Troubleshoot Access Denied (403 Forbidden) errors in Amazon S3

Bucket policies and IAM policies

Bucket-level operations

If there is no bucket policy in place, then the bucket implicitly allows requests from any AWS Identity and Access Management (IAM) identity in the bucket-owning account. The bucket also implicitly denies requests from any other IAM identities from any other accounts, and anonymous (unsigned) requests. However, if there is no IAM user policy in place, the requester (unless they are the root user) is implicitly denied from making any requests. For more information about this evaluation logic, see Determining whether a request is denied or allowed within an account in the *IAM User Guide*.

Object-level operations

If the object is owned by the bucket-owning account, the bucket policy and IAM user policy will function in the same way for object-level operations as they do for bucket-level operations. For example, if there is no bucket policy in place, then the bucket implicitly allows object requests from any IAM identity in the bucket-owning account. The bucket also implicitly denies object requests from any other IAM identities from any other accounts, and anonymous (unsigned) requests. However, if there is no IAM user policy in place, the requester (unless they are the root user) is implicitly denied from making any object requests.

If the object is owned by an external account, then access to the object can be granted only through object access control lists (ACLs). The bucket policy and IAM user policy can still be used to deny object requests.

Therefore, to ensure that your bucket policy or IAM user policy is not causing an Access Denied (403 Forbidden) error, make sure that the following requirements are met:

- For same-account access, there must not be an explicit `Deny` statement against the requester you are trying to grant permissions to, in either the bucket policy or the IAM user policy. If you want to grant permissions by using only the bucket policy and the IAM user policy, there must be at least one explicit `Allow` statement in one of these policies.

- For cross-account access, there must not be an explicit `Deny` statement against the requester you are trying to grant permissions to, in either the bucket policy or the IAM user policy. If you want to grant cross-account permissions by using only the bucket policy and IAM user policy, then both the bucket policy and the IAM user policy of the requester must include an explicit `Allow` statement.

Note

`Allow` statements in a bucket policy apply only to objects that are owned by the same bucket-owning account. However, `Deny` statements in a bucket policy apply to all objects regardless of object ownership.

To review or edit your bucket policy

Note

To view or edit a bucket policy, you must have the `s3:GetBucketPolicy` permission.

1. Sign in to the AWS Management Console and open the Amazon S3 console at https://console.aws.amazon.com/s3/.
2. In the left navigation pane, choose **Buckets**.
3. From the **Buckets** list, choose the name of the bucket that you want to view or edit a bucket policy for.
4. Choose the **Permissions** tab.
5. Under **Bucket policy**, choose **Edit**. The **Edit bucket policy** page appears.

To review or edit your bucket policy by using the AWS Command Line Interface (AWS CLI), use the get-bucket-policy command.

Note

If you get locked out of a bucket because of an incorrect bucket policy, sign in to the AWS Management Console by using your root user credentials. To regain access to your bucket, make sure to delete the bucket policy by using your root user credentials.

Tips for checking permissions

To check whether the requester has proper permissions to perform an Amazon S3 operation, try the following:

- Identify the requester. If it’s an unsigned request, then it's an anonymous request without an IAM user policy. If it’s a request using a presigned URL, then the user policy will be the same as the one for the IAM user or role that signed the request.
- Verify that you're using the correct IAM user or role. You can verify your IAM user or role by checking the upper-right corner of the AWS Management Console or by using the aws sts get-caller-identity command.
- Check the IAM policies that are related to the IAM user or role. You can use one of the following methods:
  - Test IAM policies with the IAM policy simulator.

    - Review the different IAM policy types.
- If needed, edit your IAM user policy.
- Review the following examples of policies that explicitly deny or allow access:
    - Explicit allow IAM user policy: IAM: Allows and denies access to multiple services programmatically and in the console
    - Explicit allow bucket policy: Granting permissions to multiple accounts to upload objects or set object ACLs for public access
    - Explicit deny IAM user policy: AWS: Denies access to AWS based on the requested AWS Region
    - Explicit deny bucket policy: Require SSE-KMS for all objects written to a bucket

---

Amazon S3 ACL settings

When checking your ACL settings, first review your Object Ownership setting to check whether ACLs are enabled on the bucket. Be aware that ACL permissions can be used only to grant permissions and cannot be used to reject requests. ACLs also cannot be used to grant access to requesters that are rejected by explicit denials in bucket policies or IAM user policies.

The Object Ownership setting is set to bucket owner enforced

If the bucket owner enforced setting is enabled, then ACL settings are unlikely to cause an Access Denied (403 Forbidden) error because this setting disables all ACLs that apply to bucket and objects. Bucket owner enforced is the default (and recommended) setting for Amazon S3 buckets.

The Object Ownership setting is set to bucket owner preferred or object writer

ACL permissions are still valid with the bucket owner preferred setting or the object writer setting. There are two kinds of ACLs: bucket ACLs and object ACLs. For the differences between these two types of ACLs, see Mapping of ACL permissions and access policy permissions.

Depending on the action of the rejected request, check the ACL permissions for your bucket or the object:

- If Amazon S3 rejected a LIST, PUT object, GetBucketAcl, or PutBucketAcl request, then review the ACL permissions for your bucket.
  Note

You cannot grant GET object permissions with bucket ACL settings.
- If Amazon S3 rejected a GET request on an S3 object, or a PutObjectAcl request, then review the ACL permissions for the object.
  Important

  If the account that owns the object is different from the account that owns the bucket, then access to the object isn't controlled by the bucket policy.

Troubleshooting an Access Denied (403 Forbidden) error from a GET object request during cross-account object ownership

Review the bucket's Object Ownership settings to determine the object owner. If you have access to the object ACLs, then you can also check the object owner's account. (To view the object owner's account, review the object ACL setting in the Amazon S3 console.) Alternatively, you can also make a GetObjectAcl request to find the object owner's canonical ID to verify the object owner account. By default, ACLs grant explicit allow permissions for GET requests to the object owner's account.

