security isolation, it is the policy we recommend.

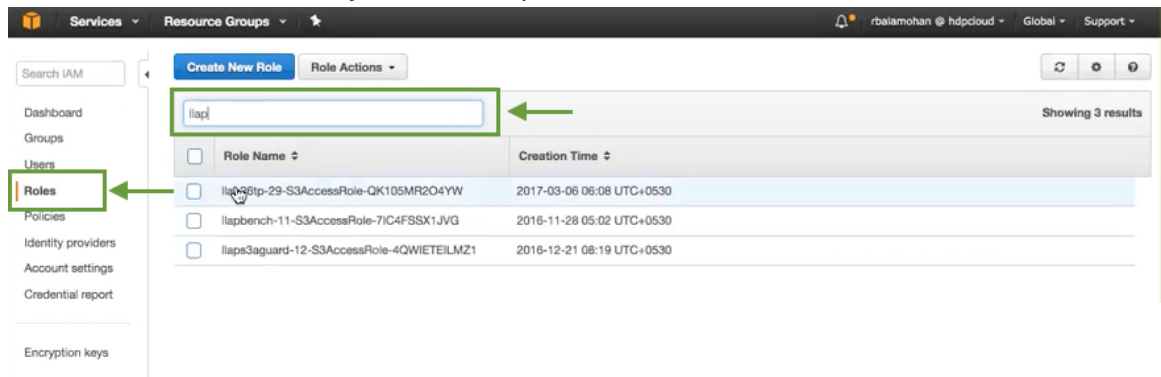
If using autoscaling, be aware that when the scaling up/down takes place, the table may briefly switch into the UPDATING state. Existing connections to the database should continue, but trying to start new connections will fail with an error message that the table "did not transition into ACTIVE state.". This should be a transient failure; after DynamoDB has marked the table as active, new work can be executed against it.

3.7.2.3. Creating The DynamoDB Access Policy

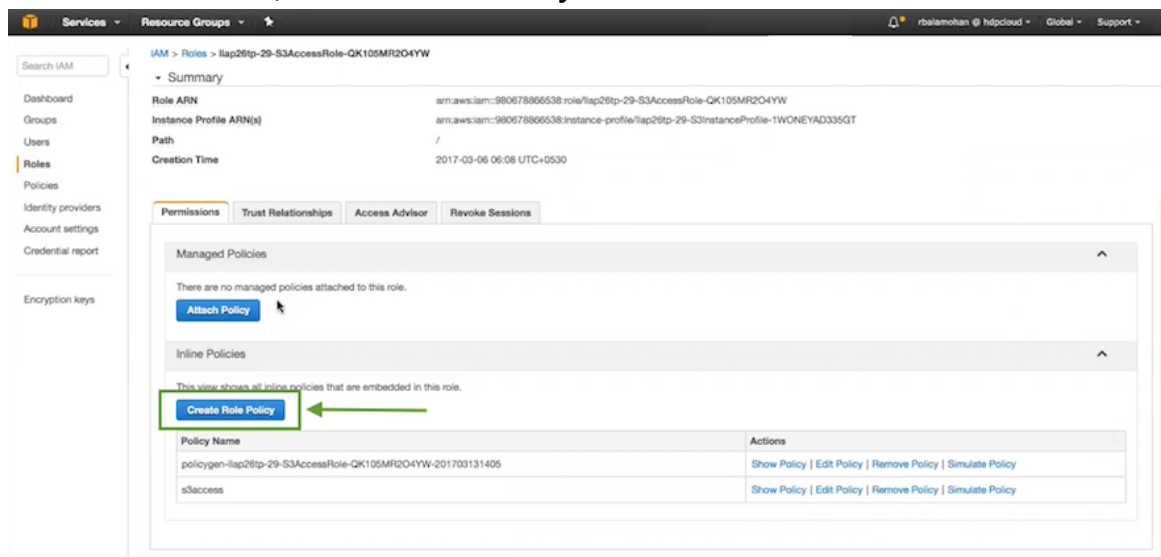
In order to configure S3Guard, you must to provide read and write permissions for the DynamoDB table that S3Guard will create and use. To do this, you must add a DynamoDB access policy to your [IAM role](#) using the following steps:

1. Log in to your AWS account and navigate to the Identity and Access Management (IAM) console.

- In the IAM console, select **Roles** from the left pane.
- Search for an IAM role that you want to update:



- Click on the role.
- In the Permissions tab, click **Create Role Policy**:



- Click **Select** next to the Policy Generator:

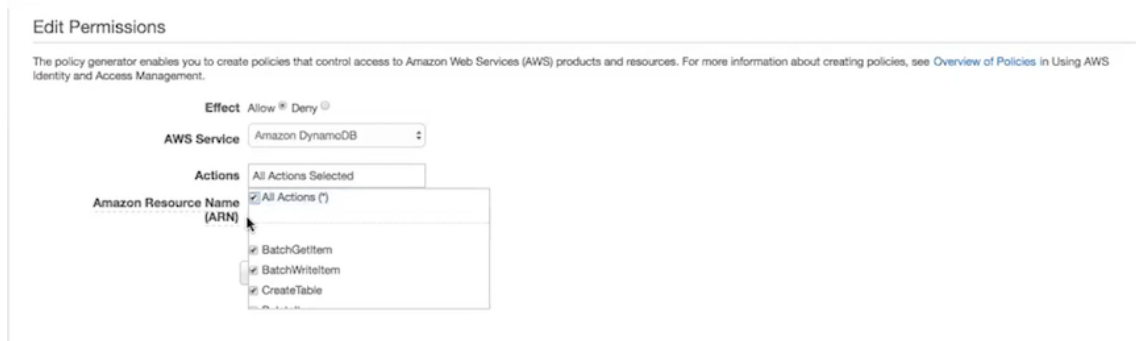


- Enter:

Parameter	Value
Effect	Allow
AWS Service	Amazon DynamoDB
Actions	All Actions

Parameter	Value
Amazon Resource Name (ARN)	*

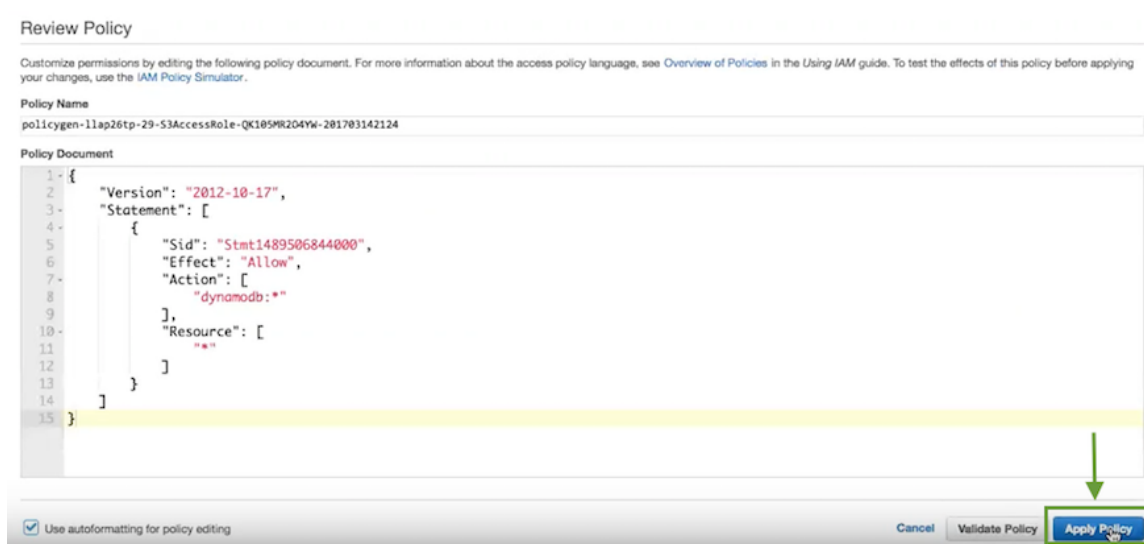
Your configuration should look similar to:



8. Click **Add Statement**.

9. Click **Next Step**.

10. On the "Review Policy" page, review your new policy and then click **Apply Policy**:



Now the policy will be attached to your IAM role and your cluster will be able to talk to DynamoDB, including creating a table for S3 metadata when S3Guard is configured.

You must also [configure S3Guard in Ambari](#).

3.7.2.4. Restricting Access to S3Guard Tables

To restricting access to S3Guard tables, here are the permissions needed for simply using the table:

```

dynamodb:BatchGetItem
dynamodb:BatchWriteItem
dynamodb>DeleteItem
dynamodb:DescribeTable
dynamodb:GetItem
dynamodb:PutItem
dynamodb:Query
dynamodb:UpdateItem

```

For the `hadoop s3guard` table management commands, extra permissions are required:

```

dynamodb:CreateTable
dynamodb:DescribeLimits
dynamodb>DeleteTable
dynamodb:Scan
dynamodb:TagResource
dynamodb:UntagResource
dynamodb:UpdateTable

```

It is best to remove these rights, especially the `dynamodb:CreateTable` and `dynamodb>DeleteTable` permissions from non-administrators.

3.7.2.5. Configuring S3Guard in Ambari

After you have created DynamoDB access policy, set the following S3Guard configuration parameters for each S3 bucket that you want to "guard".

1. In Ambari web UI, select the **HDFS** service and navigate to **Configs > Advanced > Custom hdfs-site**.
2. Set the following configuration parameters for each bucket that you want to "guard". To configure S3Guard for a specific bucket, replace `fs.s3a.` with the `fs.s3a.bucket.<bucketname>`, where "bucketname" is the name of your bucket.
3. After you've added the configuration parameters, click **Save** to save the configuration changes.
4. Restart all affected components.

Table 3.2. S3Guard Configuration Parameters

Base Parameter	Default Value	Setting for S3Guard
<code>fs.s3a.metastorestore.impl</code>	<code>org.apache.hadoop.fs.s3a.S3NullMetadataStore</code>	Set this to <code>org.apache.hadoop.fs.s3a.s3guard.DynamoDBMetadataStore</code> .
<code>fs.s3a.s3guard.ddb.table.create</code>	<code>false</code>	Set this to "true" to automatically create the DynamoDB table.
<code>fs.s3a.s3guard.ddb.table</code>	(Unset)	Enter a name for the table that will be created in DynamoDB for S3Guard. If you leave this blank while setting <code>fs.s3a.s3guard.ddb.table.create</code> to "true", a separate DynamoDB table will be created for each accessed bucket. For each bucket, the respective S3 bucket name being used as the DynamoDB table name.
<code>fs.s3a.s3guard.ddb.region</code>	(Unset)	Set this parameter to one of the values from AWS. Refer to Amazon documentation. The "region" column value needs to be set as this parameter value.

Base Parameter	Default Value	Setting for S3Guard
		If you leave this blank, the same region as where the S3 bucket is will be used.
<code>fs.s3a.s3guard.ddb.table.capacity.read</code>	500	Specify read capacity for DynamoDB or use the default. You can monitor the DynamoDB workload in the DynamoDB console on AWS portal and adjust the read/write capacities on the fly based on the workload requirements.
<code>fs.s3a.s3guard.ddb.table.capacity.write</code>	100	Specify write capacity for DynamoDB or use the default. You can monitor the DynamoDB workload in the DynamoDB console on AWS portal and adjust the read/write capacities on the fly based on the workload requirements.

