
Amazon Simple Storage Service

User Guide

API Version 2006-03-01



Amazon Simple Storage Service: User Guide

Copyright © 2022 Amazon Web Services, Inc. and/or its affiliates. All rights reserved.

Amazon's trademarks and trade dress may not be used in connection with any product or service that is not Amazon's, in any manner that is likely to cause confusion among customers, or in any manner that disparages or discredits Amazon. All other trademarks not owned by Amazon are the property of their respective owners, who may or may not be affiliated with, connected to, or sponsored by Amazon.

Table of Contents

What is Amazon S3	1
Features of Amazon S3	1
Storage classes	1
Storage management	1
Access management	2
Data processing	2
Storage logging and monitoring	2
Analytics and insights	3
Strong consistency	3
How Amazon S3 works	3
Buckets	4
Objects	4
Keys	5
S3 Versioning	5
Version ID	5
Bucket policy	5
S3 Access Points	5
Access control lists (ACLs)	6
Regions	6
Amazon S3 data consistency model	6
Concurrent applications	7
Related services	8
Accessing Amazon S3	9
AWS Management Console	9
AWS Command Line Interface	9
AWS SDKs	9
Amazon S3 REST API	9
Paying for Amazon S3	10
PCI DSS compliance	10
Getting started	11
Setting up	11
Sign up for AWS	11
Create an IAM user	12
Sign in as an IAM user	13
Step 1: Create a bucket	13
Step 2: Upload an object	15
Step 3: Download an object	15
Using the S3 console	16
Step 4: Copy an object	16
Step 5: Delete the objects and bucket	17
Deleting an object	17
Emptying your bucket	18
Deleting your bucket	18
Next steps	18
Understand common use cases	19
Control access to your buckets and objects	19
Explore training and support	19
Manage and monitor your storage	20
Develop with Amazon S3	20
Access control	22
Creating a new bucket	22
Storing and sharing data	23
Sharing resources	24
Protecting data	24

Tutorials	27
Transforming data with S3 Object Lambda	27
Prerequisites	28
Step 1: Create an S3 bucket	30
Step 2: Upload a file to the S3 bucket	30
Step 3: Create an S3 access point	31
Step 4: Create a Lambda function	31
Step 5: Configure an IAM policy for your Lambda function's execution role	36
Step 6: Create an S3 Object Lambda access point	36
Step 7: View the transformed data	37
Step 8: Clean up	39
Next steps	41
Detecting and redacting PII data	41
Prerequisites: Create an IAM user with permissions	43
Step 1: Create an S3 bucket	44
Step 2: Upload a file to the S3 bucket	44
Step 3: Create an S3 access point	45
Step 4: Configure and deploy a prebuilt Lambda function	46
Step 5: Create an S3 Object Lambda access point	46
Step 6: Use the S3 Object Lambda access point to retrieve the redacted file	48
Step 7: Clean up	48
Next steps	51
Hosting video streaming	51
Prerequisites: Register and configure a custom domain with Route 53	52
Step 1: Create an S3 bucket	53
Step 2: Upload a video to the S3 bucket	54
Step 3: Create a CloudFront origin access identity	54
Step 4: Create a CloudFront distribution	55
Step 5: Access the video through the CloudFront distribution	56
Step 6: Configure your CloudFront distribution to use your custom domain name	57
Step 7: Access the S3 video through the CloudFront distribution with the custom domain name ..	60
(Optional) Step 8: View data about requests received by your CloudFront distribution	61
Step 9: Clean up	61
Next steps	64
Batch-transcoding videos	64
Prerequisites	65
Step 1: Create an S3 bucket for the output media files	66
Step 2: Create an IAM role for MediaConvert	67
Step 3: Create an IAM role for your Lambda function	68
Step 4: Create a Lambda function for video transcoding	69
Step 5: Configure Amazon S3 Inventory for your S3 source bucket	81
Step 6: Create an IAM role for S3 Batch Operations	84
Step 7: Create and run an S3 Batch Operations job	86
Step 8: Check the output media files from your S3 destination bucket	89
Step 9: Clean up	90
Next steps	92
Configuring a static website	92
Step 1: Create a bucket	92
Step 2: Enable static website hosting	93
Step 3: Edit Block Public Access settings	93
Step 4: Add a bucket policy that makes your bucket content publicly available	94
Step 5: Configure an index document	95
Step 6: Configure an error document	96
Step 7: Test your website endpoint	97
Step 8: Clean up	97
Configuring a static website using a custom domain	97
Before you begin	98

Step 1: Register a custom domain with Route 53	98
Step 2: Create two buckets	99
Step 3: Configure root Domain bucket	99
Step 4: Configure subdomain bucket for redirect	100
Step 5: Configure logging	101
Step 6: Upload index and website content	102
Step 7: Upload an error document	102
Step 8: Edit Block Public Access	103
Step 9: Attach a bucket policy	104
Step 10: Test your domain endpoint	105
Step 11: Add alias records	105
Step 12: Test the website	108
Speeding up your website with Amazon CloudFront	109
Cleaning up example resources	112
Working with buckets	114
Buckets overview	114
About permissions	115
Managing public access to buckets	115
Bucket configuration	116
Naming rules	118
Example bucket names	118
Creating a bucket	119
Viewing bucket properties	124
Methods for accessing a bucket	125
Virtual-hosted-style access	126
Path-style access	126
Accessing an S3 bucket over IPv6	126
Accessing a bucket through S3 access points	126
Accessing a bucket using S3://	127
Emptying a bucket	127
Deleting a bucket	129
Setting default bucket encryption	132
Using encryption for cross-account operations	132
Using default encryption with replication	133
Using Amazon S3 Bucket Keys with default encryption	133
Enabling default encryption	133
Monitoring default encryption	136
Configuring Transfer Acceleration	136
Why use Transfer Acceleration?	136
Requirements for using Transfer Acceleration	137
Getting Started	137
Enabling Transfer Acceleration	139
Speed Comparison tool	143
Using Requester Pays	144
How Requester Pays charges work	144
Configuring Requester Pays	145
Retrieving the requestPayment configuration	146
Downloading objects in Requester Pays buckets	146
Restrictions and limitations	147
Working with objects	149
Objects	149
Subresources	150
Creating object keys	150
Object key naming guidelines	151
Working with metadata	153
System-defined object metadata	154
User-defined object metadata	155

Editing object metadata	157
Uploading objects	158
Using multipart upload	167
Multipart upload process	167
Checksums with multipart upload operations	168
Concurrent multipart upload operations	169
Multipart upload and pricing	169
API support for multipart upload	169
AWS Command Line Interface support for multipart upload	170
AWS SDK support for multipart upload	170
Multipart upload API and permissions	170
Configuring a lifecycle policy	172
Uploading an object using multipart upload	174
Uploading a directory	187
Listing multipart uploads	188
Tracking a multipart upload	190
Aborting a multipart upload	193
Copying an object	196
Multipart upload limits	201
Copying objects	201
To copy an object	202
Downloading an object	209
Checking object integrity	215
Using supported checksum algorithms	215
Using Content-MD5 when uploading objects	221
Using Content-MD5 and the ETag to verify uploaded objects	221
Using trailing checksums	221
Using part-level checksums for multipart uploads	222
Deleting objects	223
Programmatically deleting objects from a version-enabled bucket	223
Deleting objects from an MFA-enabled bucket	224
Deleting a single object	224
Deleting multiple objects	231
Organizing and listing objects	243
Using prefixes	243
Listing objects	245
Using folders	254
Viewing an object overview	256
Viewing object properties	256
Using presigned URLs	257
Limiting presigned URL capabilities	257
Who can create a presigned URL	258
When does Amazon S3 check the expiration date and time of a presigned URL?	258
Sharing objects	258
Uploading objects	262
Deleting an object	270
Transforming objects	271
Creating Object Lambda Access Points	273
Using Amazon S3 Object Lambda Access Points	277
Getting started with an AWS CloudFormation template	278
Configuring IAM policies	281
Writing Lambda functions	284
Using AWS built functions	296
Best practices and guidelines for S3 Object Lambda	298
Security considerations	299
S3 Object Lambda tutorials	300
Working with access points	301