After you've confirmed that the object owner is different from the bucket owner, then depending on your use case and access level, choose one of the following methods to help address the Access Denied (403 Forbidden) error:

- **Disable ACLs (recommended)** – This method will apply to all objects and can be performed by the bucket owner. This method automatically gives the bucket owner ownership and full control over every object in the bucket. Before you implement this method, check the prerequisites for disabling ACLs. For information about how to set your bucket to bucket owner enforced (recommended) mode, see Setting Object Ownership on an existing bucket.
  Important

  To prevent an Access Denied (403 Forbidden) error, be sure to migrate the ACL permissions to a bucket policy before you disable ACLs. For more information, see Bucket policy examples for migrating from ACL permissions.
- **Change the object owner to the bucket owner** – This method can be applied to individual objects, but only the object owner (or a user with the appropriate permissions) can change an object's ownership. Additional PUT costs might apply. (For more information, see Amazon S3 pricing.) This method grants the bucket owner full ownership of the object, allowing the bucket owner to control access to the object through a bucket policy.

  To change the object's ownership, do one of the following:
  - You (the bucket owner) can copy the object back to the bucket.
  - You can change the Object Ownership setting of the bucket to bucket owner preferred. If versioning is disabled, the objects in the bucket are overwritten. If versioning is enabled, duplicate versions of the same object

will appear in the bucket, which the bucket owner can set a lifecycle rule to expire. For instructions on how to change your Object Ownership setting, see Setting Object Ownership on an existing bucket.
    Note
    When you update your Object Ownership setting to bucket owner preferred, the setting is only applied to new objects that are uploaded to the bucket.
  - You can have the object owner upload the object again with the bucket-owner-full-control canned object ACL.

  Note

  For cross-account uploads, you can also require the bucket-owner-full-control canned object ACL in your bucket policy. For an example bucket policy, see Grant cross-account permissions to upload objects while ensuring that the bucket owner has full control.
- **Keep the object writer as the object owner** – This method doesn't change the object owner, but it does allow you to grant access to objects individually. To grant access to an object, you must have the PutObjectAcl permission for the object. Then, to fix the Access Denied (403 Forbidden) error, add the requester as a grantee to access the object in the object's ACLs. For more information, see Configuring ACLs.

S3 Block Public Access settings

If the failed request involves public access or public policies, then check the S3 Block Public Access settings on your account, bucket, or S3 access point. Starting in April 2023, all Block Public Access settings are enabled by default for new buckets. For more information about how Amazon S3 defines "public," see The meaning of "public".

When set to TRUE, Block Public Access settings act as explicit deny policies that override permissions allowed by ACLs, bucket policies, and IAM user policies. To determine whether your Block Public Access settings are rejecting your request, review the following scenarios:

- If the specified access control list (ACL) is public, then the BlockPublicAcls setting rejects your PutBucketAcl and PutObjectACL calls.
- If the request includes a public ACL, then the BlockPublicAcls setting rejects your PutObject calls.
- If the BlockPublicAcls setting is applied to an account and the request includes a public ACL, then any CreateBucket calls that include public ACLs will fail.
- If your request's permission is granted only by a public ACL, then the IgnorePublicAcls setting rejects the request.
- If the specified bucket policy allows public access, then the BlockPublicPolicy setting rejects your PutBucketPolicy calls.

- If the BlockPublicPolicy setting is applied to an access point, then all PutAccessPointPolicy and PutBucketPolicy calls that specify a public policy and are made through the access point will fail.
- If the access point or bucket has a public policy, then the RestrictPublicBuckets setting rejects all cross-account calls except for AWS service principals. This setting also rejects all anonymous (or unsigned) calls.

To review and update your Block Public Access setting configurations, see Configuring block public access settings for your S3 buckets.

Amazon S3 encryption settings

Amazon S3 supports server-side encryption on your bucket. Server-side encryption is the encryption of data at its destination by the application or service that receives it. Amazon S3 encrypts your data at the object level as it writes it to disks in AWS data centers and decrypts it for you when you access it.

By default, Amazon S3 now applies server-side encryption with Amazon S3 managed keys (SSE-S3) as the base level of encryption for every bucket in Amazon S3. Amazon S3 also allows you to specify the server-side encryption method when uploading objects.

To review your bucket's server-side encryption status and encryption settings

1. Sign in to the AWS Management Console and open the Amazon S3 console at https://console.aws.amazon.com/s3/.
2. In the left navigation pane, choose **Buckets**.
3. From the **Buckets** list, choose the bucket that you want to check the encryption settings for.
4. Choose the **Properties** tab.
5. Scroll down to the **Default encryption** section and view the **Encryption type** settings.

To check your encryption settings by using the AWS CLI, use the get-bucket-encryption command.

To check the encryption status of an object

1. Sign in to the AWS Management Console and open the Amazon S3 console at https://console.aws.amazon.com/s3/.
2. In the left navigation pane, choose **Buckets**.
3. From the **Buckets** list, choose the name of the bucket that contains the object.
4. From the **Objects** list, choose the name of the object that you want to add or change encryption for.

The object's details page appears.

5. Scroll down to the **Server-side encryption settings** section to view the object's server-side encryption settings.

To check your object encryption status by using the AWS CLI, use the head-object command.

Encryption and permissions requirements

Amazon S3 supports three types of server-side encryption:

- Server-side encryption with Amazon S3 managed keys (SSE-S3)
- Server-side encryption with AWS Key Management Service (AWS KMS) keys (SSE-KMS)
- Server-side encryption with customer-provided keys (SSE-C)

Based on your encryption settings, make sure that the following permissions requirements are met:

- **SSE-S3** – No extra permissions are required.
- **SSE-KMS (with a customer managed key)** – To upload objects, the `kms:GenerateDataKey` permission on the AWS KMS key is required. To download objects and perform multipart uploads of objects, the `kms:Decrypt` permission on the KMS key is required.
- **SSE-KMS (with an AWS managed key)** – The requester must be from the same account that owns the `aws/s3` KMS key. The requester must also have the correct Amazon S3 permissions to access the object.
- **SSE-C (with a customer provided key)** – No additional permissions are required. You can configure the bucket policy to require and restrict server-side encryption with customer-provided encryption keys for objects in your bucket.