Example

Adding the following custom properties will create a DynamoDB table called "guarded-table" in the "eu-west-1" region (where the "guarded-table" bucket is located). The configuration will be valid for a bucket called "guarded-table", which means that "guarded-table" will only be used for storing metadata related to this bucket.

```
<property>
  <name>fs.s3a.bucket.guarded-table.metastore.impl</name>
  <value>org.apache.hadoop.fs.s3a.s3guard.DynamoDBMetadataStore</value>
</property>

<property>
  <name>fs.s3a.bucket.guarded-table.s3guard.ddb.table</name>
  <value>my-table</value>
</property>

<property>
  <name>fs.s3a.bucket.guarded-table.s3guard.ddb.table.create</name>
  <value>>true</value>
</property>

<property>
  <name>fs.s3a.bucket.guarded-table.s3guard.ddb.region</name>
  <value>eu-west-1</value>
</property>
```

3.7.2.6. Create the S3Guard Table in DynamoDB

Unless the property `fs.s3a.bucket.test.s3guard.ddb.table.create` is set to enable the table to be automatically created, the table must be created on the command line using the `hadoop s3guard init` command:

```
hadoop s3guard init s3a://guarded-table/

2018-05-31 15:25:59,070 [main] INFO  s3guard.DynamoDBMetadataStore
(DynamoDBMetadataStore.java:createTable(1025)) -
    Creating non-existent DynamoDB table guarded-table in region eu-
west-1
2018-05-31 15:26:11,759 [main] INFO  s3guard.S3GuardTool (S3GuardTool.
java:initMetadataStore(270)) -
    Metadata store DynamoDBMetadataStore{region=eu-west-1, tableName=
guarded-table} is initialized.
Metadata Store Diagnostics:
ARN=arn:aws:dynamodb:eu-west-1:980678866538:table/guarded-table
```



```

description=S3Guard metadata store in DynamoDB
name=guarded-table
read-capacity=500
region=eu-west-1
retryPolicy=ExponentialBackoffRetry(maxRetries=9, sleepTime=100 MILLISECONDS)
size=0
status=ACTIVE
table={AttributeDefinitions: [{AttributeName: child,AttributeType: S},
  {AttributeName: parent,AttributeType: S}],
  TableName: guarded-table,
  KeySchema: [{AttributeName: parent,KeyType: HASH}, {AttributeName: child,
  KeyType: RANGE}],
  TableStatus: ACTIVE,
  CreationDateTime: Thu May 31 15:25:59 BST 2018,
  ProvisionedThroughput: {NumberOfDecreasesToday: 0,ReadCapacityUnits: 500,
  WriteCapacityUnits: 100},
  TableSizeBytes: 0,
  ItemCount: 0,
  TableArn: arn:aws:dynamodb:eu-west-1:980678866538:table/guarded-table,
  TableId: fb50922a-90bf-4df2-a3c3-9f4aa6914d56,}
write-capacity=100

```

Once created, the table may be used immediately.

3.7.3. Monitoring and Maintaining S3Guard

The DynamoDB console on AWS allows you to monitor the workload on DynamoDB.

If you are not using auto scaling, the AWS console allows you to adjust the read/write capacities of the table. The command `hadoop s3guard set-capacity` can also be used to alter these values.

3.7.4. Disabling S3Guard and Destroying a S3Guard Database

If you decide to disable S3Guard:

1. From the Ambari web UI, delete the per-bucket `fs.s3a.metastore.impl` parameter or set it back to the default "org.apache.hadoop.fs.s3a.s3guard.NullMetadataStore".
2. If you are not using a shared table, in DynamoDB console on AWS, delete the DynamoDB table to avoid incurring unnecessary costs.

The command `hadoop s3guard destroy` can be used on the command line to destroy a table.

```

hadoop s3guard destroy s3a://guarded-table/

2018-05-31 15:35:39,075 [main] INFO  s3guard.S3GuardTool (S3GuardTool.
java:initMetadataStore(270)) -
    Metadata store DynamoDBMetadataStore{region=eu-west-1, tableName=
guarded-table} is initialized.
2018-05-31 15:35:39,077 [main] INFO  s3guard.DynamoDBMetadataStore
(DynamoDBMetadataStore.java:destroy(793)) -
    Deleting DynamoDB table guarded-table in region eu-west-1
Metadata store is deleted.

```

Destroying a table does not destroy the data in S3; it merely removes the summary data used by S3Guard to provide consistent listings.

3.7.5. Pruning Old Data from S3Guard Tables

S3Guard keeps tombstone markers of deleted files. It is good to clean these regularly, just to keep costs down. This can be done with the `hadoop s3guard prune` command. This can be used to delete entries older than a certain number of days, minutes or hours:

```
hadoop s3guard prune -days 3 -hours 6 -minutes 15 s3a://guarded-table/

2018-05-31 15:39:27,981 [main] INFO s3guard.S3GuardTool (S3GuardTool.
java:initMetadataStore(270)) -
    Metadata store DynamoDBMetadataStore{region=eu-west-1, tableName=guarded-
table} is initialized.
2018-05-31 15:39:33,770 [main] INFO s3guard.DynamoDBMetadataStore
(DynamoDBMetadataStore.java:prune(851)) -
    Finished pruning 366 items in batches of 25
```

3.7.6. Importing a Bucket into S3Guard

The `hadoop s3guard import` command can list and import a bucket's metadata into a S3Guard table. This is harmless if the contents are already imported.

```
hadoop s3guard import s3a://guarded-table/

2018-05-31 15:47:45,672 [main] INFO s3guard.S3GuardTool (S3GuardTool.
java:initMetadataStore(270)) -
    Metadata store DynamoDBMetadataStore{region=eu-west-1, tableName=guarded-
table} is initialized.
Inserted 0 items into Metadata Store
```

You do not need to issue this command after creating a table; the data is added as listings of S3 paths find new entries. It merely saves by proactively building up the database.

3.7.7. Verifying that S3Guard is Enabled on a Bucket

When S3Guard is working, apart from some messages in the logs, there is no obvious clue that it is enabled. To verify that a bucket does have S3Guard enabled, use the command line command `hadoop s3guard bucket-info`. This will print bucket information, and can be used to explicitly check that a bucket has s3guard enabled

On a guarded bucket, it will list details about the bucket, the S3Guard Database on DynamoDB, and some client information.

```
hadoop s3guard bucket-info -guarded s3a://guarded-table/

Filesystem s3a://guarded-table
Location: eu-west-1
Filesystem s3a://guarded-table is using S3Guard with store
DynamoDBMetadataStore{region=eu-west-1, tableName=guarded-table}
Authoritative S3Guard: fs.s3a.metadatastore.authoritative=false
Metadata Store Diagnostics:
ARN=arn:aws:dynamodb:eu-west-1:980678866538:table/guarded-table
description=S3Guard metadata store in DynamoDB
name=guarded-table
```

```

read-capacity=100
region=eu-west-1
retryPolicy=ExponentialBackoffRetry(maxRetries=9, sleepTime=100 MILLISECONDS)
size=61261
status=ACTIVE
table={AttributeDefinitions: [{AttributeName: child,AttributeType: S},
  {AttributeName: parent,AttributeType: S}],
  TableName: guarded-table,
  KeySchema: [{AttributeName: parent,KeyType: HASH}, {AttributeName: child,
  KeyType: RANGE}],
  TableStatus: ACTIVE
  ,CreationDateTime: Sat Apr 28 22:14:22 BST 2018,
  ProvisionedThroughput: {LastDecreaseDateTime: Thu May 31 15:09:04 BST 2018,
  NumberOfDecreasesToday: 2,ReadCapacityUnits: 100,WriteCapacityUnits: 20},
  TableSizeBytes: 61261,ItemCount: 419,
  TableArn: arn:aws:dynamodb:eu-west-1:980678866538:table/guarded-table,
  TableId: dc465257-8aaf-4a80-ad3e-c7fc708322fb,}
write-capacity=20
The "magic" committer is supported

S3A Client
Endpoint: fs.s3a.endpoint=s3.amazonaws.com
Encryption: fs.s3a.server-side-encryption-algorithm=none
Input seek policy: fs.s3a.experimental.input.fadvise=normal

```

When invoked against an unguarded bucket the same command will fail.

```

hadoop s3guard bucket-info -guarded s3a://unguarded/

Filesystem s3a://unguarded
Location: eu-west-1
Filesystem s3a://unguarded is not using S3Guard
The "magic" committer is supported

S3A Client
Endpoint: fs.s3a.endpoint=s3-eu-west-1.amazonaws.com
Encryption: fs.s3a.server-side-encryption-algorithm=none
Input seek policy: fs.s3a.experimental.input.fadvise=normal
2018-05-31 16:37:16,691 [main] INFO util.ExitUtil (ExitUtil.
java:terminate(210)) - Exiting with status 46: 46: S3Guard is not enabled for
s3a://unguarded

```

3.7.8. Using the S3Guard CLI

The S3Guard CLI offers other maintenance commands. for information on how to use them, refer to [Apache](#) documentation.

3.7.9. S3Guard: Operational Issues

The following operational issues have been identified while testing S3Guard.

S3Guard and Hive

S3Guard provides the consistent listings needed for listing data and committing output. Without S3Guard, it is not safe to use an S3 bucket as a direct destination of Hive queries.

S3Guard and MapReduce and Spark

S3Guard provides the consistent listings needed for listing data and committing output. Apache Spark and Hadoop MapReduce now support a high-performance “committer” to safely write their output to an S3 Bucket, even if S3Guard is disabled. However, without S3Guard it is not safe for one query to read the output of another query “recently” written to the S3A bucket. For this reason, even if a new “S3A Committer” is used, we recommend the use of S3Guard.

Third-party S3-compatible object stores

Third-party object stores which reimplement the AWS S3 protocol are usually “consistent”. As such, there is no need to use S3Guard. Consult the object store's supplier as to its consistency model.

S3Guard Security Aspects

The DynamoDB table needs to be writeable by all users/services using S3Guard. If a single DynamoDB table is used to store metadata about multiple buckets, then clients with access to the table will be able to read the metadata about objects in any bucket to which their read access restricted via AWS permissions.

The standard S3 Bucket and Object Access permissions do not provide any restriction on accessing the S3Guard index data. As this is only the Hadoop file status data of object name, type, size and timestamp, the actual object data and any tags attached to the object are still protected by AWS permissions. However, directory and filenames will be visible.

Limitations of S3Guard

The key limitation of S3Guard is that it only provides consistent file and directory listings. It does not address update and delete consistency of the data.

It is only consistent with respect to changes made by client applications using the S3A connector with S3Guard enabled and the same DynamoDB table. Changes which are made by other applications are only eventually consistent from the perspective of S3A clients.

Unsupported Feature: Authoritative Mode

S3Guard has an experimental option, `fs.s3a.metastore.authoritative`, which declares that the S3Guard database should be treated as the reference for all file status and directory listings, and that the S3 repository itself need not be queried. That is, the DynamoDB table moves from being a cache of file information to the “source of truth”.

This makes listing operations significantly faster, because there is no need to ever make slow “LIST” calls of the S3 store. However, it is dangerous because if S3Guard ever becomes inconsistent with the S3 store, then files may not be listed when working with the S3 bucket, resulting in incomplete/wrong data with Hive, MapReduce and Spark queries. It also requires every single application working with an S3 Bucket to use S3Guard.

We do not currently recommend using authoritative mode.

3.7.10. S3Guard: Known Issues

The following known issues have been identified while testing S3Guard.

Credentials in URLs are unsupported

S3Guard cannot be used when the AWS login credentials are in the S3 URL ([HADOOP-15422](#))

Putting AWS credentials in the URLs such as `s3a://AWSID:SECRETKEY/bucket/path()` is very insecure, because the paths are widely logged: it is very hard to keep the secrets private. Losing the keys can be expensive and expose all information to which the account has access. S3Guard does not support this authentication mechanism. Place secrets in Hadoop configuration files, or (Better) JCECKs credential files.