Configuring IAM policies	301
Condition keys	302
Delegating access control to access points	302
Access point policy examples	303
Creating access points	306
Rules for naming Amazon S3 access points	306
Creating an access point	306
Creating access points restricted to a VPC	308
Managing public access	309
Using access points	310
Monitoring and logging	311
Managing access points	312
Using a bucket-style alias for your access point	314
Using access points	315
Restrictions and limitations	318
Working with Multi-Region Access Points	319
Creating Multi-Region Access Points	320
Rules for naming Amazon S3 Multi-Region Access Points	321
Rules for choosing buckets for Amazon S3 Multi-Region Access Points	322
Blocking public access with Amazon S3 Multi-Region Access Points	323
Creating Amazon S3 Multi-Region Access Points	323
Configuring AWS PrivateLink	324
Using a Multi-Region Access Point	326
Multi-Region Access Point hostnames	327
Multi-Region Access Points and Amazon S3 Transfer Acceleration	328
Multi-Region Access Point permissions	328
Request routing	329
Bucket replication	330
Supported operations	331
Managing Multi-Region Access Points	331
Monitoring and logging	332
Monitoring and logging requests made to Multi-Region Access Point management APIs	333
Using CloudTrail	333
Restrictions and limitations	334
Security	336
Data protection	336
Data encryption	337
Server-side encryption	338
Using client-side encryption	381
Internetwork privacy	385
Traffic between service and on-premises clients and applications	385
Traffic between AWS resources in the same Region	385
AWS PrivateLink for Amazon S3	385
Types of VPC endpoints	386
Restrictions and limitations of AWS PrivateLink for Amazon S3	387
Creating a VPC endpoint	387
Accessing Amazon S3 interface endpoints	387
Accessing buckets and S3 access points from S3 interface endpoints	387
Updating an on-premises DNS configuration	391
Creating a VPC endpoint policy	392
Identity and access management	394
Overview	395
Access policy guidelines	400
Request authorization	404
Bucket policies and user policies	411
AWS managed policies	552
Managing access with ACLs	554

Using CORS	573
Blocking public access	584
Reviewing bucket access	593
Verifying bucket ownership	598
Controlling object ownership	601
Object Ownership settings	602
Changes introduced by disabling ACLs	603
Prerequisites for disabling ACLs	605
Object Ownership permissions	606
Disabling ACLs for all new buckets	606
Replication and Object Ownership	607
Setting Object Ownership	607
Prerequisites for disabling ACLs	607
Creating a bucket	616
Setting Object Ownership	618
Viewing Object Ownership settings	621
Disabling ACLs for all new buckets	622
Troubleshooting	624
Logging and monitoring	626
Compliance Validation	627
Resilience	628
Backup encryption	630
Infrastructure security	631
Configuration and vulnerability analysis	632
Security Best Practices	633
Amazon S3 Preventative Security Best Practices	633
Amazon S3 Monitoring and Auditing Best Practices	635
Managing storage	638
Using S3 Versioning	638
Unversioned, versioning-enabled, and versioning-suspended buckets	639
Using S3 Versioning with S3 Lifecycle	639
S3 Versioning	640
Enabling versioning on buckets	643
Configuring MFA delete	648
Working with versioning-enabled objects	649
Working with versioning-suspended objects	667
Using AWS Backup for Amazon S3	669
Working with archived objects	670
Archive retrieval options	671
Restoring an archived object	673
Querying archived objects	677
Using Object Lock	680
S3 Object Lock	681
Configuring Object Lock on the console	684
Managing Object Lock	685
Managing storage classes	688
Frequently accessed objects	688
Automatically optimizing data with changing or unknown access patterns	689
Infrequently accessed objects	689
Archiving objects	690
Amazon S3 on Outposts	691
Comparing storage classes	692
Setting the storage class of an object	692
Amazon S3 Intelligent-Tiering	693
How S3 Intelligent-Tiering works	693
Using S3 Intelligent-Tiering	695
Managing S3 Intelligent-Tiering	698

Managing lifecycle	701
Managing object lifecycle	702
Creating a lifecycle configuration	702
Transitioning objects	703
Expiring objects	707
Setting lifecycle configuration	708
Using other bucket configurations	719
Configuring Lifecycle event notifications	720
Lifecycle configuration elements	721
Examples of S3 Lifecycle configuration	728
Managing inventory	739
Amazon S3 Inventory buckets	739
Inventory lists	740
Configuring Amazon S3 Inventory	741
Setting up notifications for inventory completion	745
Locating your inventory	746
Querying inventory with Athena	749
Converting empty version ID strings to null strings	751
Replicating objects	753
Why use replication	753
When to use Cross-Region Replication	754
When to use Same-Region Replication	754
When to use S3 Batch Replication	755
Requirements for replication	755
What's replicated?	756
Setting up replication	758
Replicate existing objects	798
Additional configurations	805
Getting replication status	820
Troubleshooting	822
Additional considerations	823
Using object tags	825
API operations related to object tagging	827
Additional configurations	827
Access control	828
Managing object tags	831
Using cost allocation tags	834
More Info	835
Billing and usage reporting	836
Using Amazon S3 Select	852
Requirements and limits	852
Constructing a request	853
Errors	853
S3 Select examples	854
SQL Reference	856
Using Batch Operations	881
Batch Operations basics	881
Granting permissions	882
Creating a job	889
Supported operations	895
Managing jobs	920
Tracking job status and completion reports	922
Using tags	932
Managing S3 Object Lock	942
S3 Batch Operations tutorial	958
Monitoring Amazon S3	959
Monitoring tools	959

Automated tools	959
Manual tools	960
Logging options	960
Logging with CloudTrail	962
Using CloudTrail logs with Amazon S3 server access logs and CloudWatch Logs	962
CloudTrail tracking with Amazon S3 SOAP API calls	963
CloudTrail events	963
Example log files	967
Enabling CloudTrail	971
Identifying S3 requests	972
Logging server access	978
How do I enable log delivery?	978
Log object key format	979
How are logs delivered?	979
Best effort server log delivery	980
Bucket logging status changes take effect over time	980
Enabling server access logging	980
Log format	989
Deleting log files	998
Identifying S3 requests	998
Monitoring metrics with CloudWatch	1002
Metrics and dimensions	1004
Accessing CloudWatch metrics	1011
CloudWatch metrics configurations	1011
Amazon S3 Event Notifications	1017
Overview	1017
Notification types and destinations	1018
Using SQS, SNS, and Lambda	1022
Using EventBridge	1041
Using analytics and insights	1048
Storage Class Analysis	1048
How to set up storage class analysis	1048
Storage class analysis	1049
How can I export storage class analysis data?	1050
Configuring storage class analysis	1051
S3 Storage Lens	1053
Understanding S3 Storage Lens	1054
Working with Organizations	1059
S3 Storage Lens permissions	1061
Viewing storage metrics	1063
Using Amazon S3 Storage Lens to optimize your storage costs	1078
Metrics glossary	1081
Working with S3 Storage Lens	1086
Tracing requests using X-Ray	1114
How X-Ray works with Amazon S3	1114
Available Regions	1115
Hosting a static website	1116
Website endpoints	1116
Website endpoint examples	1117
Adding a DNS CNAME	1117
Using a custom domain with Route 53	1118
Key differences between a website endpoint and a REST API endpoint	1118
Enabling website hosting	1118
Configuring an index document	1122
Index document and folders	1122
Configure an index document	1123
Configuring a custom error document	1124

Amazon S3 HTTP response codes	1124
Configuring a custom error document	1125
Setting permissions for website access	1126
Step 1: Edit S3 Block Public Access settings	1127
Step 2: Add a bucket policy	1128
Object access control lists	1129
Logging web traffic	1130
Configuring a redirect	1130
Redirect requests to another host	1131
Configure redirection rules	1131
Redirect requests for an object	1136
Developing with Amazon S3	1138
Making requests	1138
About access keys	1138
Request endpoints	1140
Making requests over IPv6	1140
Making requests using the AWS SDKs	1147
Making requests using the REST API	1174
Using the AWS CLI	1183
Using the AWS SDKs	1184
Working with AWS SDKs	1185
Specifying the Signature Version in Request Authentication	1186
Using the AWS SDK for Java	1191
Using the AWS SDK for .NET	1192
Using the AWS SDK for PHP and Running PHP Examples	1193
Using the AWS SDK for Ruby - Version 3	1194
Using the AWS SDK for Python (Boto)	1196
Using the AWS Mobile SDKs for iOS and Android	1196
Using the AWS Amplify JavaScript Library	1196
Using the AWS SDK for JavaScript	1196
Using the REST API	1197
Request routing	1197
Error handling	1201
The REST error response	1201
The SOAP error response	1202
Amazon S3 error best practices	1203
Reference	1204
Appendix a: Using the SOAP API	1204
Appendix b: Authenticating requests (AWS signature version 2)	1207
Optimizing Amazon S3 performance	1235
Performance Guidelines	1235
Measure Performance	1236
Scale Horizontally	1236
Use Byte-Range Fetches	1236
Retry Requests	1236
Combine Amazon S3 and Amazon EC2 in the Same Region	1236
Use Transfer Acceleration to Minimize Latency	1237
Use the Latest AWS SDKs	1237
Performance Design Patterns	1237
Caching Frequently Accessed Content	1238
Timeouts and Retries for Latency-Sensitive Apps	1238
Horizontal Scaling and Request Parallelization	1239
Accelerating Geographically Disparate Data Transfers	1240
What is S3 on Outposts?	1241
How S3 on Outposts works	1241
Regions	1241
Buckets	1242