If the object is encrypted with a customer managed key, make sure that the KMS key policy allows you to perform the `kms:GenerateDataKey` or `kms:Decrypt` actions. For instructions on checking your KMS key policy, see Viewing a key policy in the *AWS Key Management Service Developer Guide*.

S3 Object Lock settings

If your bucket has S3 Object Lock enabled and the object is protected by a retention period or legal hold, Amazon S3 returns an Access Denied (403 Forbidden) error when you try to delete the object.

To check whether the bucket has Object Lock enabled

1. Sign in to the AWS Management Console and open the Amazon S3 console at https://console.aws.amazon.com/s3/.
2. In the left navigation pane, choose **Buckets**.
3. From the **Buckets** list, choose the name of the bucket that you want to review.
4. Choose the **Properties** tab.
5. Scroll down to the **Object Lock** section. Verify whether the **Object Lock** setting is **Enabled** or **Disabled**.

To determine whether the object is protected by a retention period or legal hold, view the lock information for your object.

If the object is protected by a retention period or legal hold, check the following:

- If the object version is protected by the compliance retention mode, there is no way to permanently delete it. A permanent DELETE request from any requester, including the root user, will result in an Access Denied (403 Forbidden) error. Also, be aware that when you submit a DELETE request for an object that's protected by the compliance retention mode, Amazon S3 creates a delete marker for the object.
- If the object version is protected with governance retention mode and you have the s3:BypassGovernanceRetention permission, you can bypass the protection and permanently delete the version. For more information, see Bypassing governance mode.
- If the object version is protected by a legal hold, then a permanent DELETE request can result in an Access Denied (403 Forbidden) error. To permanently delete the object version, you must remove the legal hold on the object version. To remove a legal hold, you must have the s3:PutObjectLegalHold permission. For more information about removing a legal hold, see Configuring S3 Object Lock using the console.

VPC endpoint policy

If you're accessing Amazon S3 by using a virtual private cloud (VPC) endpoint, make sure that the VPC endpoint policy is not blocking you from accessing your Amazon S3 resources. By default, the VPC endpoint policy allows all requests to Amazon S3. You can also configure the VPC endpoint policy to restrict certain requests. For information about how to check your VPC endpoint policy, see Control access to VPC endpoints by using endpoint policies in the *AWS PrivateLink Guide*.

AWS Organizations policies

If your AWS account belongs to an organization, AWS Organizations policies can block you from accessing Amazon S3 resources. By default, AWS Organizations policies do not block any requests to Amazon S3. However, make sure that your AWS Organizations policies haven’t been configured to block access to S3 buckets. For instructions on how to check your AWS Organizations policies, see Listing all policies in the *AWS Organizations User Guide*.

Access point settings

If you receive an Access Denied (403 Forbidden) error while making requests through Amazon S3 access points, you might need to check the following:

- The configurations for your access points
- The IAM user policy that's used for your access points
- The bucket policy that's used to manage or configure your cross-account access points

Access point configurations and policies

- When you create an access point, you can choose to designate **Internet** or **VPC** as the network origin. If the network origin is set to VPC only, Amazon S3 will reject any requests made to the access point that don't originate from the specified VPC. To check the network origin of your access point, see Creating access points restricted to a virtual private cloud.
- With access points, you can also configure custom Block Public Access settings, which work similarly to the Block Public Access settings at the bucket or account level. To check your custom Block Public Access settings, see Managing public access to access points.
- To make successful requests to Amazon S3 by using access points, make sure that the requester has the necessary IAM permissions. For more information, see Configuring IAM policies for using access points.
- If the request involves cross-account access points, make sure that the bucket owner has updated the bucket policy to authorize requests from the access point. For more information, see Granting permissions for cross-account access points.

If the Access Denied (403 Forbidden) error still persists after checking all the items in this topic, retrieve your Amazon S3 request ID and contact AWS Support for additional guidance.

Troubleshooting AWS Glue

Gathering AWS Glue troubleshooting information

Gather the following information for each of these types of failures:

When a crawler fails, gather the following information:

- Crawler name

  Logs from crawler runs are located in CloudWatch Logs under `/aws-glue/crawlers`.

When a test connection fails, gather the following information:

- Connection name
- Connection ID
- JDBC connection string in the form `jdbc:protocol://host:port/database-name`.

  Logs from test connections are located in CloudWatch Logs under `/aws-glue/testconnection`.

When a job fails, gather the following information:

- Job name
- Job run ID in the form `jr_xxxxx`.

  Logs from job runs are located in CloudWatch Logs under `/aws-glue/jobs`.

Troubleshooting errors in AWS Glue for Spark

If you encounter errors in AWS Glue, use the following solutions to help you find the source of the problems and fix them.

Error: Resource unavailable

If AWS Glue returns a resource unavailable message, you can view error messages or logs to help you learn more about the issue. The following tasks describe general methods for troubleshooting.

- For any connections and development endpoints that you use, check that your cluster has not run out of elastic network interfaces.

Error: Could not find S3 endpoint or NAT gateway for subnetId in VPC

Check the subnet ID and VPC ID in the message to help you diagnose the issue.

- Check that you have an Amazon S3 VPC endpoint set up, which is required with AWS Glue. In addition, check your NAT gateway if that's part of your configuration. For more information, see Amazon VPC endpoints for Amazon S3.

Error: Inbound rule in security group required

At least one security group must open all ingress ports. To limit traffic, the source security group in your inbound rule can be restricted to the same security group.

- For any connections that you use, check your security group for an inbound rule that is self-referencing. For more information, see Setting up network access to data stores.
- When you are using a development endpoint, check your security group for an inbound rule that is self-referencing. For more information, see Setting up network access to data stores.