Error when using trailing / in some hadoop fs commands

Some `hadoop fs` operations fail when there is a trailing `/` in the path, including the `fs -mkdir` command:

```
$ hadoop fs -mkdir -p s3a://guarded-table/dir/child/

mkdir: get on s3a://guarded-table/dir/child/:
com.amazonaws.services.dynamodbv2.model.AmazonDynamoDBException:
One or more parameter values were invalid:
An AttributeValue may not contain an empty string (Service:
AmazonDynamoDBv2;
Status Code: 400; Error Code: ValidationException
```

There is a straightforward workaround: remove the trailing `/` if a command fails.

```
$ hadoop fs -mkdir -p s3a://guarded-table/dir/child
```

Fix: remove a trailing `/` if the `fs -mkdir` command fails.

The hadoop s3guard command output contains the error message "hadoop-aws.sh was not found"

This is a warning message about a file which is not found in HDP and which is not actually needed by the `s3guard` command. It is safe to ignore.

Failure handling of rename() operations

If a `rename()` operation fails partway through, including due to permissions, the S3Guard database is not reliably updated.

If this `rename` failed due to a network problem it's moot: if an application can't connect to S3, then DynamoDB will inevitably be unreachable; updates will be impossible. It can also surface if the bucket has been set up with complex permissions where not all callers have full write (including delete) access to the bucket. S3Guard, (and the S3A connector), prefers unrestricted write access to an entire R/W bucket.

3.8. Safely Writing to S3 Through the S3A Committers

3.8.1. Introducing the S3A Committers

As covered in the S3Guard section, Amazon's S3 Object Store is not a filesystem: some expected behaviors of one are missing. Specifically:

- Directory listings are only eventually consistent.
- File overwrites and deletes are only eventually consistent: readers may get old data.
- There is no rename operation; it is mimicked in the S3A client by listing a directory and copying each file underneath, one-by-one.

Because directory rename is mimicked by listing and then copying files the eventual consistency of both listing and reading fails may result in incorrect data. And, because of the copying: it is slow.

S3Guard addresses the listing inconsistency problem. However, it does not address the update consistency or performance.

The normal means by which Hadoop MapReduce and Apache Spark commit work from multiple tasks is through renaming the output. Each task attempt writes to a private task attempt directory; when the task is given permission to commit by the MapReduce Application Master or Spark Driver, this task attempt directory is renamed into the job attempt directory. When the job is ready to commit, the output of all the tasks is merged into the final output directory, again by renaming files and directories.

This is fast and safe on HDFS and similar filesystems, and on object stores with rename operations, such as Azure WASB. On S3, it is unreliable without S3Guard, and even with S3Guard, the the time to commit work to S3 is proportional to the amount of data written. An operation which may work well during development can turn out to be unusable in production.

To address this the S3A committers were developed. They allow the output of MapReduce and Spark jobs to be written directly to S3, with a time to commit the job independent of the amount of data created.

What Are the S3A Committers?

The S3A committers are three different committers which can be used to commit work directly to Map-reduce and Spark. They differ in how they deal with conflict and how they upload data to the destination bucket—but underneath they all share much of the same code.

They rely on a specific S3 feature: multipart upload of large files.

When writing a large object to S3, S3A and other S3 clients use a mechanism called “Multipart Upload”.

The caller initiates a “multipart upload request”, listing the destination path and receiving an upload ID to use in the upload operations.

The caller then uploads the data in a series of HTTP POST requests, closing with a final POST listing the blocks uploaded.

The uploaded data is not visible until that final POST request is made, at which point it is published in a single atomic operation.

This mechanism is always used by S3A whenever it writes large files; the size of each part is set to the value of `fs.s3a.multipart.size`

The S3A Committers use the same multipart upload process, but convert it into a mechanism for committing the work of tasks because of a special feature of the mechanism: The final POST request does not need to be issued by the same process or host which uploaded the data.

The output of each worker task can be uploaded to S3 as a multipart upload, but without issuing the final POST request to complete the upload. Instead, all the information needed to issue that request is saved to a cluster-wide filesystem (HDFS or potentially S3 itself)

When a job is committed, this information is loaded, and the upload completed. If a task is aborted or fails: the upload is not completed, so the output does not become visible. If the entire job fails, none of its output becomes visible.

For further reading, see:

- [HADOOP-13786](#): Add S3A committers for zero-rename commits to S3 endpoints.
- [S3A Committers: Architecture and Implementation](#).
- [A Zero-Rename Committer](#): Object-storage as a Destination for Apache Hadoop and Spark.

Why Three Committers?

Why three different committers? Work was underway addressing the S3 commit problem with the “magic committer” when Netflix donated their own committers to this work. Their directory and partitioned committers, were in production use, and was gratefully accepted.

Directory Committer: Buffers working data to the local disk, uses HDFS to propagate commit information from tasks to job committer, and manages conflict across the entire destination directory tree.

Partitioned Committer: Identical to the Directory committer except that conflict is managed on a partition-by-partition basis. This allows it to be used for in-place updates of existing datasets. It is only suitable for use with Spark.

Magic Committer: Data is written directly to S3, but “magically” retargeted at the final destination. Conflict is managed across the directory tree. It requires a consistent S3 object store, which means S3Guard is a mandatory pre-requisite.

We currently recommend use of the “Directory” committer: it is the simplest of the set, and by using HDFS to propagate data from workers to the job committer, does not directly require S3Guard – this makes it easier to set up. And, as Netflix have been using its predecessor in production, it can be considered more mature.

The rest of the documentation only covers the Directory Committer: to explore the other options, consult the Apache documentation.

3.8.2. Enabling the Directory Committer in Hadoop

For backwards compatibility, output to S3 defaults to using the original file committer.

To switch to the Directory Committer in Ambari, edit `core-site.xml` and set the property `fs.s3a.committer.name` to `directory`.

```
<property>
  <name>fs.s3a.committer.name</name>
  <value>directory</value>
</property>
```

The other values for this option are `file`, `partitioned` and `magic`; `file` is the original file output committer, which uses file and directory renames to commit output. The `partitioned` committer is a variant of the directory committer, with different conflict resolution policies. The `magic` committer is a potentially faster committer, but which should be considered less tested.

3.8.3. Configuring Directories for Intermediate Data

In addition to `fs.s3a.committer.name`, two other `core-site.xml` configuration options are used to control where intermediate is stored.

A location is in the local filesystem for buffering data

```
<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.</description>
</property>
```

These directories will store the output created by all active tasks until each task is committed; the more worker processes/spark worker threads a host can support, the more disk space will be needed. Multiple disks can be listed to help spread the load, and recover from disk failure.

A location in the cluster's HDFS filesystem to share summary data about pending uploads.

```
<property>
  <name>fs.s3a.committer.staging.tmp.path</name>
  <value>tmp/staging</value>
</property>
```

These files are generally quite small: a few kilobytes per task.

Refer to the security section for advice on securing these directories.

3.8.4. Using the Directory Committer in MapReduce

Once the property `fs.s3a.committer.name` is set, Hadoop MapReduce jobs writing to paths using the `s3a://` schema will automatically switch to the new committers.

Jobs using any other filesystem as a destination will still use the normal file committer.

3.8.5. Enabling the Directory Committer in Spark

Spark has its own internal output committer which needs to be switched to the new committer mechanism, and, when using Apache Parquet-formatted output, Spark expects the committer `Parquet` to be a subclass of `ParquetOutputCommitter`.

As a result three lines need to be added to `spark-defaults.conf` to switch to the new committers:


```
spark.hadoop.fs.s3a.committer.name directory
spark.sql.sources.commitProtocolClass org.apache.spark.internal.io.cloud.
PathOutputCommitProtocol
spark.sql.parquet.output.committer.class org.apache.spark.internal.io.cloud.
BindingParquetOutputCommitter
```

This is all that is needed. Note that the S3A Committer is only used for Spark SQL, Datasets and Dataframes; some simple examples such as the wordcount examples do not use these APIs, so do use the new committers.

Here is an example pyspark application using the committer. There is no difference between this and other applications

```
from pyspark import SparkContext, SparkConf
from pyspark.sql import HiveContext, SparkSession
from pyspark.sql.functions import *

sconf = SparkConf()
sconf.set("spark.hadoop.fs.s3a.committer.name", "directory")
sconf.set("spark.sql.sources.commitProtocolClass",
          "org.apache.spark.internal.io.cloud.PathOutputCommitProtocol")
sconf.set("spark.sql.parquet.output.committer.class",
          "org.apache.spark.internal.io.cloud.BindingParquetOutputCommitter")

sc = SparkContext(appName="s3acommitter", conf = sconf)

spark = SparkSession(sc)

sourceDF = spark.range(0, 10000)
datasets = "s3a://guarded-bucket/datasets/"

sourceDF.write.format("orc").save(datasets + "orc")
sourceDF.write.format("parquet").save(datasets + "parquet")
sourceDF.write.format("csv").save(datasets + "csv")

sc.stop()
```

3.8.6. Verifying That an S3A Committer Was Used

When working correctly, the only sign the new committers are in use is that it should be faster to use S3 as a destination of work.

There is a straightforward way to determine if a new committer was used: examine the `_SUCCESS` file created in the destination directory of a query. With the original file committer, this is a zero-byte file. The new S3A committers all write a JSON file describing the committer used, the files created and various diagnostics information.

Listing this file is enough to show whether an S3A committer was used:

```
hadoop fs -ls s3a://guarded-bucket/datasets/orc/_SUCCESS
```

If this file is of size 0, then no S3A committer was used. If the file length is greater than zero, then an S3A committer was used.

To see more details about the job commit operation, the file's contents can be printed.

```
hadoop fs -cat s3a://guarded-bucket/datasets/orc/_SUCCESS
```

3.8.7. Cleaning up After Failed Jobs

The S3A committers upload data in the tasks, completing the uploads when the job is committed.

Amazon AWS still bill you for all data held in this “pending” state. The `hadoop s3guard uploads` command can be used to list and cancel such uploads. However, it is simplest to automate cleanup with a bucket lifecycle rule.

1. Go to the AWS S3 console, <https://s3.console.aws.amazon.com/>.
2. Find the bucket you are using as a destination of work.
3. Select the “management” tab.
4. Select “add a new lifecycle rule”.
5. Create a rule “cleanup uploads” with no filter, and without any “transitions”; Configure an “Expiration” action of *Clean up incomplete multipart uploads*.
6. Select a time limit for outstanding uploads, such as 1 Day.
7. Review and confirm the lifecycle rule

You need to select a limit of how long uploads can be outstanding. For Hadoop applications, this is the maximum time that either an application can write to the same file and the maximum time which a job may take. If the timeout is shorter than either of these, then programs are likely to fail.

Once the rule is set, the cleanup is automatic.