Objects	1242
Keys	1242
Storage class and encryption	1243
Bucket policy	1243
S3 on Outposts access points	1243
Features of S3 on Outposts	1243
Access management	1243
Storage logging and monitoring	1244
Strong consistency	1244
Related services	1244
Accessing S3 on Outposts	1245
AWS Management Console	1245
AWS Command Line Interface	1245
AWS SDKs	1245
Paying for S3 on Outposts	1245
Next steps	1245
Setting up your Outpost	1246
Order a new Outpost	1246
Add Amazon S3 storage to an existing Outpost	1246
How S3 on Outposts is different	1247
Specifications	1247
Supported API operations	1247
Unsupported Amazon S3 features	1247
Network restrictions	1248
Getting started with S3 on Outposts	1249
Setting up IAM	1249
Using the S3 console	1254
Using the AWS CLI and SDK for Java	1255
Networking for S3 on Outposts	1260
Choosing your networking access type	1260
Accessing your S3 on Outposts buckets and objects	1260
Managing connections using cross-account elastic network interfaces	1261
Working with S3 on Outposts buckets	1261
Buckets	1261
Access points	1261
Endpoints	1262
API operations on S3 on Outposts	1262
Creating and managing S3 on Outposts buckets	1263
Creating a bucket	1263
Adding tags	1265
Creating and managing a lifecycle configuration	1266
Using bucket policies	1271
Listing buckets	1274
Getting a bucket	1275
Deleting your bucket	1276
Working with access points	1277
Working with endpoints	1283
Working with S3 on Outposts objects	1286
Copying an object	1287
Getting an object	1288
Listing objects	1290
Deleting objects	1292
Using HeadBucket	1295
Performing a multipart upload	1296
Using presigned URLs	1301
Security	1310
Data encryption	1310

AWS PrivateLink for S3 on Outposts	1311
Signature Version 4 (SigV4) policy keys	1315
Managing S3 on Outposts storage	1317
CloudWatch metrics	1318
Amazon CloudWatch Events	1319
CloudTrail logs	1319
Sharing S3 on Outposts	1320
Other services	1322
Developing with S3 on Outposts	1323
S3 on Outposts APIs	1323
Configuring S3 control client	1325
Code examples	1326
Actions	1327
Add CORS rules to a bucket	1328
Add a lifecycle configuration to a bucket	1332
Add a policy to a bucket	1336
Copy an object from one bucket to another	1341
Create a bucket	1349
Delete CORS rules from a bucket	1357
Delete a policy from a bucket	1358
Delete an empty bucket	1362
Delete an object	1367
Delete multiple objects	1373
Delete the lifecycle configuration of a bucket	1381
Delete the website configuration from a bucket	1382
Determine the existence and content type of an object	1384
Determine the existence of a bucket	1385
Get CORS rules for a bucket	1386
Get an object from a bucket	1388
Get the ACL of a bucket	1398
Get the ACL of an object	1401
Get the Region location for a bucket	1405
Get the lifecycle configuration of a bucket	1406
Get the policy for a bucket	1407
Get the website configuration for a bucket	1411
List buckets	1413
List in-progress multipart uploads	1417
List object versions in a bucket	1418
List objects in a bucket	1419
Restore an archived copy of an object	1427
Set a new ACL for a bucket	1428
Set the ACL of an object	1431
Set the website configuration for a bucket	1432
Upload an object to a bucket	1436
Scenarios	1449
Create a presigned URL	1450
Getting started with buckets and objects	1457
Manage versioned objects in batches with a Lambda function	1492
Use a transfer manager to upload and download files	1492
Work with versioned objects	1506
Cross-service examples	1510
Build an Amazon Transcribe app	1510
Convert text to speech and back to text	1511
Create a long-lived Amazon EMR cluster and run several steps	1512
Create a short-lived Amazon EMR cluster and run a step	1512
Create an Amazon Textract explorer application	1513
Detect PPE in images	1514

Detect entities in text extracted from an image	1515
Detect faces in an image	1515
Detect objects in images	1516
Detect people and objects in a video	1518
Save EXIF and other image information	1519
Troubleshooting	1521
Troubleshooting Amazon S3 by Symptom	1521
Significant Increases in HTTP 503 Responses to Requests to Buckets with Versioning Enabled ..	1521
Unexpected Behavior When Accessing Buckets Set with CORS	1522
Getting Amazon S3 Request IDs for AWS Support	1522
Using HTTP to Obtain Request IDs	1522
Using a Web Browser to Obtain Request IDs	1523
Using AWS SDKs to Obtain Request IDs	1523
Using the AWS CLI to Obtain Request IDs	1524
Related Topics	1524
Document history	1526
Earlier updates	1537
AWS glossary	1551

What is Amazon S3?

Amazon Simple Storage Service (Amazon S3) is an object storage service that offers industry-leading scalability, data availability, security, and performance. Customers of all sizes and industries can use Amazon S3 to store and protect any amount of data for a range of use cases, such as data lakes, websites, mobile applications, backup and restore, archive, enterprise applications, IoT devices, and big data analytics. Amazon S3 provides management features so that you can optimize, organize, and configure access to your data to meet your specific business, organizational, and compliance requirements.

Topics

- [Features of Amazon S3 \(p. 1\)](#)
- [How Amazon S3 works \(p. 3\)](#)
- [Amazon S3 data consistency model \(p. 6\)](#)
- [Related services \(p. 8\)](#)
- [Accessing Amazon S3 \(p. 9\)](#)
- [Paying for Amazon S3 \(p. 10\)](#)
- [PCI DSS compliance \(p. 10\)](#)

Features of Amazon S3

Storage classes

Amazon S3 offers a range of storage classes designed for different use cases. For example, you can store mission-critical production data in S3 Standard for frequent access, save costs by storing infrequently accessed data in S3 Standard-IA or S3 One Zone-IA, and archive data at the lowest costs in S3 Glacier Instant Retrieval, S3 Glacier Flexible Retrieval, and S3 Glacier Deep Archive.

You can store data with changing or unknown access patterns in S3 Intelligent-Tiering, which optimizes storage costs by automatically moving your data between four access tiers when your access patterns change. These four access tiers include two low-latency access tiers optimized for frequent and infrequent access, and two opt-in archive access tiers designed for asynchronous access for rarely accessed data.

For more information, see [Using Amazon S3 storage classes \(p. 688\)](#). For more information about S3 Glacier Flexible Retrieval, see the [Amazon S3 Glacier Developer Guide](#).

Storage management

Amazon S3 has storage management features that you can use to manage costs, meet regulatory requirements, reduce latency, and save multiple distinct copies of your data for compliance requirements.

- [S3 Lifecycle](#) – Configure a lifecycle policy to manage your objects and store them cost effectively throughout their lifecycle. You can transition objects to other S3 storage classes or expire objects that reach the end of their lifetimes.
- [S3 Object Lock](#) – Prevent Amazon S3 objects from being deleted or overwritten for a fixed amount of time or indefinitely. You can use Object Lock to help meet regulatory requirements that require *write-once-read-many (WORM)* storage or to simply add another layer of protection against object changes and deletions.

- [S3 Replication](#) – Replicate objects and their respective metadata and object tags to one or more destination buckets in the same or different AWS Regions for reduced latency, compliance, security, and other use cases.
- [S3 Batch Operations](#) – Manage billions of objects at scale with a single S3 API request or a few clicks in the Amazon S3 console. You can use Batch Operations to perform operations such as [Copy](#), [Invoke AWS Lambda function](#), and [Restore](#) on millions or billions of objects.