Error: Outbound rule in security group required

At least one security group must open all egress ports. To limit traffic, the source security group in your outbound rule can be restricted to the same security group.

- For any connections that you use, check your security group for an outbound rule that is self-referencing. For more information, see Setting up network access to data stores.

- When you are using a development endpoint, check your security group for an outbound rule that is self-referencing. For more information, see Setting up network access to data stores.

Error: Job run failed because the role passed should be given assume role permissions for the AWS Glue service

The user who defines a job must have permission for `iam:PassRole` for AWS Glue.

- When a user creates an AWS Glue job, confirm that the user's role contains a policy that contains `iam:PassRole` for AWS Glue. For more information, see Step 3: Attach a policy to users or groups that access AWS Glue.

Error: DescribeVpcEndpoints action is unauthorized. unable to validate VPC ID vpc-id

- Check the policy passed to AWS Glue for the `ec2:DescribeVpcEndpoints` permission.

Error: DescribeRouteTables action is unauthorized. unable to validate subnet id: Subnet-id in VPC id: vpc-id

- Check the policy passed to AWS Glue for the `ec2:DescribeRouteTables` permission.

Error: Failed to call ec2:DescribeSubnets

- Check the policy passed to AWS Glue for the `ec2:DescribeSubnets` permission.

Error: Failed to call ec2:DescribeSecurityGroups

- Check the policy passed to AWS Glue for the `ec2:DescribeSecurityGroups` permission.

Error: Could not find subnet for AZ

- The Availability Zone might not be available to AWS Glue. Create and use a new subnet in a different Availability Zone from the one specified in the message.

Error: Job run exception when writing to a JDBC target

When you are running a job that writes to a JDBC target, the job might encounter errors in the following scenarios:

- If your job writes to a Microsoft SQL Server table, and the table has columns defined as type Boolean, then the table must be predefined in the SQL Server database. When you define the job on the AWS Glue console using a SQL Server target with the option **Create tables in your data target**, don't map any source columns to a target column with data type Boolean. You might encounter an error when the job runs.

  You can avoid the error by doing the following:
  - Choose an existing table with the **Boolean** column.
  - Edit the ApplyMapping transform and map the **Boolean** column in the source to a number or string in the target.
  - Edit the ApplyMapping transform to remove the **Boolean** column from the source.
- If your job writes to an Oracle table, you might need to adjust the length of names of Oracle objects. In some versions of Oracle, the maximum identifier length is limited to 30 bytes or 128 bytes. This limit affects the table names and column names of Oracle target data stores.

  You can avoid the error by doing the following:
  - Name Oracle target tables within the limit for your version.
  - The default column names are generated from the field names in the data. To handle the case when the column names are longer than the limit, use ApplyMapping or RenameField transforms to change the name of the column to be within the limit.

Error: Amazon S3 timeout

If AWS Glue returns a connect timed out error, it might be because it is trying to access an Amazon S3 bucket in another AWS Region.

- An Amazon S3 VPC endpoint can only route traffic to buckets within an AWS Region. If you need to connect to buckets in other Regions, a possible workaround is to use a NAT gateway. For more information, see NAT Gateways.

Error: Amazon S3 access denied

If AWS Glue returns an access denied error to an Amazon S3 bucket or object, it might be because the IAM role provided does not have a policy with permission to your data store.

- An ETL job must have access to an Amazon S3 data store used as a source or target. A crawler must have access to an Amazon S3 data store that it crawls. For more information, see Step 2: Create an IAM role for AWS Glue.

Error: Amazon S3 access key ID does not exist

If AWS Glue returns an access key ID does not exist error when running a job, it might be because of one of the following reasons:

- An ETL job uses an IAM role to access data stores, confirm that the IAM role for your job was not deleted before the job started.
- An IAM role contains permissions to access your data stores, confirm that any attached Amazon S3 policy containing `s3:ListBucket` is correct.