3.8.8. Using the S3Guard Command to List and Delete Uploads

The `hadoop s3guard uploads` command can also be used to list all outstanding uploads under a path, and delete them.

```
hadoop s3guard uploads s3a://guarded-bucket/
tests3ascale/scale/hugefile_nB3gctgP34ZSpzJ5_o2AVb7kKRiicXfbkBqIflA0y.
IH7CimH100VUzohRCj3QfFstoQIdcge478ZidXu764yN0vuGI1j5kcOV3rDhsrc.
RBZ5skZ93jVCN9g2c21QgB
Total 1 uploads found.
```

All outstanding uploads can be aborted, or those older than a certain age.

```
hadoop s3guard uploads -abort -days 1 -force s3a://guarded-bucket/
Deleting: tests3ascale/scale/hugefile
_nB3gctgP34ZSpzJ5_o2AVb7kKRiicXfbkBqIflA0y.
IH7CimH100VUzohRCj3QfFstoQIdcge478ZidXu764yN0vuGI1j5kcOV3rDhsrc.
RBZ5skZ93jVCN9g2c21QgB
2018-06-28 20:58:18,504 [main] INFO s3a.S3AFileSystem
(S3AFileSystem.java:abortMultipartUpload(3266)) - Aborting
multipart upload_nB3gctgP34ZSpzJ5_o2AVb7kKRiicXfbkBqIflA0y.
IH7CimH100VUzohRCj3QfFstoQIdcge478ZidXu764yN0vuGI1j5kcOV3rDhsrc.
RBZ5skZ93jVCN9g2c21QgB to tests3ascale/scale/hugefile
Total 1 uploads deleted.
```

While bucket lifecycle is the best way to guarantee that all outstanding uploads are deleted; this command line tool is useful to verify that such cleanup is taking place, or explicitly clean up after a failure.

3.8.9. Advanced Committer Configuration

The Apache documentation covers the full set of configuration options for the committers. Here are the key options.

3.8.9.1. Enabling Speculative Execution

The S3A committers all support speculative execution

For MapReduce, enable the following properties in the job:

```
mapreduce.map.speculative true
mapreduce.reduce.speculative true
```

For Spark, set the following configuration option:

```
spark.speculation true
```

3.8.9.2. Using Unique Filenames to Avoid File Update Inconsistency

When updating existing datasets, if a new file overwrites an existing file of the same name, S3's eventual consistency on file updates means that the old data may still be returned.

To avoid this, unique filenames should be used when creating files.

The property `fs.s3a.committer.staging.unique-filenames` enables this.

```
<property>
  <name>fs.s3a.committer.staging.unique-filename</name>
  <value>true</value>
</property>
```

It is set to true by default; you only need to disable it if you explicitly want newly created files to have the same name as any predecessors.

3.8.9.3. Speeding up Job Commits by Increasing the Number of Threads

When an individual job writes many files to S3, the time to commit the job can increase.

It can be speeded up by increasing the value of `fs.s3a.committer.threads`. However, the value of `fs.s3a.connection.maximum` must be at least this size otherwise the limit on the number of parallel uploads is still limited.

```
<property>
  <name>fs.s3a.committer.threads</name>
  <value>8</value>
</property>
<property>
  <name>fs.s3a.connection.maximum</name>
  <value>15</value>
</property>
```

3.8.10. Securing the S3A Committers

Here are a few security considerations when using the S3A committers.

S3 Bucket Permissions

To use an S3A committer, the account/role must have the same AWS permissions as needed to [write to the destination bucket](#).

The multipart upload list/abort operations may be a new addition to the permissions for the active role.

When using S3Guard, the account/role must also have full read/write/delete permissions for the DynamoDB table used.

Local Filesystem Security

All the committers use the directories listed in `fs.s3a.buffer.dir` to store staged/ buffered output before uploading it to S3.

To ensure that other processes running on the same host do not have access to this data, unique paths should be used per-account.

This requirement holds for all applications working with S3 through the S3A connector.

HDFS Security

The directory and partitioned committers use HDFS to propagate commit information between workers and the job committer.

These intermediate files are saved into a job-specific directory under the path `${fs.s3a.committer.staging.tmp.path}/${user}` where `${user}` is the name of the user running the job.

The default value of `fs.s3a.committer.staging.tmp.path` is `tmp/staging`, which will be converted at run time to a path under the current user's home directory, essentially `/user/${user}/tmp/staging/${user}/`.

Provided the user's home directory has access restricted to trusted accounts, this intermediate data will be secure.

If the property `fs.s3a.committer.staging.tmp.path` is changed to a different location, then this path will need to be secured to protect pending jobs from tampering. Note: this intermediate data does not contain the output, merely the listings of all pending files and the MD5 checksums of each block.

3.8.11. The S3A Committers and Third-Party Object Stores

The S3A committers will work with any object store which implements the AWS S3 protocols.

The directory committer requires a consistent cluster filesystem to propagate commit information from the worker processes to the job committer. This is normally done in HDFS.

If the third-party object store is consistent, then it may also be used as the cluster filesystem. Set `fs.s3a.committer.staging.tmp.path` to a chosen path in the store.

3.8.12. Limitations of the S3A Committers

Custom file output formats and their committers

Output formats which implement their own committers do not automatically switch to the new committers. If such a custom committer relies on renaming files to commit output, then it will depend on S3Guard for a consistent view of the object store, and take time to commit output proportional to the amount of data and the number of files.

To determine if this is the case, find the subclass of `org.apache.hadoop.mapreduce.lib.output.FileOutputFormat` which implements the custom format, to see if it subclasses `getOutputCommitter()` to return its own committer, or has a custom subclass of `org.apache.hadoop.mapreduce.lib.output.FileOutputCommitter`.

It may be possible to migrate such a committer to support store-specific committers, as was done for Apache Parquet support in Spark. Here a subclass of Parquet's `ParquetOutputCommitter` was implemented to delegates all operations to the real committer.

MapReduce V1 API Output Format and Committers

Only the MapReduce V2 APIs under `org.apache.hadoop.mapreduce` support the new committer binding mechanism. The V1 APIs under the `org.apache.hadoop.mapred` package only bind to the file committer and subclasses. The v1 APIs date from Hadoop 1.0 and should be considered obsolete. Please migrate to the v2 APIs, not just for the new committers, but because the V2 APIs are still being actively developed and maintained.

No Hive Support

There is currently no Hive support for the S3A committers. To safely use S3 as a destination of Hive work, you must use S3Guard.

3.8.13. Troubleshooting the S3A Committers

Refer to the [Apache documentation](#) for troubleshooting the committers.

The primary issue which surfaces is actually programs not switching to the new committers. There are three common reasons for this.

1. The configuration settings to switch to the new committer are not being picked up. This is particularly common in Spark, which is harder to set up.
2. The program is using the older V1 MapReduce APIs. Fix: switch to the V2 API.
3. The output format the program uses is explicitly creating its own committer. This can only be fixed by modifying the program.

To help debug Spark's configuration, there is an option which can be set to forcibly fail the spark query if the path output committer is used, but for some reason the file committer is being returned.

```
spark.hadoop.pathoutputcommit.reject.fileoutput true
```

There is also the blunt-instrument approach of causing the original output committer to crash with an invalid configuration.

```
spark.hadoop.mapreduce.fileoutputcommitter.algorithm.version 3
```

This (invalid) option ensures that if the original file committer is used, it will raise an exception.

To enable low-level logging of the committers, set the log-level of the package `org.apache.hadoop.fs.s3a.commit` to `DEBUG`. With Log4J, this can be one in `log4j.properties`:

```
log4j.logger.org.apache.hadoop.fs.s3a.commit=DEBUG
```

3.9. 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 they don'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+NQVj9gWgZVdX1PU4KBbYMSw/gA
+hyhRXcaQ+PogOsDgHh31H1TCebQ=
```

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

3.10. S3A and Checksums (Advanced Feature)

The S3A connector can be configured to export the HTTP etag of an object as a checksum, by setting the option `fs.s3a.etag.checksum.enabled` to `true`. When unset (the default), S3A objects have no checksum.

```
$ hadoop fs -touchz s3a://hwdev-bucket/src/something.txt
$ hadoop fs -checksum s3a://hwdev-bucket/src/something.txt
s3a://hwdev-bucket/src/something.txt NONE
```

Once set, S3A objects have a checksum which is created on upload.

```
$ hadoop fs -Dfs.s3a.etag.checksum.enabled=true -checksum s3a://hwdev-bucket/
src/something.txt
s3a://hwdev-bucket/src/something.txt etag
6434316438636439386630306232303465393830303939386563663834323765
```

This checksum is not compatible with that of HDFS, so cannot be used to compare file versions when using the `-update` option on DistCp between S3 and HDFS. More specifically, unless `-skipcrccheck` is set, the DistCP operation will fail with a checksum mismatch. However, it can be used for incremental updates within and across S3A buckets.

```
$ hadoop distcp -Dfs.s3a.etag.checksum.enabled=true --update s3a://hwdev-
bucket/src s3a://hwdev-bucket/dest