Access management

Amazon S3 provides features for auditing and managing access to your buckets and objects. By default, S3 buckets and the objects in them are private. You have access only to the S3 resources that you create. To grant granular resource permissions that support your specific use case or to audit the permissions of your Amazon S3 resources, you can use the following features.

- [S3 Block Public Access](#) – Block public access to S3 buckets and objects. By default, Block Public Access settings are turned on at the account and bucket level.
- [AWS Identity and Access Management \(IAM\)](#) – Create IAM users for your AWS account to manage access to your Amazon S3 resources. For example, you can use IAM with Amazon S3 to control the type of access a user or group of users has to an S3 bucket that your AWS account owns.
- [Bucket policies](#) – Use IAM-based policy language to configure resource-based permissions for your S3 buckets and the objects in them.
- [Amazon S3 access points](#) – Configure named network endpoints with dedicated access policies to manage data access at scale for shared datasets in Amazon S3.
- [Access control lists \(ACLs\)](#) – Grant read and write permissions for individual buckets and objects to authorized users. As a general rule, we recommend using S3 resource-based policies (bucket policies and access point policies) or IAM policies for access control instead of ACLs. ACLs are an access control mechanism that predates resource-based policies and IAM. For more information about when you'd use ACLs instead of resource-based policies or IAM policies, see [Access policy guidelines \(p. 400\)](#).
- [S3 Object Ownership](#) – Disable ACLs and take ownership of every object in your bucket, simplifying access management for data stored in Amazon S3. You, as the bucket owner, automatically own and have full control over every object in your bucket, and access control for your data is based on policies.
- [Access Analyzer for S3](#) – Evaluate and monitor your S3 bucket access policies, ensuring that the policies provide only the intended access to your S3 resources.

Data processing

To transform data and trigger workflows to automate a variety of other processing activities at scale, you can use the following features.

- [S3 Object Lambda](#) – Add your own code to S3 GET requests to modify and process data as it is returned to an application. Filter rows, dynamically resize images, redact confidential data, and much more.
- [Event notifications](#) – Trigger workflows that use Amazon Simple Notification Service (Amazon SNS), Amazon Simple Queue Service (Amazon SQS), and AWS Lambda when a change is made to your S3 resources.

Storage logging and monitoring

Amazon S3 provides logging and monitoring tools that you can use to monitor and control how your Amazon S3 resources are being used. For more information, see [Monitoring tools](#).

Automated monitoring tools

- [Amazon CloudWatch metrics for Amazon S3](#) – Track the operational health of your S3 resources and configure billing alerts when estimated charges reach a user-defined threshold.
- [AWS CloudTrail](#) – Record actions taken by a user, a role, or an AWS service in Amazon S3. CloudTrail logs provide you with detailed API tracking for S3 bucket-level and object-level operations.

Manual monitoring tools

- [Server access logging](#) – Get detailed records for the requests that are made to a bucket. You can use server access logs for many use cases, such as conducting security and access audits, learning about your customer base, and understanding your Amazon S3 bill.
- [AWS Trusted Advisor](#) – Evaluate your account by using AWS best practice checks to identify ways to optimize your AWS infrastructure, improve security and performance, reduce costs, and monitor service quotas. You can then follow the recommendations to optimize your services and resources.

Analytics and insights

Amazon S3 offers features to help you gain visibility into your storage usage, which empowers you to better understand, analyze, and optimize your storage at scale.

- [Amazon S3 Storage Lens](#) – Understand, analyze, and optimize your storage. S3 Storage Lens provides 29+ usage and activity metrics and interactive dashboards to aggregate data for your entire organization, specific accounts, AWS Regions, buckets, or prefixes.
- [Storage Class Analysis](#) – Analyze storage access patterns to decide when it's time to move data to a more cost-effective storage class.
- [S3 Inventory with Inventory reports](#) – Audit and report on objects and their corresponding metadata and configure other Amazon S3 features to take action in Inventory reports. For example, you can report on the replication and encryption status of your objects. For a list of all the metadata available for each object in Inventory reports, see [Amazon S3 Inventory list \(p. 740\)](#).

Strong consistency

Amazon S3 provides strong read-after-write consistency for PUT and DELETE requests of objects in your Amazon S3 bucket in all AWS Regions. This behavior applies to both writes of new objects as well as PUT requests that overwrite existing objects and DELETE requests. In addition, read operations on Amazon S3 Select, Amazon S3 access control lists (ACLs), Amazon S3 Object Tags, and object metadata (for example, the HEAD object) are strongly consistent. For more information, see [Amazon S3 data consistency model \(p. 6\)](#).

How Amazon S3 works

Amazon S3 is an object storage service that stores data as objects within buckets. An *object* is a file and any metadata that describes the file. A *bucket* is a container for objects.

To store your data in Amazon S3, you first create a bucket and specify a bucket name and AWS Region. Then, you upload your data to that bucket as objects in Amazon S3. Each object has a *key* (or *key name*), which is the unique identifier for the object within the bucket.

S3 provides features that you can configure to support your specific use case. For example, you can use S3 Versioning to keep multiple versions of an object in the same bucket, which allows you to restore objects that are accidentally deleted or overwritten.

Buckets and the objects in them are private and can be accessed only if you explicitly grant access permissions. You can use bucket policies, AWS Identity and Access Management (IAM) policies, access control lists (ACLs), and S3 Access Points to manage access.

Topics

- [Buckets \(p. 4\)](#)
- [Objects \(p. 4\)](#)
- [Keys \(p. 5\)](#)
- [S3 Versioning \(p. 5\)](#)
- [Version ID \(p. 5\)](#)
- [Bucket policy \(p. 5\)](#)
- [S3 Access Points \(p. 5\)](#)
- [Access control lists \(ACLs\) \(p. 6\)](#)
- [Regions \(p. 6\)](#)

Buckets

A bucket is a container for objects stored in Amazon S3. You can store any number of objects in a bucket and can have up to 100 buckets in your account. To request an increase, visit the [Service Quotas Console](#).

Every object is contained in a bucket. For example, if the object named `photos/puppy.jpg` is stored in the `DOC-EXAMPLE-BUCKET` bucket in the US West (Oregon) Region, then it is addressable using the URL `https://DOC-EXAMPLE-BUCKET.s3.us-west-2.amazonaws.com/photos/puppy.jpg`. For more information, see [Accessing a Bucket \(p. 125\)](#).

When you create a bucket, you enter a bucket name and choose the AWS Region where the bucket will reside. After you create a bucket, you cannot change the name of the bucket or its Region. Bucket names must follow the [bucket naming rules](#). You can also configure a bucket to use [S3 Versioning \(p. 638\)](#) or other [storage management](#) features.

Buckets also:

- Organize the Amazon S3 namespace at the highest level.
- Identify the account responsible for storage and data transfer charges.
- Provide access control options, such as bucket policies, access control lists (ACLs), and S3 Access Points, that you can use to manage access to your Amazon S3 resources.
- Serve as the unit of aggregation for usage reporting.

For more information about buckets, see [Buckets overview \(p. 114\)](#).

Objects

Objects are the fundamental entities stored in Amazon S3. Objects consist of object data and metadata. The metadata is a set of name-value pairs that describe the object. These pairs include some default metadata, such as the date last modified, and standard HTTP metadata, such as `Content-Type`. You can also specify custom metadata at the time that the object is stored.

An object is uniquely identified within a bucket by a [key \(name\) \(p. 5\)](#) and a [version ID \(p. 5\)](#) (if S3 Versioning is enabled on the bucket). For more information about objects, see [Amazon S3 objects overview \(p. 149\)](#).

Keys

An *object key* (or *key name*) is the unique identifier for an object within a bucket. Every object in a bucket has exactly one key. The combination of a bucket, object key, and optionally, version ID (if S3 Versioning is enabled for the bucket) uniquely identify each object. So you can think of Amazon S3 as a basic data map between "bucket + key + version" and the object itself.

Every object in Amazon S3 can be uniquely addressed through the combination of the web service endpoint, bucket name, key, and optionally, a version. For example, in the URL `https://DOC-EXAMPLE-BUCKET.s3.us-west-2.amazonaws.com/photos/puppy.jpg`, DOC-EXAMPLE-BUCKET is the name of the bucket and photos/puppy.jpg is the key.