Error: Job run fails when accessing Amazon S3 with an s3a:// URI

If a job run returns an error like *Failed to parse XML document with handler class* , it might be because of a failure trying to list hundreds of files using an `s3a://` URI. Access your data store using an `s3://` URI instead. The following exception trace highlights the errors to look for:

```
1.        com.amazonaws.SdkClientException: Failed to parse XML document with handler class com.amazonaws.services.s3.model.transform.XmlResponsesSaxParser$ListBucketHandler
```

```
2.        at
com.amazonaws.services.s3.model.transform.XmlResponsesSaxParser.parseXmlInput
Stream(XmlResponsesSaxParser.java:161)

3.        at
com.amazonaws.services.s3.model.transform.XmlResponsesSaxParser.parseListBuck
etObjectsResponse(XmlResponsesSaxParser.java:317)

4.        at
com.amazonaws.services.s3.model.transform.Unmarshallers$ListObjectsUnmarshaller.
unmarshall(Unmarshallers.java:70)

5.        at
com.amazonaws.services.s3.model.transform.Unmarshallers$ListObjectsUnmarshaller.
unmarshall(Unmarshallers.java:59)

6.        at
com.amazonaws.services.s3.internal.S3XmlResponseHandler.handle(S3XmlResponse
Handler.java:62)

7.        at
com.amazonaws.services.s3.internal.S3XmlResponseHandler.handle(S3XmlResponse
Handler.java:31)

8.        at
com.amazonaws.http.response.AwsResponseHandlerAdapter.handle(AwsResponseHa
ndlerAdapter.java:70)

9.        at
com.amazonaws.http.AmazonHttpClient$RequestExecutor.handleResponse(AmazonHt
tpClient.java:1554)

10.       at
com.amazonaws.http.AmazonHttpClient$RequestExecutor.executeOneRequest(Amazo
nHttpClient.java:1272)

11.       at
com.amazonaws.http.AmazonHttpClient$RequestExecutor.executeHelper(AmazonHttp
Client.java:1056)

12.       at
com.amazonaws.http.AmazonHttpClient$RequestExecutor.doExecute(AmazonHttpClie
nt.java:743)
```

```
13.        at com.amazonaws.http.AmazonHttpClient$RequestExecutor.executeWithTimer(AmazonHttpClient.java:717)
14.        at com.amazonaws.http.AmazonHttpClient$RequestExecutor.execute(AmazonHttpClient.java:699)
15.        at com.amazonaws.http.AmazonHttpClient$RequestExecutor.access$500(AmazonHttpClient.java:667)
16.        at com.amazonaws.http.AmazonHttpClient$RequestExecutionBuilderImpl.execute(AmazonHttpClient.java:649)
17.        at com.amazonaws.http.AmazonHttpClient.execute(AmazonHttpClient.java:513)
18.        at com.amazonaws.services.s3.AmazonS3Client.invoke(AmazonS3Client.java:4325)
19.        at com.amazonaws.services.s3.AmazonS3Client.invoke(AmazonS3Client.java:4272)
20.        at com.amazonaws.services.s3.AmazonS3Client.invoke(AmazonS3Client.java:4266)
21.        at com.amazonaws.services.s3.AmazonS3Client.listObjects(AmazonS3Client.java:834)
22.        at org.apache.hadoop.fs.s3a.S3AFileSystem.getFileStatus(S3AFileSystem.java:971)
23.        at org.apache.hadoop.fs.s3a.S3AFileSystem.deleteUnnecessaryFakeDirectories(S3AFileSystem.java:1155)
24.        at org.apache.hadoop.fs.s3a.S3AFileSystem.finishedWrite(S3AFileSystem.java:1144)
25.        at org.apache.hadoop.fs.s3a.S3AOutputStream.close(S3AOutputStream.java:142)
26.        at org.apache.hadoop.fs.FSDataOutputStream$PositionCache.close(FSDataOutputStream.java:74)
```

```
27.        at
org.apache.hadoop.fs.FSDataOutputStream.close(FSDataOutputStream.java:108)

28.        at
org.apache.parquet.hadoop.ParquetFileWriter.end(ParquetFileWriter.java:467)

29.        at
org.apache.parquet.hadoop.InternalParquetRecordWriter.close(InternalParquetRecord
Writer.java:117)

30.        at
org.apache.parquet.hadoop.ParquetRecordWriter.close(ParquetRecordWriter.java:112)

31.        at
org.apache.spark.sql.execution.datasources.parquet.ParquetOutputWriter.close(Parque
tOutputWriter.scala:44)

32.        at
org.apache.spark.sql.execution.datasources.FileFormatWriter$SingleDirectoryWriteTask
.releaseResources(FileFormatWriter.scala:252)

33.        at
org.apache.spark.sql.execution.datasources.FileFormatWriter$$anonfun$org$apache$s
park$sql$execution$datasources$FileFormatWriter$$executeTask$3.apply(FileFormat
Writer.scala:191)

34.        at
org.apache.spark.sql.execution.datasources.FileFormatWriter$$anonfun$org$apache$s
park$sql$execution$datasources$FileFormatWriter$$executeTask$3.apply(FileFormat
Writer.scala:188)

35.        at
org.apache.spark.util.Utils$.tryWithSafeFinallyAndFailureCallbacks(Utils.scala:1341)

36.        at
org.apache.spark.sql.execution.datasources.FileFormatWriter$.org$apache$spark$sql$
execution$datasources$FileFormatWriter$$executeTask(FileFormatWriter.scala:193)

37.        at
org.apache.spark.sql.execution.datasources.FileFormatWriter$$anonfun$write$1$$ano
nfun$3.apply(FileFormatWriter.scala:129)

38.        at
org.apache.spark.sql.execution.datasources.FileFormatWriter$$anonfun$write$1$$ano
nfun$3.apply(FileFormatWriter.scala:128)
```

```
	at org.apache.spark.scheduler.ResultTask.runTask(ResultTask.scala:87)
	at org.apache.spark.scheduler.Task.run(Task.scala:99)
	at org.apache.spark.executor.Executor$TaskRunner.run(Executor.scala:282)
	at java.util.concurrent.ThreadPoolExecutor.runWorker(ThreadPoolExecutor.java:1149)
	at java.util.concurrent.ThreadPoolExecutor$Worker.run(ThreadPoolExecutor.java:624)
	at java.lang.Thread.run(Thread.java:748)
```

Error: Amazon S3 service token expired

When moving data to and from Amazon Redshift, temporary Amazon S3 credentials, which expire after 1 hour, are used. If you have a long running job, it might fail. For information about how to set up your long running jobs to move data to and from Amazon Redshift, see aws-glue-programming-etl-connect-redshift-home.

Error: No private DNS for network interface found

If a job fails or a development endpoint fails to provision, it might be because of a problem in the network setup.

- If you are using the Amazon provided DNS, the value of `enableDnsHostnames` must be set to true. For more information, see DNS .

Error: Development endpoint provisioning failed

If AWS Glue fails to successfully provision a development endpoint, it might be because of a problem in the network setup.

- When you define a development endpoint, the VPC, subnet, and security groups are validated to confirm that they meet certain requirements.
- If you provided the optional SSH public key, check that it is a valid SSH public key.
- Check in the VPC console that your VPC uses a valid **DHCP option set**. For more information, see DHCP option sets.
- If the cluster remains in the PROVISIONING state, contact AWS Support.

Error: Notebook server CREATE_FAILED

If AWS Glue fails to create the notebook server for a development endpoint, it might be because of one of the following problems:

- AWS Glue passes an IAM role to Amazon EC2 when it is setting up the notebook server. The IAM role must have a trust relationship to Amazon EC2.
- The IAM role must have an instance profile of the same name. When you create the role for Amazon EC2 with the IAM console, the instance profile with the same name is automatically created. Check for an error in the log regarding the instance profile name iamInstanceProfile.name that is not valid. For more information, see Using Instance Profiles.
- Check that your role has permission to access aws-glue* buckets in the policy that you pass to create the notebook server.