$ hadoop fs -Dfs.s3a.etag.checksum.enabled=true -checksum s3a://hwdev-bucket/
dest/something.txt
s3a://hwdev-bucket/src/something.txt etag
6434316438636439386630306232303465393830303939386563663834323765
```

As the checksums match small files created as a single block, incremental updates will not copy unchanged files. For large files uploaded using multiple blocks, the checksum values may differ in which case the source file will be copied again.

3.11. 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.access.key</name>
  <description>AWS access key ID used by S3A file system. Omit for IAM role-
based or provider-based authentication.</description>
</property>

<property>
  <name>fs.s3a.secret.key</name>
  <description>AWS secret key used by S3A file system. Omit for IAM role-based
or provider-based authentication.</description>
</property>

<property>
  <name>fs.s3a.aws.credentials.provider</name>
  <description>
    Comma-separated class names of credential provider classes which implement
    com.amazonaws.auth.AWSCredentialsProvider.

    These are loaded and queried in sequence for a valid set of credentials.
    Each listed class must implement one of the following means of
    construction, which are attempted in order:
    1. a public constructor accepting java.net.URI and
    org.apache.hadoop.conf.Configuration,
    2. a public static method named getInstance that accepts no
    arguments and returns an instance of
    com.amazonaws.auth.AWSCredentialsProvider, or
    3. a public default constructor.

    Specifying org.apache.hadoop.fs.s3a.AnonymousAWSCredentialsProvider allows
    anonymous access to a publicly accessible S3 bucket without any
    credentials.

    Please note that allowing anonymous access to an S3 bucket compromises
    security and therefore is unsuitable for most use cases. It can be useful
    for accessing public data sets without requiring AWS credentials.

    If unspecified, then the default list of credential provider classes,
    queried in sequence, is:
    1. org.apache.hadoop.fs.s3a.BasicAWSCredentialsProvider: supports static
    configuration of AWS access key ID and secret access key. See also
    fs.s3a.access.key and fs.s3a.secret.key.
    2. com.amazonaws.auth.EnvironmentVariableCredentialsProvider: supports
    configuration of AWS access key ID and secret access key in
    environment variables named AWS_ACCESS_KEY_ID and
    AWS_SECRET_ACCESS_KEY, as documented in the AWS SDK.
    3. com.amazonaws.auth.InstanceProfileCredentialsProvider: supports use
    of instance profile credentials if running in an EC2 VM.
  </description>
</property>

<property>
  <name>fs.s3a.session.token</name>
  <description>Session token, when using org.apache.hadoop.fs.s3a.
TemporaryAWSCredentialsProvider
as one of the providers.
```



```
</description>
</property>

<property>
  <name>fs.s3a.security.credential.provider.path</name>
  <value/>
  <description>
    Optional comma separated list of credential providers, a list
    which is prepended to that set in hadoop.security.credential.provider.path
  </description>
</property>

<property>
  <name>fs.s3a.assumed.role.arn</name>
  <value/>
  <description>
    AWS ARN for the role to be assumed.
    Required if the fs.s3a.aws.credentials.provider contains
    org.apache.hadoop.fs.s3a.AssumedRoleCredentialProvider
  </description>
</property>

<property>
  <name>fs.s3a.assumed.role.session.name</name>
  <value/>
  <description>
    Session name for the assumed role, must be valid characters according to
    the AWS APIs.
    Only used if AssumedRoleCredentialProvider is the AWS credential provider.
    If not set, one is generated from the current Hadoop/Kerberos username.
  </description>
</property>

<property>
  <name>fs.s3a.assumed.role.policy</name>
  <value/>
  <description>
    JSON policy to apply to the role.
    Only used if AssumedRoleCredentialProvider is the AWS credential provider.
  </description>
</property>

<property>
  <name>fs.s3a.assumed.role.session.duration</name>
  <value>30m</value>
  <description>
    Duration of assumed roles before a refresh is attempted.
    Only used if AssumedRoleCredentialProvider is the AWS credential provider.
    Range: 15m to 1h
  </description>
</property>

<property>
  <name>fs.s3a.assumed.role.sts.endpoint</name>
  <value/>
  <description>
    AWS Simple Token Service Endpoint. If unset, uses the default endpoint.
    Only used if AssumedRoleCredentialProvider is the AWS credential provider.
  </description>
</property>
```

```
<property>
  <name>fs.s3a.assumed.role.credentials.provider</name>
  <value>org.apache.hadoop.fs.s3a.SimpleAWSCredentialsProvider</value>
  <description>
    List of credential providers to authenticate with the STS endpoint and
    retrieve short-lived role credentials.
    Only used if AssumedRoleCredentialProvider is the AWS credential provider.
    If unset, uses "org.apache.hadoop.fs.s3a.SimpleAWSCredentialsProvider".
  </description>
</property>

<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.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.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>The total number of threads available in the filesystem for
data
  uploads *or any other queued filesystem operation*.
</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>The number of operations which can be queued for execution</
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. The corresponding purge age is defined in
  fs.s3a.multipart.purge.age.
  If set, when the filesystem is instantiated then all outstanding uploads
  older than the purge age will be terminated -across the entire bucket.
  This will impact multipart uploads by other applications and users. so
  should
  be used sparingly, with an age value chosen to stop failed uploads,
  without
  breaking ongoing operations.
  </description>
</property>

<property>
  <name>fs.s3a.multipart.purge.age</name>
  <value>86400</value>
  <description>Minimum age in seconds of multipart uploads to purge
  on startup if "fs.s3a.multipart.purge" is true
  </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. It supports the following values: 'AES256' (for SSE-S3),
  'SSE-KMS' and 'SSE-C'.
  </description>
</property>

<property>
  <name>fs.s3a.server-side-encryption.key</name>
  <description>Specific encryption key to use if fs.s3a.server-side-
  encryption-algorithm
  has been set to 'SSE-KMS' or 'SSE-C'. In the case of SSE-C, the value of
  this property
  should be the Base64 encoded key. If you are using SSE-KMS and leave this
  property empty,
  you'll be using your default's S3 KMS key, otherwise you should set this
  property to
  the specific KMS key id.
  </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.block.size</name>
  <value>32M</value>
```

```
<description>Block size to use when reading files using s3a: file system.
  A suffix from the set {K,M,G,T,P} may be used to scale the numeric value.
</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.
  </description>
</property>

<property>
  <name>fs.s3a.fast.upload.buffer</name>
  <value>disk</value>
  <description>
    The buffering mechanism to for data being written.
    Values: disk, array, bytebuffer.

    "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.
  </description>
</property>

<property>
  <name>fs.s3a.fast.upload.active.blocks</name>
  <value>4</value>
  <description>
    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.
  </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.
    A suffix from the set {K,M,G,T,P} may be used to scale the numeric value.
  </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.metastore.authoritative</name>
  <value>false</value>
  <description>
    When true, allow MetadataStore implementations to act as source of
    truth for getting file status and directory listings. Even if this
    is set to true, MetadataStore implementations may choose not to
    return authoritative results. If the configured MetadataStore does
    not support being authoritative, this setting will have no effect.
  </description>
</property>

<property>
  <name>fs.s3a.metastore.impl</name>
  <value>org.apache.hadoop.fs.s3a.s3guard.NullMetadataStore</value>
  <description>
    Fully-qualified name of the class that implements the MetadataStore
    to be used by s3a. The default class, NullMetadataStore, has no
    effect: s3a will continue to treat the backing S3 service as the one
    and only source of truth for file and directory metadata.
  </description>
</property>

<property>
  <name>fs.s3a.s3guard.cli.prune.age</name>
  <value>86400000</value>
  <description>
    Default age (in milliseconds) after which to prune metadata from the
    metastore when the prune command is run. Can be overridden on the
    command-line.
  </description>
</property>

<property>
  <name>fs.s3a.impl</name>
  <value>org.apache.hadoop.fs.s3a.S3AFileSystem</value>
  <description>The implementation class of the S3A Filesystem</description>
```

```
</property>

<property>
  <name>fs.s3a.s3guard.ddb.region</name>
  <value></value>
  <description>
    AWS DynamoDB region to connect to. An up-to-date list is
    provided in the AWS Documentation: regions and endpoints. Without this
    property, the S3Guard will operate table in the associated S3 bucket
    region.
  </description>
</property>

<property>
  <name>fs.s3a.s3guard.ddb.table</name>
  <value></value>
  <description>
    The DynamoDB table name to operate. Without this property, the respective
    S3 bucket name will be used.
  </description>
</property>

<property>
  <name>fs.s3a.s3guard.ddb.table.create</name>
  <value>false</value>
  <description>
    If true, the S3A client will create the table if it does not already
    exist.
  </description>
</property>

<property>
  <name>fs.s3a.s3guard.ddb.table.capacity.read</name>
  <value>500</value>
  <description>
    Provisioned throughput requirements for read operations in terms of
    capacity
    units for the DynamoDB table. This config value will only be used when
    creating a new DynamoDB table, though later you can manually provision by
    increasing or decreasing read capacity as needed for existing tables.
    See DynamoDB documents for more information.
  </description>
</property>

<property>
  <name>fs.s3a.s3guard.ddb.table.capacity.write</name>
  <value>100</value>
  <description>
    Provisioned throughput requirements for write operations in terms of
    capacity units for the DynamoDB table. Refer to related config
    fs.s3a.s3guard.ddb.table.capacity.read before usage.
  </description>
</property>

<property>
  <name>fs.s3a.s3guard.ddb.max.retries</name>
  <value>9</value>
  <description>
    Max retries on batched DynamoDB operations before giving up and
    throwing an IOException. Each retry is delayed with an exponential
```



```
    backoff timer which starts at 100 milliseconds and approximately
    doubles each time. The minimum wait before throwing an exception is
     $\text{sum}(100, 200, 400, 800, \dots, 100 \cdot 2^{N-1}) = 100 \cdot ((2^N) - 1)$ 
    So  $N = 9$  yields at least 51.1 seconds (51,100) milliseconds of blocking
    before throwing an IOException.
  </description>
</property>

<property>
  <name>fs.s3a.s3guard.ddb.background.sleep</name>
  <value>25</value>
  <description>
    Length (in milliseconds) of pause between each batch of deletes when
    pruning metadata. Prevents prune operations (which can typically be low
    priority background operations) from overly interfering with other I/O
    operations.
  </description>
</property>

<property>
  <name>fs.s3a.retry.limit</name>
  <value>${fs.s3a.attempts.maximum}</value>
  <description>
    Number of times to retry any repeatable S3 client request on failure,
    excluding throttling requests.
  </description>
</property>

<property>
  <name>fs.s3a.retry.interval</name>
  <value>500ms</value>
  <description>
    Interval between attempts to retry operations for any reason other
    than S3 throttle errors.
  </description>
</property>

<property>
  <name>fs.s3a.retry.throttle.limit</name>
  <value>${fs.s3a.attempts.maximum}</value>
  <description>
    Number of times to retry any throttled request.
  </description>
</property>

<property>
  <name>fs.s3a.retry.throttle.interval</name>
  <value>1000ms</value>
  <description>
    Interval between retry attempts on throttled requests.
  </description>
</property>

<property>
  <name>fs.s3a.committer.name</name>
  <value>file</value>
  <description>
    Committer to create for output to S3A, one of:
    "file", "directory", "partitioned", "magic".
  </description>
</property>
```

```
</property>

<property>
  <name>fs.s3a.committer.magic.enabled</name>
  <value>>false</value>
  <description>
    Enable support in the filesystem for the S3 "Magic" committer.
    When working with AWS S3, S3Guard must be enabled for the destination
    bucket, as consistent metadata listings are required.
  </description>
</property>

<property>
  <name>fs.s3a.committer.threads</name>
  <value>8</value>
  <description>
    Number of threads in committers for parallel operations on files
    (upload, commit, abort, delete...)
  </description>
</property>

<property>
  <name>fs.s3a.committer.staging.tmp.path</name>
  <value>tmp/staging</value>
  <description>
    Path in the cluster filesystem for temporary data.
    This is for HDFS, not the local filesystem.
    It is only for the summary data of each file, not the actual
    data being committed.
    Using an unqualified path guarantees that the full path will be
    generated relative to the home directory of the user creating the job,
    hence private (assuming home directory permissions are secure).
  </description>
</property>

<property>
  <name>fs.s3a.committer.staging.unique-filenames</name>
  <value>>true</value>
  <description>
    Option for final files to have a unique name through job attempt info,
    or the value of fs.s3a.committer.staging.uuid
    When writing data with the "append" conflict option, this guarantees
    that new data will not overwrite any existing data.
  </description>
</property>

<property>
  <name>fs.s3a.committer.staging.conflict-mode</name>
  <value>fail</value>
  <description>
    Staging committer conflict resolution policy.
    Supported: "fail", "append", "replace".
  </description>
</property>

<property>
  <name>fs.s3a.committer.staging.abort.pending.uploads</name>
  <value>>true</value>
  <description>
    Should the staging committers abort all pending uploads to the destination
```

```
directory?

Changing this if more than one partitioned committer is
writing to the same destination tree simultaneously; otherwise
the first job to complete will cancel all outstanding uploads from the
others. However, it may lead to leaked outstanding uploads from failed
tasks. If disabled, configure the bucket lifecycle to remove uploads
after a time period, and/or set up a workflow to explicitly delete
entries. Otherwise there is a risk that uncommitted uploads may run up
bills.
</description>
</property>

<property>
  <name>fs.s3a.list.version</name>
  <value>2</value>
  <description>
    Select which version of the S3 SDK's List Objects API to use. Currently
    support 2 (default) and 1 (older API).
  </description>
</property>

<property>
  <name>fs.s3a.etag.checksum.enabled</name>
  <value>>false</value>
  <description>
    Should calls to getFileChecksum() return the etag value of the remote
    object.
    WARNING: if enabled, distcp operations between HDFS and S3 will fail
    unless
    -skipcrccheck is set.
  </description>
</property>
```

3.12. 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 Encryption \[71\]](#)

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

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

3.12.1.1. 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.



Note

When encrypted files are renamed, they are de-encrypted and then re-encrypted with the encryption settings algorithm of the client application performing the rename. *If a client with encryption disabled renames an encrypted file, the new file will be unencrypted.*

3.12.2. 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.

3.12.2.1. 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 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 the AWS documentation.

3.12.2.2. IAM Role permissions for working with SSE-KMS

All IAM roles which need to read data encrypted with SSE-KMS must have the permissions to decrypt using the specific key the data was encrypted with:

```
kms:Decrypt
```

All IAM roles which need to both read and write data need the encrypt and decrypt permissions (that is: encrypt-only permission is not supported).

```
kms:Decrypt
kms:GenerateDatakey
```

If a role does not have the permissions to read data, it will fail with an `java.nio.AccessDeniedException`. Note: renaming files requires the permission to decrypt the data, as it is decrypted and then reencrypted as it is copied. See [AWS KMS API Permissions: Actions and Resources Reference](#) for more details on KMS permissions.