For more information about object keys, see [Creating object key names \(p. 150\)](#).

S3 Versioning

You can use S3 Versioning to keep multiple variants of an object in the same bucket. With S3 Versioning, you can preserve, retrieve, and restore every version of every object stored in your buckets. You can easily recover from both unintended user actions and application failures.

For more information, see [Using versioning in S3 buckets \(p. 638\)](#).

Version ID

When you enable S3 Versioning in a bucket, Amazon S3 generates a unique version ID for each object added to the bucket. Objects that already existed in the bucket at the time that you enable versioning have a version ID of `null`. If you modify these (or any other) objects with other operations, such as [CopyObject](#) and [PutObject](#), the new objects get a unique version ID.

For more information, see [Using versioning in S3 buckets \(p. 638\)](#).

Bucket policy

A bucket policy is a resource-based AWS Identity and Access Management (IAM) policy that you can use to grant access permissions to your bucket and the objects in it. Only the bucket owner can associate a policy with a bucket. The permissions attached to the bucket apply to all of the objects in the bucket that are owned by the bucket owner. Bucket policies are limited to 20 KB in size.

Bucket policies use JSON-based access policy language that is standard across AWS. You can use bucket policies to add or deny permissions for the objects in a bucket. Bucket policies allow or deny requests based on the elements in the policy, including the requester, S3 actions, resources, and aspects or conditions of the request (for example, the IP address used to make the request). For example, you can create a bucket policy that grants cross-account permissions to upload objects to an S3 bucket while ensuring that the bucket owner has full control of the uploaded objects. For more information, see [Bucket policy examples \(p. 490\)](#).

In your bucket policy, you can use wildcard characters on Amazon Resource Names (ARNs) and other values to grant permissions to a subset of objects. For example, you can control access to groups of objects that begin with a common [prefix](#) or end with a given extension, such as `.html`.

S3 Access Points

Amazon S3 Access Points are named network endpoints with dedicated access policies that describe how data can be accessed using that endpoint. Access Points are attached to buckets that you can use to

perform S3 object operations, such as `GetObject` and `PutObject`. Access Points simplify managing data access at scale for shared datasets in Amazon S3.

Each access point has its own access point policy. You can configure [Block Public Access \(p. 584\)](#) settings for each access point. To restrict Amazon S3 data access to a private network, you can also configure any access point to accept requests only from a virtual private cloud (VPC).

For more information, see [Managing data access with Amazon S3 access points \(p. 301\)](#).

Access control lists (ACLs)

You can use ACLs to grant read and write permissions to authorized users for individual buckets and objects. Each bucket and object has an ACL attached to it as a subresource. The ACL defines which AWS accounts or groups are granted access and the type of access. ACLs are an access control mechanism that predates IAM. For more information about ACLs, see [Access control list \(ACL\) overview \(p. 554\)](#).

By default, when another AWS account uploads an object to your S3 bucket, that account (the object writer) owns the object, has access to it, and can grant other users access to it through ACLs. You can use Object Ownership to change this default behavior so that ACLs are disabled and you, as the bucket owner, automatically own every object in your bucket. As a result, access control for your data is based on policies, such as IAM policies, S3 bucket policies, virtual private cloud (VPC) endpoint policies, and AWS Organizations service control policies (SCPs).

A majority of modern use cases in Amazon S3 no longer require the use of ACLs, and we recommend that you disable ACLs except in unusual circumstances where you need to control access for each object individually. With Object Ownership, you can disable ACLs and rely on policies for access control. When you disable ACLs, you can easily maintain a bucket with objects uploaded by different AWS accounts. You, as the bucket owner, own all the objects in the bucket and can manage access to them using policies. For more information, see [Controlling ownership of objects and disabling ACLs for your bucket \(p. 601\)](#).

Regions

You can choose the geographical AWS Region where Amazon S3 stores the buckets that you create. You might choose a Region to optimize latency, minimize costs, or address regulatory requirements. Objects stored in an AWS Region never leave the Region unless you explicitly transfer or replicate them to another Region. For example, objects stored in the Europe (Ireland) Region never leave it.

Note

You can access Amazon S3 and its features only in the AWS Regions that are enabled for your account. For more information about enabling a Region to create and manage AWS resources, see [Managing AWS Regions in the AWS General Reference](#).

For a list of Amazon S3 Regions and endpoints, see [Regions and endpoints in the AWS General Reference](#).

Amazon S3 data consistency model

Amazon S3 provides strong read-after-write consistency for PUT and DELETE requests of objects in your Amazon S3 bucket in all AWS Regions. This behavior applies to both writes to new objects as well as PUT requests that overwrite existing objects and DELETE requests. In addition, read operations on Amazon S3 Select, Amazon S3 access controls lists (ACLs), Amazon S3 Object Tags, and object metadata (for example, the HEAD object) are strongly consistent.

Updates to a single key are atomic. For example, if you make a PUT request to an existing key from one thread and perform a GET request on the same key from a second thread concurrently, you will get either the old data or the new data, but never partial or corrupt data.

Amazon S3 achieves high availability by replicating data across multiple servers within AWS data centers. If a PUT request is successful, your data is safely stored. Any read (GET or LIST request) that is initiated following the receipt of a successful PUT response will return the data written by the PUT request. Here are examples of this behavior:

- A process writes a new object to Amazon S3 and immediately lists keys within its bucket. The new object appears in the list.
- A process replaces an existing object and immediately tries to read it. Amazon S3 returns the new data.
- A process deletes an existing object and immediately tries to read it. Amazon S3 does not return any data because the object has been deleted.
- A process deletes an existing object and immediately lists keys within its bucket. The object does not appear in the listing.

Note

- Amazon S3 does not support object locking for concurrent writers. If two PUT requests are simultaneously made to the same key, the request with the latest timestamp wins. If this is an issue, you must build an object-locking mechanism into your application.
- Updates are key-based. There is no way to make atomic updates across keys. For example, you cannot make the update of one key dependent on the update of another key unless you design this functionality into your application.

Bucket configurations have an eventual consistency model. Specifically, this means that:

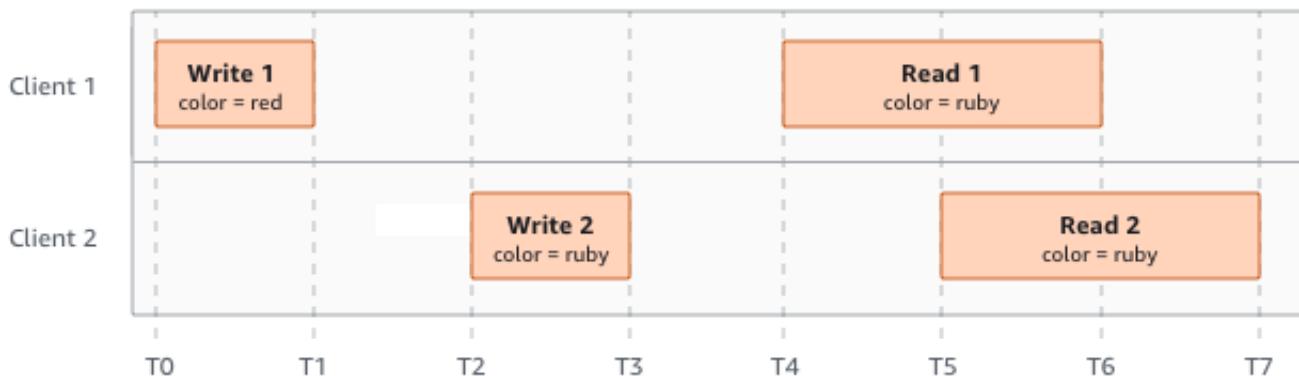
- If you delete a bucket and immediately list all buckets, the deleted bucket might still appear in the list.
- If you enable versioning on a bucket for the first time, it might take a short amount of time for the change to be fully propagated. We recommend that you wait for 15 minutes after enabling versioning before issuing write operations (PUT or DELETE requests) on objects in the bucket.

Concurrent applications

This section provides examples of behavior to be expected from Amazon S3 when multiple clients are writing to the same items.