Error: Local notebook fails to start

If your local notebook fails to start and reports errors that a directory or folder cannot be found, it might be because of one of the following problems:

- If you are running on Microsoft Windows, make sure that the JAVA_HOME environment variable points to the correct Java directory. It's possible to update Java without updating this variable, and if it points to a folder that no longer exists, Jupyter notebooks fail to start.

Error: Running crawler failed

If AWS Glue fails to successfully run a crawler to catalog your data, it might be because of one of the following reasons. First check if an error is listed in the AWS Glue console crawlers list. Check if there is an exclamation icon next to the crawler name and hover over the icon to see any associated messages.

- Check the logs for the crawler run in CloudWatch Logs under /aws-glue/crawlers.

Error: Partitions were not updated

In case your partitions were not updated in the Data Catalog when you ran an ETL job, these log statements from the DataSink class in the CloudWatch logs may be helpful:

- "Attempting to fast-forward updates to the Catalog - nameSpace:" — Shows which database, table, and catalogId are attempted to be modified by this job. If this statement is not here, check if enableUpdateCatalog is set to true and properly passed as a getSink() parameter or in additional_options.
- "Schema change policy behavior:" — Shows which schema updateBehavior value you passed in.
- "Schemas qualify (schema compare):" — Will be true or false.
- "Schemas qualify (case-insensitive compare):" — Will be true or false.
- If both are false and your updateBehavior is not set to UPDATE_IN_DATABASE, then your DynamicFrame schema needs to be identical or contain a subset of the columns seen in the Data Catalog table schema.

For more information on updating partitions, see Creating tables, updating the schema, and adding new partitions in the Data Catalog from AWS Glue ETL jobs.

Error: Upgrading Athena Data Catalog

If you encounter errors while upgrading your Athena Data Catalog to the AWS Glue Data Catalog, see Upgrading to the AWS Glue Data Catalog Step-by-Step.

Error: Job bookmark update failed due to version mismatch

You may be trying to parametize AWS Glue jobs to apply the same transformation/logic on different datasets in Amazon S3. You want to track processed files on the locations provided. When you run the same job on the same source bucket and write to the same/different destination concurrently (concurrency >1) the job fails with this error:

```
py4j.protocol.Py4JJavaError: An error occurred while callingz:com.amazonaws.services.glue.util.Job.commit.:com.amazonaws.services.gluejobexecutor.model.VersionMismatchException: Continuation update failed due to version mismatch. Expected version 2 but found version 3
```

Solution: set concurrency to 1 or don't run the job concurrently.

Currently AWS Glue bookmarks don't support concurrent job runs and commits will fail.

Error: A job is reprocessing data when job bookmarks are enabled

There might be cases when you have enabled AWS Glue job bookmarks, but your ETL job is reprocessing data that was already processed in an earlier run. Check for these common causes of this error:

Max Concurrency

Ensure that the maximum number of concurrent runs for the job is 1. For more information, see the discussion of max concurrency in Adding jobs in AWS Glue. When you have multiple concurrent jobs with job bookmarks and the maximum concurrency is set to 1, the job bookmark doesn't work correctly.

Missing Job Object

Ensure that your job run script ends with the following commit:

```
job.commit()
```

When you include this object, AWS Glue records the timestamp and path of the job run. If you run the job again with the same path, AWS Glue processes only the new files. If you don't include this object and job bookmarks are enabled, the job reprocesses the already processed files along with the new files and creates redundancy in the job's target data store.

Missing Transformation Context Parameter

Transformation context is an optional parameter in the `GlueContext` class, but job bookmarks don't work if you don't include it. To resolve this error, add the transformation context parameter when you create the DynamicFrame, as shown following:

```
sample_dynF=create_dynamic_frame_from_catalog(database,
table_name,transformation_ctx="sample_dynF")
```

Input Source

If you are using a relational database (a JDBC connection) for the input source, job bookmarks work only if the table's primary keys are in sequential order. Job bookmarks work for new rows, but not for updated rows. That is because job bookmarks look for the primary keys, which already exist. This does not apply if your input source is Amazon Simple Storage Service (Amazon S3).

Last Modified Time

For Amazon S3 input sources, job bookmarks check the last modified time of the objects, rather than the file names, to verify which objects need to be reprocessed. If your input source data has been modified since your last job run, the files are reprocessed when you run the job again.

Error: Failover behavior between VPCs in AWS Glue

The following process is used for failover for jobs in AWS Glue 4.0 and previous versions.

Summary: an AWS Glue connection is selected at the time a job run is submitted. If the job run encounters some issues, (lack of IP addresses, connectivity to source, routing problem), the job run will fail. If retries are configured, AWS Glue will retry with the same connection.

1. At the time of job run submission, AWS Glue picks the connection based on the order they are listed in the job configuration. It runs a validation, and if the validation succeeds AWS Glue will use that connection. AWS Glue tries the next connection if any of the validations fail.
2. AWS Glue validates the connection with the following:
    - checks for valid Amazon VPC id and subnet.
    - checks that a NAT gateway or Amazon VPC endpoint exists.
    - checks that the subnet has more than 0 allocated IP addresses.
    - checks that the AZ is healthy.

   AWS Glue cannot verify connectivity at the time of job run submission.
3. For jobs using Amazon VPC, all drivers and executors will be created in the same AZ with the connection selected at the time of job run submission.
4. If retries are configured, AWS Glue will retry with the same connection. This is because we cannot guarantee problems with this connection are long-running. If an AZ fails, existing job runs (depending on the stage of the job run) in that AZ can fail. A retry should detect an AZ failure and choose another AZ for the new run.

AWS Glue FAQ, or How to Get Things Done

1. How do I repartition or coalesce my output into more or fewer files?

AWS Glue is based on Apache Spark, which partitions data across multiple nodes to achieve high throughput. When writing data to a file-based sink like Amazon S3, Glue will write a separate file for each partition. In some cases it may be desirable to change the number of partitions, either to change the degree of parallelism or the number of output files.