3.12.3. 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.

3.12.3.1. 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>RG8gbm90IGV2ZXIgbG9nIHRoaXMga2V5IG9yIG90aGVyd21zZSBzaGFyZSBpdA==</
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.

3.12.4. 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.

3.12.5. 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. It guarantees that when a file is renamed, it will be re-encrypted, even if the client does not explicitly request encryption.
4. 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, or decrypting data when renaming files.

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/AES256 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.

3.12.6. 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.

3.13. Improving Performance for S3A

This section includes tips for improving performance when working with data stored in Amazon S3

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 \[58\]](#)
- [Configuring and Tuning S3A Block Upload \[58\]](#)

3.13.1. 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.

3.13.2. Configuring and Tuning S3A Block Upload

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 $O(\text{data}/\text{bandwidth})$. 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 store, 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.

3.13.2.1. Tuning S3A Uploads

When data is written to S3, it is buffered locally and then uploaded in multi-Megabyte blocks, with the final upload taking place as the file is closed.

The following major configuration options are available for the S3A block upload options. These are used whenever data is written to S3.

Table 3.3. S3A Fast Upload Configuration Options

Parameter	Default Value	Description
<code>fs.s3a.multipart.size</code>	100M	Defines the size (in bytes) of the blocks 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.
<code>fs.s3a.fast.upload.active.blocks</code>	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.
<code>fs.s3a.buffer.dir</code>	Empty value	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 <code>hadoop.tmp.dir</code> is used.
<code>fs.s3a.fast.upload.buffer</code>	disk	<p>The <code>fs.s3a.fast.upload.buffer</code> determines the buffering mechanism to use when uploading data.</p> <p>Allowed values are: disk, array, bytearray:</p> <ul style="list-style-type: none"> • (default) "disk" will use the directories listed in <code>fs.s3a.buffer.dir</code> 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. <p>Both "array" and "bytebuffer" will consume memory in a single stream up to the number of blocks set by: <code>fs.s3a.multipart.size * fs.s3a.fast.upload.active.blocks</code>. If using either of these mechanisms, keep this value low.</p> <p>The total number of threads performing work across all threads is set by <code>fs.s3a.threads.max</code>, with <code>fs.s3a.max.total.tasks</code> values setting the number of queued work items.</p>

Note that:

- If the amount of data written to a stream is below that set in `fs.s3a.multipart.size`, the upload takes place after the application has written all its data.

- The maximum size of a single file in S3 is one thousand blocks, which, for uploads means `10000 * fs.s3a.multipart.size`. Too A small value of `fs.s3a.multipart.size` can limit the maximum size of files.
- 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.

Buffering uploads to disk or RAMs

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.

Buffering uploads in 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).

Buffering Uploads 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. 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).

3.13.2.2. Thread Tuning for S3A Data 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:

Table 3.4. S3A Upload Tuning Options

Parameter	Default Value	Description
<code>fs.s3a.fast.upload.active.blocks</code>	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.
<code>fs.s3a.threads.max</code>	10	The total number of threads available in the filesystem for data uploads or any other queued filesystem operation.
<code>fs.s3a.max.total.tasks</code>	5	The number of operations which can be queued for execution
<code>fs.s3a.threads.Keepalivetime</code>	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.

3.13.3. Optimizing S3A read performance for different file types

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

Policy	Description
"normal"	This starts off as "sequential": it asks for the whole file. As soon as the application tries to seek backwards in the

Policy	Description
	file it switches into "random" IO mode. This is not quite as efficient for Random IO as the "random" mode, because that first read may have to be aborted. However, because it is adaptive, it is the best choice if you do not know the data formats which will be read.
"sequential" (default)	<p>Read through the file, possibly with some short forward seeks.</p> <p>The whole document is requested in a single HTTP request; forward seeks within the readahead range are supported by skipping over the intermediate data.</p> <p>This leads to maximum read throughput, but with very expensive backward seeks.</p>
"random"	<p>Optimized for random IO, specifically the Hadoop <code>PositionedReadable</code> operations — though <code>seek(offset); read(byte_buffer)</code> also benefits.</p> <p>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 <code>read</code> operation - rounded up to the readahead value set in <code>setReadahead()</code> if necessary.</p> <p>By reducing the cost of closing existing HTTP requests, this is highly efficient for file IO accessing a binary file through a series of <code>PositionedReadable.read()</code> and <code>PositionedReadable.readFully()</code> calls. Sequential reading of a file is expensive, as now many HTTP requests must be made to read through the file.</p>

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 `fadvice` policy in the configuration option `fs.s3a.experimental.input.fadvice` 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.fadvice</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.

3.13.4. 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.

3.13.5. S3 Performance Checklist

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.
- [] When copying data using DistCp, use the following [performance optimizations](#).
- [] If planning to use Hive with S3, review *Improving Hive Performance with Cloud Object Stores*.
- [] If planning to use Spark with S3, review *Improving Spark Performance with Cloud Object Stores*.

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](#).

3.14. Working with Third-party S3-compatible Object Stores

The S3A Connector can work with third-party object stores; some vendors test the connector against their stores—and even actively collaborate in developing the connector in the open source community.

Option	Change to
<code>fs.s3a.endpoint</code>	(the hostname of the S3 Store)
<code>fs.s3a.path.style.access</code>	<code>true</code>
<code>fs.s3a.signing-algorithm</code>	If the default signing mechanism is rejected, another mechanism may work from: "QueryStringSignerType", "S3SignerType", "AWS3SignerType", "AWS4SignerType", "AWS4UnsignedPayloadSignerType" and "NoOpSignerType".
<code>fs.s3a.connection.ssl.enabled</code>	Set to "false" if HTTP is used instead of HTTPS
<code>fs.s3a.multiobjectdelete.enable</code>	Set to "false" if bulk delete operations are not supported.
<code>fs.s3a.list.version</code>	Set to "1" if the list directories with the default "2" option fails with an error.

Third-party object stores are generally *consistent*; there is no need for S3Guard. The S3A Committers will still offer better performance, and should be used for MapReduce and Spark.

Encryption may or may not be supported: consult the documentation.

Security permissions are likely to be implemented differently from the IAM role mode—again, consult the documentation to see what is available.

3.15. Troubleshooting S3

It can be a bit troublesome to get the S3A connector to work, with classpath and authentication being the usual troublespots.

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

- [Authentication Failures \[65\]](#)
- [Classpath Related Errors \[67\]](#)
- [Connectivity Problems \[67\]](#)
- [Errors During Delete or Rename of Files \[70\]](#)
- [Errors Related to Visible S3 Inconsistency \[71\]](#)
- [Troubleshooting Encryption \[71\]](#)

3.15.1. Authentication Failures

You may encounter the following S3 authentication issues.

3.15.1.1. 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.

3.15.1.2. 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.

3.15.1.3. 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 "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.

3.15.1.4. 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.

3.15.2. 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`.

3.15.2.1. ClassNotFoundException Errors

```
ClassNotFoundException: org.apache.hadoop.fs.s3a.S3AFileSystem /code>
```

This means that the `hadoop-aws` 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.

3.15.2.2. 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.

3.15.3. Connectivity Problems

You may encounter the following S3 connectivity issues.

3.15.3.1. 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.

3.15.3.2. 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](#).

3.15.3.3. 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 using the 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>
```

3.15.4. 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.

3.15.5. Errors Related to Visible S3 Inconsistency

Amazon S3 is an *eventually consistent object store*.

It may take time for a newly created object to appear in directory listings, while deleted objects may still be visible.

After an object is overwritten, attempting to read the new data may still return the old data.

After an object is deleted, it may still be visible in directory, listings; attempting to open the file may return the deleted data.

The directory inconsistency is precisely the problem which [S3Guard](#) aims to correct.

Inconsistent directory listings can surface as a `FileNotFoundException` during a file or directory rename, including when the output of a Hive, MapReduce or Spark job is renamed into its final destination.

- Enable S3Guard on any bucket used for the output of Hive, Spark or MR jobs.
- Use an S3A committer to safely and efficiently commit the output of MR and Spark jobs.
- When writing new data into an existing location, give the new files different names.

3.15.6. Troubleshooting Encryption

Refer to this section when troubleshooting S3 Encryption

3.15.6.1. 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)
```

3.15.6.2. 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.

```
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)
```

3.15.6.3. 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.getSSEEncryptionAlgorithm(S3AUtils.java:760)
```

Make sure to enter the correct algorithm name.

4. Working with ADLS

Azure Data Lake Store (ADLS) is a cloud object store 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.

4.1. Configuring Access to 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. Configure Access by 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.

Prerequisites

In one of the steps, you will be required to assign the **Owner** role to your application. If you do not have sufficient permissions, the role assignment step may have to be performed by your Azure admin.

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 <code>fs.adl.oauth2.client.id</code>
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 <code>fs.adl.oauth2.credential</code>
Token endpoint	You can obtain this from the App Registrations>Endpoints page by copying the <code>OAUTH 2.0 TOKEN ENDPOINT</code> value. This will be your <code>fs.adl.oauth2.refresh.url</code>

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. Configure Access by 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>
```

Set 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](#).

4.2. 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](#)

4.3. Referencing ADLS in 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
```

4.4. 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.

4.5. ADLS Proxy Setup

The ADLS connector uses the JVM proxy settings to control its proxy setup.

The [Oracle Java documentation](#) covers the options to set.

As the ADLS connector uses HTTPS, the `https.proxy` options are those which must be configured. See **DistCp and Proxy settings** for the specifics of how to set these proxy settings in DistCp (or indeed, any MapReduce job).

5. Working 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).

5.1. Configuring Access to 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](#).

5.2. 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](#)

5.3. 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>
```

5.4. Referencing WASB in 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
```

5.5. 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.

5.6. 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>
```

5.7. 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.

5.8. 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.