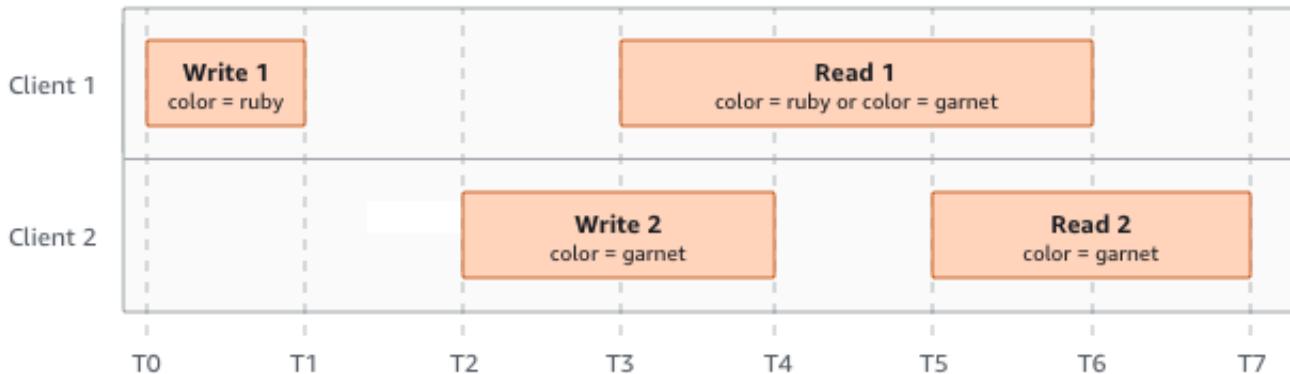
In this example, both W1 (write 1) and W2 (write 2) finish before the start of R1 (read 1) and R2 (read 2). Because S3 is strongly consistent, R1 and R2 both return color = ruby.

Domain = MyDomain, Item = StandardFez



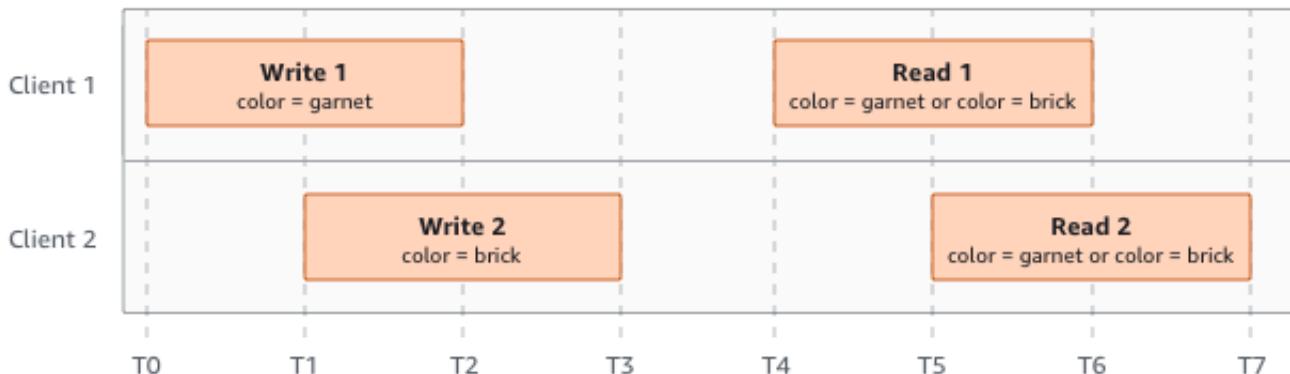
In the next example, W2 does not finish before the start of R1. Therefore, R1 might return `color = ruby` or `color = garnet`. However, because W1 and W2 finish before the start of R2, R2 returns `color = garnet`.

Domain = MyDomain, Item = StandardFez



In the last example, W2 begins before W1 has received an acknowledgement. Therefore, these writes are considered concurrent. Amazon S3 internally uses last-writer-wins semantics to determine which write takes precedence. However, the order in which Amazon S3 receives the requests and the order in which applications receive acknowledgements cannot be predicted because of various factors, such as network latency. For example, W2 might be initiated by an Amazon EC2 instance in the same Region, while W1 might be initiated by a host that is farther away. The best way to determine the final value is to perform a read after both writes have been acknowledged.

Domain = MyDomain, Item = StandardFez



Related services

After you load your data into Amazon S3, you can use it with other AWS services. The following are the services that you might use most frequently:

- **Amazon Elastic Compute Cloud (Amazon EC2)** – Provides secure and scalable computing capacity in the AWS Cloud. Using Amazon EC2 eliminates your need to invest in hardware up front, so you can develop and deploy applications faster. You can use Amazon EC2 to launch as many or as few virtual servers as you need, configure security and networking, and manage storage.

- **Amazon EMR** – Helps businesses, researchers, data analysts, and developers easily and cost-effectively process vast amounts of data. Amazon EMR uses a hosted Hadoop framework running on the web-scale infrastructure of Amazon EC2 and Amazon S3.
- **AWS Snow Family** – Helps customers that need to run operations in austere, non-data center environments, and in locations where there's a lack of consistent network connectivity. You can use AWS Snow Family devices to locally and cost-effectively access the storage and compute power of the AWS Cloud in places where an internet connection might not be an option.
- **AWS Transfer Family** – Provides fully managed support for file transfers directly into and out of Amazon S3 or Amazon Elastic File System (Amazon EFS) using Secure Shell (SSH) File Transfer Protocol (SFTP), File Transfer Protocol over SSL (FTPS), and File Transfer Protocol (FTP).

Accessing Amazon S3

You can work with Amazon S3 in any of the following ways:

AWS Management Console

The console is a web-based user interface for managing Amazon S3 and AWS resources. If you've signed up for an AWS account, you can access the Amazon S3 console by signing into the AWS Management Console and choosing **S3** from the AWS Management Console home page.

AWS Command Line Interface

You can use the AWS command line tools to issue commands or build scripts at your system's command line to perform AWS (including S3) tasks.

The [AWS Command Line Interface \(AWS CLI\)](#) provides commands for a broad set of AWS services. The AWS CLI is supported on Windows, macOS, and Linux. To get started, see the [AWS Command Line Interface User Guide](#). For more information about the commands for Amazon S3, see `s3api` and `s3control` in the [AWS CLI Command Reference](#).

AWS SDKs

AWS provides SDKs (software development kits) that consist of libraries and sample code for various programming languages and platforms (Java, Python, Ruby, .NET, iOS, Android, and so on). The AWS SDKs provide a convenient way to create programmatic access to S3 and AWS. Amazon S3 is a REST service. You can send requests to Amazon S3 using the AWS SDK libraries, which wrap the underlying Amazon S3 REST API and simplify your programming tasks. For example, the SDKs take care of tasks such as calculating signatures, cryptographically signing requests, managing errors, and retrying requests automatically. For information about the AWS SDKs, including how to download and install them, see [Tools for AWS](#).

Every interaction with Amazon S3 is either authenticated or anonymous. If you are using the AWS SDKs, the libraries compute the signature for authentication from the keys that you provide. For more information about how to make requests to Amazon S3, see [Making requests \(p. 1138\)](#).

Amazon S3 REST API

The architecture of Amazon S3 is designed to be programming language-neutral, using AWS-supported interfaces to store and retrieve objects. You can access S3 and AWS programmatically by using the Amazon S3 REST API. The REST API is an HTTP interface to Amazon S3. With the REST API, you use standard HTTP requests to create, fetch, and delete buckets and objects.

To use the REST API, you can use any toolkit that supports HTTP. You can even use a browser to fetch objects, as long as they are anonymously readable.

The REST API uses standard HTTP headers and status codes, so that standard browsers and toolkits work as expected. In some areas, we have added functionality to HTTP (for example, we added headers to support access control). In these cases, we have done our best to add the new functionality in a way that matches the style of standard HTTP usage.

If you make direct REST API calls in your application, you must write the code to compute the signature and add it to the request. For more information about how to make requests to Amazon S3, see [Making requests \(p. 1138\)](#).

Note

SOAP API support over HTTP is deprecated, but it is still available over HTTPS. Newer Amazon S3 features are not supported for SOAP. We recommend that you use either the REST API or the AWS SDKs.

Paying for Amazon S3

Pricing for Amazon S3 is designed so that you don't have to plan for the storage requirements of your application. Most storage providers require you to purchase a predetermined amount of storage and network transfer capacity. In this scenario, if you exceed that capacity, your service is shut off or you are charged high overage fees. If you do not exceed that capacity, you pay as though you used it all.