To change the number of partitions in a DynamicFrame, you can first convert it into a DataFrame and then leverage Apache Spark's partitioning capabilities. For example, the first line of the following snippet converts the DynamicFrame called "datasource0" to a DataFrame and then repartitions it to a single partition. The second line converts it back to a DynamicFrame for further processing in AWS Glue.

```
# Convert to a dataframe and partition based on "partition_col"
partitioned_dataframe = datasource0.toDF().repartition(1)

# Convert back to a DynamicFrame for further processing.
partitioned_dynamicframe = DynamicFrame.fromDF(partitioned_dataframe,
glueContext, "partitioned_df")
```

You can also pass the name of a column to the repartition method to use that field as a partitioning key. This may help performance in certain cases where there is benefit in co-locating data. Note that while different records with the same value for this column will be assigned to the same partition, there is no guarantee that there will be a separate partition for each distinct value.

2. I have a ChoiceType in my schema. So...

a. **How can I convert to a data frame?**

Since DataFrames do not have the type flexibility that DynamicFrames do, you have to resolve the choice type in your DynamicFrame before conversion. Glue provides a transformation called ResolveChoice with the following signature:

```
ResolveChoice.apply(self, frame, specs = None, choice = "",
                    transformation_ctx = "", info = "",
                    stageThreshold = 0, totalThreshold = 0)
```

This transformation provides you two general ways to resolve choice types in a DynamicFrame.

- You can specify a list of (path, action) tuples for each individual choice column, where *path* is the full path of the column and *action* is the strategy to resolve the choice in this column.
- You can give an action for *all* the potential choice columns in your data using the *choice* parameter.

The *action* above is a string, one of four strategies that AWS Glue provides:

1. `cast` - When this is specified, the user must specify a type to cast to, such as `cast:int`.
2. `make_cols` This flattens a potential choice. For instance, if `col1` is `choice<int, string>`, then using `make_cols` creates two columns in the target: `col1_int` and `col1_string`.
3. `make_struct` This creates a struct containing both choices. For example, if `col1` is `choice<int, string>`, then using `make_struct` creates a column called `struct<int, string>` that contains one or the other of the choice types.
4. `project` When this is specified, the user must also specify a type. For instance, when `project:string` is specified for `col1` that is `choice<int, string>`, then the column produced in the target would only contain `string` values.

Once all the choice types in your DynamicFrame are resolved, you can convert it to a data frame using the 'toDF()' method.

b. **How do I write to targets that do not handle ChoiceTypes?**

Resolve the choice types as described above and then write the data out using DynamicFrame writers or DataFrame write, depending on your use case.

3. There are some transforms that I cannot figure out.

a. **How can I use SQL queries with DynamicFrames?**
You can leverage Spark's SQL engine to run SQL queries over your data. If you have a DynamicFrame called `my_dynamic_frame`, you can use the following snippet to convert the DynamicFrame to a DataFrame, issue a SQL query, and then convert back to a DynamicFrame.

```
df = my_dynamic_frame.toDF()
df.createOrReplaceTempView("temptable")
sql_df = spark.sql("SELECT * FROM temptable")
new_dynamic_frame = DynamicFrame.fromDF(sql_df, glueContext,
"new_dynamic_frame")
```

Note that we assign the `spark` variable at the start of generated scripts for convenience, but if you modify your script and delete this variable, you can also reference the SparkSession using `glueContext.spark_session`.

b. **How do I do filtering in DynamicFrames?**
DynamicFrames support basic filtering via the SplitRows transformation which partitions it into two new DynamicFrames based on a predicate. For example, the snippet partition my_dynamic_frame into two frames called "adults" and "youths".:

```
frame_collection = SplitRows.apply(my_dynamic_frame,
                                   {"age": {">": 21}},
                                   "adults", "youths")
```

You can access these by indexing into the frame_collection. For instance, `frame_collection['adults']` returns the DynamicFrame containing all records with `age > 21`.
It is possible to perform more sophisticated filtering by converting to a DataFrame and then using the filter method. For instance, the query above could be expressed as:

```
result = my_dynamic_frame.toDF().filter("age > 21")
new_dynamic_frame = DynamicFrame.fromDF(result, glueContext,
"new_dynamic_frame")
```

More information about Spark SQL's filter syntax can be found in the Spark SQL Programming Guide.

c. **Can I use a lambda function with a DynamicFrame?** Lambda functions and other user-defined functions are currently only supported using SparkSQL DataFrames. You can convert a DynamicFrame to a DataFrame using the `toDF()` method and then specify Python functions (including lambdas) when calling methods like `foreach`. More information about methods on DataFrames can be found in the Spark SQL Programming Guide or the PySpark Documentation.

d. **So, what else can I do with DynamicFrames?**
DynamicFrames are designed to provide maximum flexibility when dealing with messy data that may lack a declared schema. Records are represented in a flexible self-describing way that preserves information about schema inconsistencies in the data.
For example, with changing requirements, an address column stored as a `string` in some records might be stored as a `struct` in later rows. Rather than failing or falling back to a `string`, DynamicFrames will track both types and gives users a number of options in how to resolve these inconsistencies, providing fine grain resolution options via the `ResolveChoice` transforms.
DynamicFrames also provide a number of powerful high-level ETL operations that are not found in DataFrames. For example, the `Relationalize` transform can be used to flatten and pivot

complex nested data into tables suitable for transfer to a relational database. In additon, the `ApplyMapping` transform supports complex renames and casting in a declarative fashion. DynamicFrames are also integrated with the AWS Glue Data Catalog, so creating frames from tables is a simple operation. Writing to databases can be done through connections without specifying the password. Moreover, DynamicFrames are integrated with job bookmarks, so running these scripts in the job system can allow the script to implictly keep track of what was read and written.
We are continually adding new transform, so be sure to check our documentation and let us know if there are new transforms that would be useful to you.