5.9. 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 provide required SAS keys. The following property can be 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" : relevant message on failure <String>,
  "sasKey" : Requested SAS Key <String>
}
```

5.10. 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. Working with Google Cloud Storage

HDP 2.6.5 and newer allows you to configure access from a cluster to Google Cloud Storage (GCS) in order to access cloud data.

6.1. Configuring Access to Google Cloud Storage

Access from a cluster to a Google Cloud Storage is possible through a [service account](#). Configuring access to Google Cloud Storage involves the following steps.

Table 6.1. Overview of Configuring Access to Google Cloud Storage

Step	Considerations
Creating a service account on Google Cloud Platform and generating a key associated with it.	<ul style="list-style-type: none"> You may need to contact your Google Cloud Platform admin in order to complete these steps. If you already have a service account, you do not need to perform these steps as long as you are able to provide the service account key. If you have a service account but do not know the service account key, you should be able to generate a new key.
Creating a role on Google Cloud Platform with sufficient permissions to access storage buckets.	<ul style="list-style-type: none"> You may need to contact your Google Cloud Platform admin in order to complete these steps. This is a one time operation, and the same role can be used across different service accounts and storage buckets.
Modifying permissions of the Google Cloud Storage bucket so that you can access it by using your service account key.	<ul style="list-style-type: none"> You may need to contact your Google Cloud Platform admin in order to complete these steps. You should perform these steps for each bucket that you want to access. You do not need to perform these steps if your service account has project-wide access to all buckets on the account.
Configuring credentials via Ambari.	<ul style="list-style-type: none"> These configuration steps are appropriate for a single-user cluster. Only one configuration per cluster is recommended; that is, you should use one service account per cluster. If required, it is possible to use multiple service account with the same cluster; In this case, each job-specific configuration should be changed to use the desired service account.

6.1.1. Create a GCP Service Account

You must create a Google Cloud Platform service account and generate an access key (in the JSON format). If you are using a corporate GCP account, it is likely that only your GCP admin can perform these steps. Example steps are described below.

These steps assume that you have a Google Cloud Platform account. If you do not have one, you can create it at <https://console.cloud.google.com>.

Steps

1. In the Google Cloud Platform web console, navigate to **IAM & admin Service accounts:**

2. Click **+Create Service Account**.
3. Provide the following information:
 - Under **Service account name**, provide some name for your service account.
 - Under **Role**, select the project-level roles that the account should have.
 - Check **Furnish a new private key** and select **JSON**.

Create service account

Service account name [?]
domi-test

Role [?]
Compute Image ...

Service account ID
domi-test @siq-haas.iam.gserviceaccount.com

Furnish a new private key
Downloads a file that contains the private key. Store the file securely because this key can't be recovered if lost.

Key type

JSON
Recommended

P12
For backward compatibility with code using the P12 format

Enable G Suite Domain-wide Delegation
Allows this service account to be authorized to access all users' data on a G Suite domain without manual authorization on their part. [Learn more](#)

CANCEL CREATE

4. Click **Create** and the file containing the key will be downloaded onto your machine. The name of the key file is usually long and contains spaces, so you may want to rename the file.

Later you will need to place this key on your cluster nodes.

6.1.2. Create a Custom Role

Specific permissions are required for the Google Cloud Storage Connector to access buckets. This set of permissions is the combination of the permissions associated with the existing [Google Cloud IAM Role](#) called "Storage Object Admin" and the [Google Cloud IAM Permission](#) called "storage.buckets.get".

Steps

1. In the Google Cloud Platform web console, navigate to **IAM & admin > Roles**
2. Click on **+Create Role**.
3. Provide the following:

- Enter some title under **Title**
 - Enter some ID under **ID**
 - Under **Role launch stage** select "General Availability"
4. Click on **Add Permissions** and add the following permissions:
- storage.bucket.get
 - storage.objects.create
 - storage.objects.delete
 - storage.objects.get
 - storage.objects.getIamPolicy
 - storage.objects.list
 - storage.objects.setIamPolicy
 - storage.objects.update

The screenshot shows the IAM & admin console with the 'Create Role' page. The role name is 'domitestgcp' and the launch stage is 'General Availability'. An 'Add permissions' dialog box is open, showing a list of permissions to be added to the role. The permissions are:

Permission	Status
<input checked="" type="checkbox"/> storage.buckets.get	Supported
<input type="checkbox"/> storage.buckets.getIamPolicy	Supported
<input checked="" type="checkbox"/> storage.objects.create	Supported
<input checked="" type="checkbox"/> storage.objects.delete	Supported
<input checked="" type="checkbox"/> storage.objects.get	Supported
<input checked="" type="checkbox"/> storage.objects.getIamPolicy	Supported
<input checked="" type="checkbox"/> storage.objects.list	Supported
<input checked="" type="checkbox"/> storage.objects.setIamPolicy	Supported
<input checked="" type="checkbox"/> storage.objects.update	Supported


5. Once done adding permissions, click on **Create**.

After performing these steps, a new role will be created.

6.1.3. Modify GCS Bucket Permissions

You or your GCP admin must set the bucket permissions so that your service account has access to the bucket that you want to access from the cluster. The IAM role created in the previous step, is the minimum role required to access the cluster. Example steps are described below.

Steps

1. In the Google Cloud Platform web console, navigate to **Storage > Browser**.
2. Find the bucket for which you want to edit permissions.
3. Click the  and select **Edit bucket permissions**:
4. In the **Permissions** tab set the bucket-level permissions:
 - Click on **Add members** and enter the service account created earlier.
 - Under **Roles**, select the IAM role created in the previous step. The role should be available under "Custom".
 - When done, click **Add**.

PERMISSIONS LABELS

To grant access to your bucket, add members and assign Identity and Access Management (IAM) roles to specify their level of access. Multiple roles allowed.

You can no longer set ACLs in the console to manage access. To learn how IAM and ACLs are related, see the [documentation](#).

Add members ?

domi-test@siq-haas.iam.gserviceaccount.com Select a role Add

Search members

Filter by name domitestgcp

Cloud Build Service Account (1 member)
Can perform builds

Security Reviewer (1 member)
Security reviewer role, with permission

Storage Admin (16 members)
Full control of GCS resources.

Cloud Build ▶

Custom ▶

Dataflow ▶

Dataproc ▶

IAM ▶

Storage ▶

Storage Legacy ▶

[Manage roles](#)

After performing these steps, the bucket-level permissions will be updated.

6.1.4. Configure Access to GCS from Your Cluster

After obtaining the service account key, perform these steps on your cluster. The steps below assume that your service account key is called `google-access-key.json`. If you chose a different name, make sure to update the commands accordingly.

Steps

1. In the Ambari web UI, set the following three properties under **custom-core-site**. To set these properties in the **custom-core-site**, navigate to **HDFS > Configs > Custom core-site** and click **Add Property**.

- Set the following properties:

```
fs.gs.auth.service.account.email=<VALUE1>
fs.gs.auth.service.account.private.key.id=<VALUE2>
fs.gs.auth.service.account.private.key=<VALUE3>
```

You can obtain the values for these parameters as follows:

Parameter	Value
fs.gs.auth.service.account.email	The <code>client_email</code> field extracted from the credential's JSON file.
fs.gs.auth.service.account.private.key.id	The <code>private_key_id</code> field extracted from the credential's JSON file.
fs.gs.auth.service.account.private.key	The <code>private_key</code> field extracted from the credential's JSON file.

The JSON key file downloaded in the previous step is in plain text. Open the file in your favorite text editor to extract the relevant values needed in the above configuration.



Note

For enhanced security, these values can also be configured using [Hadoop CredentialProvider](#).

- **Configure the following properties if they're not already set by Ambari**

```
fs.gs.working.dir=/
fs.gs.path.encoding=uri-path
```



Note

Setting `fs.gs.working.dir` configures the initial working directory of a GHFS instance. This should always be set to `"/`.

Setting `fs.gs.path.encoding` sets the path encoding to be used, and allows for spaces in the filename. This should always be set to `"uri-path"`.

2. Save the configuration change and restart affected services. Additionally - depending on what services you are using - you must restart other services that access cloud storage such as Spark Thrift Server, HiveServer2, and Hive Metastore; These will not be listed as affected by Ambari, but require a restart to pick up the configuration changes.
3. Test access to the Google Cloud Storage bucket by running a few commands from any cluster node. For example, you can use the command listed below (replace `"mytestbucket"` with the name of your bucket):

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

After performing these steps, you should be able to start working with the Google Cloud Storage bucket(s).

6.2. Additional Configuration Options for GCS

You may optionally set the following properties related to Google Cloud Storage:

Table 6.2. Optional Properties Related to Google Cloud Storage

Property	Description
<code>fs.gs.project.id</code>	Allows you to enter Google Cloud's Project ID with access to configured GCS buckets. By default, this option is unset.
<code>fs.gs.block.size</code>	This does not have any effect on storage but allows you to change the way split sizes are computed. The value should be in bytes. We recommend that you set this to the HDFS block size on the cluster, or 134217728 for a block size of 128MB.

7. Accessing Cloud Data in Hive

Datasets stored in cloud object stores can be made available in Hive via external tables:

1. [Exposing Cloud Data as Hive Tables \[93\]](#)
2. [Populating Partition-Related Information \[94\]](#)
3. [Analyzing Tables \[95\]](#)

When using Hive with S3, we recommend that you [use S3Guard](#).

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 Object Stores, refer to [Improving Hive Performance](#).

7.1. Hive and S3: The Need for S3Guard

Hive's listing of source files, and its renaming operations, all need a consistent view of the filesystem's metadata: listings of directories are required to return a complete list of all files which exist underneath, and none which have been deleted. HDFS, Azure and ADL all meet this requirement. Amazon S3 does not directly meet the requirements of Hive.

To safely use S3 as a destination of Hive jobs, S3Guard must be enabled for the destination bucket.

It may be possible to use unguarded S3 Buckets as source of S3 data, provided the source data has been unchanged "long enough" for the object metadata to become consistent across all S3 servers. There is, however, no general consensus on what duration constitutes "long enough", especially in the presence of failures. Unchanging data is of course not at risk.

7.2. Exposing Cloud Data as Hive Tables

Datasets stored in cloud object stores 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 object store, 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.

7.3. Populating Partition-Related Information

When working with data stored in cloud object stores, 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 (default is 15). This will speed up load if you have hardware capacity.

7.4. Analyzing Tables

When working with data in cloud object stores, 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;
```

7.5. Improving Hive Performance with Cloud Object Stores

Tune the following parameters to improve Hive performance when working with Cloud Object Stores

Table 7.1. Improving General Performance

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

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

Table 7.2. Accelerating ORC Reads in Hive

Parameter	Recommended Setting
<code>hive.orc.compute.splits.num.threads</code>	<p>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.</p> <p>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 Cloud Object Stores.</p>
<code>hive.orc.splits.include.file.footer</code>	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 7.3. Accelerating ETL Jobs

Parameter	Recommended Setting
<code>hive.metastore.fshandler.threads</code>	<p>Query launches can be slightly slower if there are no stats available or when <code>hive.stats.fetch.partition.stats=false</code>. In such cases, Hive ends up looking at file sizes for every file that it tries to access.</p> <p>Tuning <code>hive.metastore.fshandler.threads</code> helps reduce the overall time taken for the metastore operation.</p>
<code>fs.trash.interval</code>	<p>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 <code>fs.trash.interval=0</code> to completely skip trash.</p>

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 renames data from a temporary folder to the final location. This move operation is actually a copy+delete action, which can be 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`.

8. Accessing Cloud Data in Spark

Datasets stored in cloud object stores can be used in Spark as if it were stored in HDFS.

All these object stores are viewed by Spark as filesystems, allowing them to be used as the source and destination 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 a Cloud Object Store 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.SparkConf
import org.apache.spark.sql.SparkSession
import org.apache.spark.sql.types.StringType
val spark = SparkSession
  .builder
  .appName("DataFrames")
  .config(sparkConf)
  .getOrCreate()
import spark.implicits._
val numRows = 1000
// generate test data
val sourceData = spark.range(0, numRows).select($"id".as("l"), $"id".
  cast(StringType).as("s"))

// define the destination
val dest = "adl://hortonworks-eu.azuredatalakestore.net/dataframes"
// write the data
val orcFile = dest + "/data.orc"
sourceData.write.format("orc").save(orcFile)

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

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

spark.stop()
```

Example 2: Spark Streaming and Cloud Storage

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


```
import org.apache.spark.SparkConf
import org.apache.spark.sql.SparkSession
import org.apache.spark.streaming._

val sparkConf = new SparkConf()
val ssc = new StreamingContext(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.

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. WASB, ADL and GCS all have faster rename operations, so do not suffer from this problem.

Related Links

[Using S3 as a Safe and Fast Destination of Work \[98\]](#)

[Improving Spark Performance with Cloud Storage \[98\]](#)

8.1. Using S3 as a Safe and Fast Destination of Work

Amazon S3 is an eventually consistent filesystem, which makes listings unreliable. It also lacks a `rename()` operation which makes the performance of committing work very slow. To address these issues, the S3A connector has two features

- S3Guard: for consistent directory listings.
- S3A Committers: For high-performance committing of the output of Spark queries to S3.

Without these, using S3 as a destination of work is slow and potentially unsafe.