Amazon S3 charges you only for what you actually use, with no hidden fees and no overage charges. This model gives you a variable-cost service that can grow with your business while giving you the cost advantages of the AWS infrastructure. For more information, see [Amazon S3 Pricing](#).

When you sign up for AWS, your AWS account is automatically signed up for all services in AWS, including Amazon S3. However, you are charged only for the services that you use. If you are a new Amazon S3 customer, you can get started with Amazon S3 for free. For more information, see [AWS free tier](#).

To see your bill, go to the Billing and Cost Management Dashboard in the [AWS Billing and Cost Management console](#). To learn more about AWS account billing, see the [AWS Billing User Guide](#). If you have questions concerning AWS billing and AWS accounts, contact [AWS Support](#).

PCI DSS compliance

Amazon S3 supports the processing, storage, and transmission of credit card data by a merchant or service provider, and has been validated as being compliant with Payment Card Industry (PCI) Data Security Standard (DSS). For more information about PCI DSS, including how to request a copy of the AWS PCI Compliance Package, see [PCI DSS Level 1](#).

Getting started with Amazon S3

You can get started with Amazon S3 by working with buckets and objects. A *bucket* is a container for objects. An *object* is a file and any metadata that describes that file.

To store an object in Amazon S3, you create a bucket and then upload the object to the bucket. When the object is in the bucket, you can open it, download it, and move it. When you no longer need an object or a bucket, you can clean up your resources.

With Amazon S3, you pay only for what you use. For more information about Amazon S3 features and pricing, see [Amazon S3](#). If you are a new Amazon S3 customer, you can get started with Amazon S3 for free. For more information, see [AWS Free Tier](#).

Prerequisites

Before you begin, confirm that you've completed the steps in [Prerequisite: Setting up Amazon S3 \(p. 11\)](#).

Topics

- [Prerequisite: Setting up Amazon S3 \(p. 11\)](#)
- [Step 1: Create your first S3 bucket \(p. 13\)](#)
- [Step 2: Upload an object to your bucket \(p. 15\)](#)
- [Step 3: Download an object \(p. 15\)](#)
- [Step 4: Copy your object to a folder \(p. 16\)](#)
- [Step 5: Delete your objects and bucket \(p. 17\)](#)
- [Next steps \(p. 18\)](#)
- [Access control best practices \(p. 22\)](#)

Prerequisite: Setting up Amazon S3

When you sign up for AWS, your AWS account is automatically signed up for all services in AWS, including Amazon S3. You are charged only for the services that you use.

With Amazon S3, you pay only for what you use. For more information about Amazon S3 features and pricing, see [Amazon S3](#). If you are a new Amazon S3 customer, you can get started with Amazon S3 for free. For more information, see [AWS Free Tier](#).

To set up Amazon S3, use the steps in the following sections.

When you sign up for AWS and set up Amazon S3, you can optionally change the display language in the AWS Management Console. For more information, see [Changing the language of the AWS Management Console](#) in the [AWS Management Console Getting Started Guide](#).

Topics

- [Sign up for AWS \(p. 11\)](#)
- [Create an IAM user \(p. 12\)](#)
- [Sign in as an IAM user \(p. 13\)](#)

Sign up for AWS

If you do not have an AWS account, complete the following steps to create one.

To sign up for an AWS account

1. Open <https://portal.aws.amazon.com/billing/signup>.
2. Follow the online instructions.

Part of the sign-up procedure involves receiving a phone call and entering a verification code on the phone keypad.

AWS sends you a confirmation email after the sign-up process is complete. At any time, you can view your current account activity and manage your account by going to <https://aws.amazon.com/> and choosing **My Account**.

Create an IAM user

When you first create an AWS account, you begin with one sign-in identity that has complete access to all AWS services and resources in the account. This identity is called the AWS account *root user* and is accessed by signing in with the email address and password that you used to create the account.

Important

We strongly recommend that you do not use the root user for your everyday tasks. Safeguard your root user credentials and use them to perform the tasks that only the root user can perform. For the complete list of tasks that require you to sign in as the root user, see [Tasks that require root user credentials](#) in the *AWS General Reference*.

If you signed up for AWS but have not created an IAM user for yourself, follow these steps.

To create an administrator user for yourself and add the user to an administrators group (console)

1. Sign in to the [IAM console](#) as the account owner by choosing **Root user** and entering your AWS account email address. On the next page, enter your password.

Note

We strongly recommend that you adhere to the best practice of using the **Administrator** IAM user that follows and securely lock away the root user credentials. Sign in as the root user only to perform a few [account and service management tasks](#).

2. In the navigation pane, choose **Users** and then choose **Add users**.
3. For **User name**, enter **Administrator**.
4. Select the check box next to **AWS Management Console access**. Then select **Custom password**, and then enter your new password in the text box.
5. (Optional) By default, AWS requires the new user to create a new password when first signing in. You can clear the check box next to **User must create a new password at next sign-in** to allow the new user to reset their password after they sign in.
6. Choose **Next: Permissions**.
7. Under **Set permissions**, choose **Add user to group**.
8. Choose **Create group**.
9. In the **Create group** dialog box, for **Group name** enter **Administrators**.
10. Choose **Filter policies**, and then select **AWS managed - job function** to filter the table contents.
11. In the policy list, select the check box for **AdministratorAccess**. Then choose **Create group**.

Note

You must activate IAM user and role access to Billing before you can use the **AdministratorAccess** permissions to access the AWS Billing and Cost Management console. To do this, follow the instructions in [step 1 of the tutorial about delegating access to the billing console](#).

12. Back in the list of groups, select the check box for your new group. Choose **Refresh** if necessary to see the group in the list.
13. Choose **Next: Tags**.
14. (Optional) Add metadata to the user by attaching tags as key-value pairs. For more information about using tags in IAM, see [Tagging IAM entities](#) in the *IAM User Guide*.
15. Choose **Next: Review** to see the list of group memberships to be added to the new user. When you are ready to proceed, choose **Create user**.

You can use this same process to create more groups and users and to give your users access to your AWS account resources. To learn about using policies that restrict user permissions to specific AWS resources, see [Access management](#) and [Example policies](#).

Sign in as an IAM user

After you create an IAM user, you can sign in to AWS with your IAM user name and password.

Before you sign in as an IAM user, you can verify the sign-in link for IAM users in the IAM console. On the IAM Dashboard, under **IAM users sign-in link**, you can see the sign-in link for your AWS account. The URL for your sign-in link contains your AWS account ID without dashes (-).

If you don't want the URL for your sign-in link to contain your AWS account ID, you can create an account alias. For more information, see [Creating, deleting, and listing an AWS account alias](#) in the *IAM User Guide*.

To sign in as an AWS user

1. Sign out of the AWS Management Console.
2. Enter your sign-in link.

Your sign-in link includes your AWS account ID (without dashes) or your AWS account alias:

```
https://aws_account_id_or_alias.signin.aws.amazon.com/console
```

3. Enter the IAM user name and password that you just created.

When you're signed in, the navigation bar displays "your_user_name @ your_aws_account_id".

Step 1: Create your first S3 bucket

After you sign up for AWS, you're ready to create a bucket in Amazon S3 using the AWS Management Console. Every object in Amazon S3 is stored in a *bucket*. Before you can store data in Amazon S3, you must create a bucket.

Note

You are not charged for creating a bucket. You are charged only for storing objects in the bucket and for transferring objects in and out of the bucket. The charges that you incur through following the examples in this guide are minimal (less than \$1). For more information about storage charges, see [Amazon S3 pricing](#).

1. Sign in to the AWS Management Console and open the Amazon S3 console at <https://console.aws.amazon.com/s3/>.
 2. Choose **Create bucket**.
- The **Create bucket** wizard opens.
3. In **Bucket name**, enter a DNS-compliant name for your bucket.

The bucket name must:

- Be unique across all of Amazon S3.
- Be between 3 and 63 characters long.
- Not contain uppercase characters.
- Start with a lowercase letter or number.

After you create the bucket, you cannot change its name. For information about naming buckets, see [Bucket naming rules \(p. 118\)](#).

Important

Avoid including sensitive information, such as account number, in the bucket name. The bucket name is visible in the URLs that point to the objects in the bucket.