4. File Formats?

a. *Which file formats do you support for input and for output?*
Out of the box we support JSON, CSV, ORC, Parquet, and Avro.
b. *What compression types do you support?*
We support gzip, bzip2, and lz4.

5. JDBC access?

a. *Which JDBC databases do you support?*
Out of the box we support Postgres, MySQL, Redshift, and Aurora.
b. *Can you support my JDBC driver for database XYZ?*
In the case of unsupported databases, we fall back to using Spark. You can specify the driver and driver options using the options fields, and make the driver available using the *–extra-jars* options in the job arguments.

6. How can I handle multi-line CSV files?

By default we assume that each CSV record is contained on a single line. This allows us to automatically split large files to achieve much better parallelism while reading. If your CSV data does contain multiline fields enclosed in double-quotes, you can set the 'multiLine' table property in the Data Catalog to 'true' to disable splitting.

7. My test connection is not working... What do I do?

AWS Glue uses private IP addresses in the subnet while creating Elastic Network Interface(s) in the customer's specified VPC/Subnet. Security groups specified in the Connection are applied on each of the ENIs. Check whether your Security Groups allow outbound access and whether they allow connectivity to the database cluster. Also, Spark requires bi-directional connectivity among driver and executor nodes. One of the security groups need to allow ingress rules on all TCP ports. You can prevent it from being open to the world by restricting the source of the Security Group to itself (self-referential security group).

8. I'm getting the error that I have no S3 access... What do I do?

AWS Glue uses private IP addresses in the subnet while creating Elastic Network Interface(s) in customer's specified VPC/Subnet. Check your VPC route tables to ensure that there is an S3 VPC Endpoint so that traffic does not leave out to the internet. If the S3 buckets that you need to access are in a different region, you need to set up a NAT Gateway (the IP addresses are private).

9. How do I take advantage of JobBookmarks?

a. *What is a JobBookmark?*
A JobBookmark captures the state of job. It is composed of states for various elements of the job, including sources, transformations, and sinks. Currently we only have implementation for S3 sources and `Relationalize`. Enabling a JobBookmark ensures that if a job is run again after a

previous successful run, it will continue from where it left off. However, if a job is run after a previous failed run, it will process the data that it failed to process in the previous attempt.
b. *How do I enable/disable Job Bookmarks?*
Bookmarks are optional and can be disabled or suspended and re-enabled in the console.
c. *I modified my script... can I reset my JobBookmark?*
Yes if you want to reset the Job bookmark, it can be reset in the console, or by calling the `ResetJobBookmark` API.
d. *I made a mistake... can I rewind my JobBookmark?*
Currently we don't support rewinding to any arbitrary state. You can manually clean up the data and reset the bookmark.

10. What is a development endpoint?

A DevEndpoint is used for developing and debugging your ETL scripts. You can connect a Notebook such as Zeppelin, or an IDE, or a terminal to a DevEndpoint, which can then provide interactive development and testing of a pyspark script. Once the script succeeds in the DevEndpoint, you can upload the script to S3 and run it in a Job.

11. Custom Libraries

a. *How do I create a Python library and use it with Glue?*
You can split your script into multiple scripts and refer to these functions in the main script or within the scripts. You can use the `extra python files` option in the job to provide a list of files that will be made available to the main script. If you have more than a handful of files or if they are in some hierarchy, you can create a `zip` archive of the files and just pass the archive as the `extra python files` in the Job option.
b. *How do I create a Java library and use it with Glue?
It is not a typical use case to write to java function and invoke it in Python. However, it is possible to invoke a java function from Python as follows:

```
from py4j.java_gateway import java_import
java_import(sc._jvm, "com.amazonaws.glue.userjava.UserJava")
uj = sc._jvm.UserJava()
uj.foo()
```

Provide the jar which has the class `com.amazonaws.glue.userjava.UserJava` using the `extra jars` options in the Job argument.

12. GlueContext, SparkContext, and SparkSession, Oh My!

With Spark 2.1, SparkSession is the recommended way to run SQL queries or create temporary table views. Here is an example of a SQL query that uses a SparkSession:

```
sql_df = spark.sql("SELECT * FROM temptable")
```

To simplify using spark for registered jobs in AWS Glue, our code generator initializes the spark session in the `spark` variable similar to GlueContext and SparkContext.

```
 spark = glueContext.spark_session
```

On DevEndpoints, a user can initialize the spark session herself in a similar way.

13. Can I use a graphical tool to build my ETL scripts?

I don't like to write programs, and the console doesn't provide all the transformations I need... Is there any other graphical tool that I can use for building ETL scripts?
We are constantly improving our suite of transformations as well as the ability to graphically construct ETL flows. We would love your feedback on what new transforms you'd like to have and what tools you'd like us to support.

14. Do you support other sources and sinks?

How about DynamoDB, Kinesis, ElasticSearch, and others like those?

We are constantly adding connectors to new data sources. In the meantime, our environment comes with Boto3 pre-installed, so for small data sets you can connect directly to these services using standard API calls through Python. For larger data sets, you can dump their data into S3 periodically and use AWS Glue to schedule jobs to process those data sets.

15. I'm having a problem recrawling an S3 bucket...

It is possible that you might be encountering this problem: Suppose you have an s3 bucket with contents like this:

```
billing
|--- year=2016
|--- year=2017
|--- unrelated.csv
```

- The `billing` folder contains billing information partitioned by year, while `unrelated.csv` is a file containing unrelated data.
- So you created a crawler with target {'S3 path' : 'billing'}, but you were unaware of the `unrelated csv` file. You expected the crawl to create a single table called `billing`.
- But instead, you ended up with three tables named `year=2016`, `year=2017`, and `unrelated_csv`.
- So, suppose you now *exclude* the `unrelated.csv` file and crawl again.
- Again, you expect that one `billing' table will be created, and the other tables will be deleted... but it doesn't work!

To make the recrawl work properly, you actually have to *remove* the `unrelated.csv` file from the bucket-- excluding it will not work.