8.2. Improving Spark Performance with Cloud Storage

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

- [Improving ORC and Parquet Read Performance \[99\]](#)

- [Accelerating S3 Read Performance \[99\]](#)
- [Accelerating Azure Read Performance \[100\]](#)
- [Putting it All Together: spark-defaults.conf \[100\]](#)

8.2.1. Improving ORC and Parquet Read Performance

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
```

8.2.2. Accelerating S3 Read Performance

The most effective way to read a large file from S3 is in a single HTTPS request, reading in all data from the beginning to the end. This is exactly the read pattern used when the source data is a CSV file or files compressed with GZIP.

ORC and Parquet files benefit from Random IO: they read footer data, seek backwards, and skip forwards, to minimize the amount of data read. This IO pattern is highly efficient for HDFS, but for object stores, making and breaking new HTTP connections, then this IO pattern is very very expensive.

By default, as soon as an application makes a backwards `seek()` in a file, the S3A connector switches into “random” IO mode, where instead of trying to read the entire file, only the amount configured in `fs.s3a.readahead.range` is read in. This results in an IO behavior where, at the possible expense of breaking the first HTTP connection, reading ORC/Parquet data is efficient.

See [Optimizing S3A read performance for different file types \[61\]](#).

8.2.3. Accelerating Azure Read Performance

The Azure WASB connector uses the same dynamic read policy as the S3A “normal” policy: files are opened for sequential IO, switching to random IO if the client application starts to make random IO calls.

The Azure ADL connector always reads data in blocks of the byte size set by the configuration option `adl.feature.client.cache.readahead`; the default value is 4194304, that is, four megabytes.

8.2.4. Putting it All Together: `spark-defaults.conf`

Combining the performance settings for ORC and Parquet input, produces the following set of options to set in the `spark-defaults.conf` file for Spark applications.

```
spark.hadoop.fs.s3a.experimental.input.policy random
spark.sql.orc.filterPushdown true
spark.hadoop.parquet.enable.summary-metadata false
spark.sql.parquet.mergeSchema false
spark.sql.parquet.filterPushdown true
spark.sql.hive.metastorePartitionPruning true
```

When working with S3, the S3A Directory committer should be enabled for both performance and safety:

```
spark.hadoop.fs.s3a.committer.name directory
spark.sql.parquet.output.committer.class org.apache.spark.internal.io.cloud.
BindingParquetOutputCommitter
spark.sql.sources.commitProtocolClass org.apache.spark.internal.io.cloud.
PathOutputCommitProtocol
```

9. Copying Cloud Data with Hadoop

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

9.1. Running FS Shell Commands

Many of the standard Hadoop FileSystem shell commands that interact with HDFS also can be used to interact with Cloud Object Stores. 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 Cloud Object Storage \[102\]](#)

[Unsupported Filesystem Operations \[103\]](#)

[Deleting Files on Cloud Object Stores \[104\]](#)

[Overwriting Objects on Amazon S3 \[104\]](#)

[Timestamps on Cloud Object Stores \[104\]](#)

9.1.1. Commands That May Be Slower with Cloud Object Storage

Some commands tend to be significantly slower with 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 a directory in an object store usually takes time at least as proportional to the number 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. Amazon S3's time to rename is proportional the amount of data being renamed, so the larger the files being worked on, the longer it will take. This can become a significant delay.

We recommend that when using the `hadoop fs put` and `hadoop fs copyFromLocal` commands, you set the `-doption` 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 may to be significantly slower with Object Stores, especially those which scan the entire directory tree:

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

Our recommendation is to use these sparingly, and avoid when working with buckets/containers with many millions of entries.

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

In Amazon S3, 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. For WASB, GCS and ADLS, the time to rename is proportionally simply to the number of files. If the a rename operation is interrupted, the object store may in an undefined, with some of the source files renamed, others still in their original paths. There may also be duplicate copies of the data.

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

Copy

The `hadoop fs -cp` 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 between the local computer and the object store.

As an example, this copy command will perform the copy by downloading all the data and uploading it again.

```
hadoop fs -cp \
adl://alice.azuredatalakestore.net/current \
adl://alice.azuredatalakestore.net/historical
```



Note

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

9.1.2. Unsupported Filesystem Operations

The cloud object stores do not offer support the complete set of of `hadoop fs` commands. In particular, the commands to create, delete and rename snapshots are all unsupported; the `-appendToFile` and `-truncate` likewise unavailable: the objects are immutable, once written.

Unless the filesystem supports permissions and acls as HDFS, some or all of the permissions operations will be unsupported: `-chgrp`, `-chmod`, `-chown`, `-getfacl`, `-getfattr`, `-setfacl`, and `-setfattr`.

the `-setrep` command is usually ignored, while the `-df` command returns a spurious value (8 Exabytes)

```
bin/hadoop fs -df adl://alice.azuredatalakestore.net/
Filesystem Size Used Available Use%
adl://alice.azuredatalakestore.net 9223372036854775807 0 9223372036854775807
0%
```

9.1.3. Deleting Files on Cloud Object Stores

The `hadoop fs -rm` command deletes objects and directories full of objects. If the object store is only 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 container/bucket of the store. The `rm` operation can be significantly slower, and the deleted files continue to incur storage costs.

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

- Use the `-skipTrash` option when removing files: `hadoop fs -rm -skipTrash s3a://bucket1/dataset`
- Regularly 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.

- Disable the trash mechanism by setting the core-site option `fs.trash.interval` to 0.

Skipping the trash option is of course dangerous: it exists to stop accidental destruction of data. If the object store provides its own versioning or backup mechanism, this can be used instead.

S3 only: because Amazon S3 is eventually consistent, deleted files may briefly still be visible in listing operations, or readable in operations such as `hadoop fs -cat`. With S3Guard enabled, the listings will be consistent, and the deleted objects no longer visible there.

9.1.4. Overwriting Objects on Amazon 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.

9.1.5. Timestamps on Cloud Object Stores

Timestamps of objects and directories in Cloud Object Stores 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. That is: if a "directory" is emulated in the client, it has no create time.
- The `atime` access time feature is not supported by any of the object stores found in the Apache Hadoop codebase.

Without an `atime` timestamp, it is not possible to determine if a directory has updated data by checking the timestamp.

9.2. Copying Data with DistCp

You can use DistCp to copy data between your cluster's HDFS filesystem 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 object store, 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 -skipcrccheck -numListstatusThreads 40 hdfs://source-  
folder s3a://destination-bucket
```

When copying between cloud object stores 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 adl://alice.azuredatalakestore.net/datasets /tmp/datasets
```

This downloads all files from the path in the ADLS bucket to `/tmp/datasets` in the cluster filesystem.

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

```
hadoop distcp -update -skipcrccheck adl://alice.azuredatalakestore.net/  
datasets /tmp/datasets
```

As when uploading data from HDFS to cloud storage, checksums of files are not verified, so updated files whose new length is the same as the previous length will *not* be downloaded.

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 an ADLS store, the following command could be used:

```
hadoop distcp -numListstatusThreads 40 s3a://hwdev-example-ireland/datasets  
adl://alice.azuredatalakestore.net/datasets
```

Irrespective of source and destination store locations, when copying data with DistCp, 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 object stores. The operations may also incur costs for the data downloaded.

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 -numListstatusThreads 40 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.



Note

When using DistCP to copy data in S3, even within the same bucket, the current encryption settings of the client are used. This happens irrespective of the encryption settings of the source data.

Related Links

[Using DistCp with S3 \[106\]](#)

[Using DistCp with Azure ADLS and WASB \[107\]](#)

[DistCp and Proxy Settings \[108\]](#)

[Apache DistCp documentation](#)

9.2.1. 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 between HDFS and S3 will not be performed. We do still recommend using the `-skipcrccheck` option to make clear that this is taking place, and so that if etag checksums are enabled on S3A through the property `fs.s3a.etag.checksum.enabled`, then DistCp between HDFS and S3 will *not* trigger checksum-mismatch errors.

9.2.1.1. Specifying Per-Bucket DistCp Options for S3 Buckets

If a source or destination 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 \  
  -D fs.s3a.bucket.hwdev-example-frankfurt.endpoint=s3.eu-central-1.amazonaws.com \  
  -update -skipcrccheck -numListstatusThreads 40 \  
  s3a://hwdev-example-us/datasets/set1 \  
  s3a://hwdev-example-frankfurt/datasets/ \  
  \
```

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 \  
  -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 \  
  -update -skipcrccheck -numListstatusThreads 40 \  
  s3a://hwdev-example-us/datasets/set1 s3a://hwdev-example-frankfurt/datasets/ \  
  \
```

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

9.2.2. Using DistCp with Azure ADLS and WASB

- The `-append` option is not supported.
- The `-diff` option is not supported.
- The `-atomic` option causes a rename of the temporary data, which slows down the upload. *Avoid using this option.*

- ADLS implements the same permissions model as HDFS, so some of the `-p` options work.
- WASB supports getting and setting the permissions, *but these permissions do not control access to the data*. What they can do is ensure is that permissions can be restored after a back-up.
- 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.

9.2.3. DistCp and Proxy Settings

When using DistCp to back up data from an on-site Hadoop cluster, proxy settings may need to be set so as to reach the cloud store. For most of the stores, these proxy settings are hadoop configuration options which must be set in `core-site.xml`, or as options to the DistCp command.

ADLS uses the JVM proxy settings, which need to be set in DistCp's map and reduce processes. This can be done through the `mapreduce.map.java.opts` and `mapreduce.reduce.java.opts` options respectively.

```
export DISTCP_PROXY_OPTS="-Dhttps.proxyHost=web-proxy.example.com -Dhttps.proxyPort=80"
hadoop distcp \
  -D mapreduce.map.java.opts="$DISTCP_PROXY_OPTS" \
  -D mapreduce.reduce.java.opts="$DISTCP_PROXY_OPTS" \
  -update -skipcrccheck -numListstatusThreads 40 \
  hdfs://namenode:8020/users/alice adl://backups.azuredatalakestore.net/users/
alice
```

Without these settings, even though access to ADLS may work from the command line, `distcp` access can fail with *"Error fetching access token"*—fetching OAuth2 access tokens being the first networked operation performed by the ADLS client.

9.2.4. Improving DistCp Performance

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 object store is usually the upper limit to how fast data can be copied into S3. The further the Hadoop cluster is from the store installation, or the narrower the network connection is, the longer the operation will take. Even a Hadoop cluster deployed within cloud 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 Stores \[109\]](#)
- [Accelerating File Listing \[109\]](#)
- [Controlling the Number of Mappers and Their Bandwidth \[109\]](#)

9.2.4.1. 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 40. With this setting, multiple threads will be used for listing the contents of the source folder.

9.2.4.2. Working with Local Stores

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

Maximum performance is achieved from working with Azure containers and S3 buckets in the same cloud location as any in-cloud the cluster. For example, if your cluster is in AWS 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.

9.2.4.3. 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 cloud infrastructure, increasing the number of mappers may speed up the operation, as well as increase the likelihood that some of the source data will be held on the hosts running the mappers.

Similarly, if copying to a remote 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.

The Challenge of Store Throttling

Some cloud stores (especially S3) throttle IO operations to a directory trees in their stores: the more load is placed on a directory tree, the more the caller is throttled. Request are either delayed, or actually rejected with a "throttled" error code, after which the client is expected to wait before retrying the operation.

Large distcp operations with many mappers can trigger this throttling, *so slowing down the upload*. In such a situation, reducing the bandwidth of each mapper can reduce the load the distcp operation places on the store, while still spreading the load around the cluster.

If adding more mappers and reducers to a distcp operation appears to actually slow down the upload, throttling is a possible cause: consider reducing the mapper count.