4. In **Region**, choose the AWS Region where you want the bucket to reside.

Choose a Region close to you to minimize latency and costs and address regulatory requirements. Objects stored in a Region never leave that Region unless you explicitly transfer them to another Region. For a list of Amazon S3 AWS Regions, see [AWS service endpoints](#) in the *Amazon Web Services General Reference*.

5. Under **Object Ownership**, to disable or enable ACLs and control ownership of objects uploaded in your bucket, choose one of the following settings:

ACLs disabled

- **Bucket owner enforced** – ACLs are disabled, and the bucket owner automatically owns and has full control over every object in the bucket. ACLs no longer affect permissions to data in the S3 bucket. The bucket uses policies to define access control.

To require that all new buckets are created with ACLs disabled by using IAM or AWS Organizations policies, see [Disabling ACLs for all new buckets \(bucket owner enforced\) \(p. 622\)](#).

ACLs enabled

- **Bucket owner preferred** – The bucket owner owns and has full control over new objects that other accounts write to the bucket with the `bucket-owner-full-control` canned ACL.

If you apply the bucket owner preferred setting, to require all Amazon S3 uploads to include the `bucket-owner-full-control` canned ACL, you can [add a bucket policy \(p. 623\)](#) that only allows object uploads that use this ACL.

- **Object writer** – The AWS account that uploads an object owns the object, has full control over it, and can grant other users access to it through ACLs.

Note

To apply the **Bucket owner enforced** setting or the **Bucket owner preferred** setting, you must have the following permission: `s3:CreateBucket` and `s3:PutBucketOwnershipControls`.

6. In **Bucket settings for Block Public Access**, choose the Block Public Access settings that you want to apply to the bucket.

We recommend that you keep all settings enabled unless you know that you need to turn off one or more of them for your use case, such as to host a public website. Block Public Access settings that you enable for the bucket are also enabled for all access points that you create on the bucket. For more information about blocking public access, see [Blocking public access to your Amazon S3 storage \(p. 584\)](#).

7. (Optional) If you want to enable S3 Object Lock, do the following:

- a. Choose **Advanced settings**.

Important

You can only enable S3 Object Lock for a bucket when you create it. If you enable Object Lock for the bucket, you cannot disable it later. Enabling Object Lock also enables versioning for the bucket. After you enable Object Lock for the bucket, you must configure the Object Lock default retention and legal hold settings to protect new objects from being deleted or overwritten. For more information, see [Configuring S3 Object Lock using the console \(p. 684\)](#).

- b. If you want to enable Object Lock, choose **Enable**, read the warning that appears, and acknowledge it.

For more information about the S3 Object Lock feature, see [Using S3 Object Lock \(p. 680\)](#).

Note

To create an Object Lock enabled bucket, you must have the following permissions: s3:CreateBucket, s3:PutBucketVersioning and s3:PutBucketObjectLockConfiguration.

8. Choose **Create bucket**.

You've created a bucket in Amazon S3.

Next step

To add an object to your bucket, see [Step 2: Upload an object to your bucket \(p. 15\)](#).

Step 2: Upload an object to your bucket

After creating a bucket in Amazon S3, you're ready to upload an object to the bucket. An object can be any kind of file: a text file, a photo, a video, and so on.

To upload an object to a bucket

1. Open the Amazon S3 console at <https://console.aws.amazon.com/s3/>.
2. In the **Buckets** list, choose the name of the bucket that you want to upload your object to.
3. On the **Objects** tab for your bucket, choose **Upload**.
4. Under **Files and folders**, choose **Add files**.
5. Choose a file to upload, and then choose **Open**.
6. Choose **Upload**.

You've successfully uploaded an object to your bucket.

Next step

To view your object, see [Step 3: Download an object \(p. 15\)](#).

Step 3: Download an object

After you upload an object to a bucket, you can view information about your object and download the object to your local computer.

Using the S3 console

This section explains how to use the Amazon S3 console to download an object from an S3 bucket using a presigned URL.

Note

- You can only download one object at a time.
- Objects with key names ending with period(s) "." downloaded using the Amazon S3 console will have the period(s) "." removed from the key name of the downloaded object. To download an object with the key name ending in period(s) "." retained in the downloaded object, you must use the AWS Command Line Interface (AWS CLI), AWS SDKs, or REST API. For AWS CLI, REST API, and AWS SDK information and examples, see [Downloading an object](#).

To download an object from an S3 bucket

1. Sign in to the AWS Management Console and open the Amazon S3 console at <https://console.aws.amazon.com/s3/>.
2. In the **Buckets** list, choose the name of the bucket that you want to download an object from.
3. You can download an object from an S3 bucket in any of the following ways:
 - Select the object and choose **Download** or choose **Download as** from the **Actions** menu if you want to download the object to a specific folder.
 - If you want to download a specific version of the object, select the **Show versions** button. Select the version of the object that you want and choose **Download** or choose **Download as** from the **Actions** menu if you want to download the object to a specific folder.

You've successfully downloaded your object.

Next step

To copy and paste your object within Amazon S3, see [Step 4: Copy your object to a folder \(p. 16\)](#).

Step 4: Copy your object to a folder

You've already added an object to a bucket and downloaded the object. Now, you create a folder and copy the object and paste it into the folder.

To copy an object to a folder

1. In the **Buckets** list, choose your bucket name.
2. Choose **Create folder** and configure a new folder:
 - a. Enter a folder name (for example, favorite-pics).
 - b. For the folder encryption setting, choose **Disable**.
 - c. Choose **Save**.
3. Navigate to the Amazon S3 bucket or folder that contains the objects that you want to copy.
4. Select the check box to the left of the names of the objects that you want to copy.
5. Choose **Actions** and choose **Copy** from the list of options that appears.

Alternatively, choose **Copy** from the options in the upper right.

6. Choose the destination folder:

- a. Choose **Browse S3**.
- b. Choose the option button to the left of the folder name.

To navigate into a folder and choose a subfolder as your destination, choose the folder name.

- c. Choose **Choose destination**.

The path to your destination folder appears in the **Destination** box. In **Destination**, you can alternately enter your destination path, for example, `s3://bucket-name/folder-name/`.

7. In the bottom right, choose **Copy**.

Amazon S3 moves your objects to the destination folder.

Next step

To delete an object and a bucket in Amazon S3, see [Step 5: Delete your objects and bucket \(p. 17\)](#).

Step 5: Delete your objects and bucket

When you no longer need an object or a bucket, we recommend that you delete them to prevent further charges. If you completed this getting started walkthrough as a learning exercise, and you don't plan to use your bucket or objects, we recommend that you delete your bucket and objects so that charges no longer accrue.

Before you delete your bucket, empty the bucket or delete the objects in the bucket. After you delete your objects and bucket, they are no longer available.

If you want to continue to use the same bucket name, we recommend that you delete the objects or empty the bucket, but don't delete the bucket. After you delete a bucket, the name becomes available to reuse. However, another AWS account might create a bucket with the same name before you have a chance to reuse it.

Topics

- [Deleting an object \(p. 17\)](#)
- [Emptying your bucket \(p. 18\)](#)
- [Deleting your bucket \(p. 18\)](#)

Deleting an object

If you want to choose which objects you delete without emptying all the objects from your bucket, you can delete an object.

1. In the **Buckets** list, choose the name of the bucket that you want to delete an object from.
2. Select the check box to the left of the names of the objects that you want to delete.
3. Choose **Actions** and choose **Delete** from the list of options that appears.

Alternatively, choose **Delete** from the options in the upper right.

4. Type **permanently delete** if asked to confirm that you want to delete these objects.

5. Choose **Delete objects** in the bottom right and Amazon S3 deletes the specified objects.

Emptying your bucket

If you plan to delete your bucket, you must first empty your bucket, which deletes all the objects in the bucket.

To empty a bucket

1. In the **Buckets** list, select the bucket that you want to empty, and then choose **Empty**.
2. To confirm that you want to empty the bucket and delete all the objects in it, in **Empty bucket**, type **permanently delete**.

Important

Emptying the bucket cannot be undone. Objects added to the bucket while the empty bucket action is in progress will be deleted.

3. To empty the bucket and delete all the objects in it, and choose **Empty**.

An **Empty bucket: Status** page opens that you can use to review a summary of failed and successful object deletions.

4. To return to your bucket list, choose **Exit**.

Deleting your bucket

After you empty your bucket or delete all the objects from your bucket, you can delete your bucket.

1. To delete a bucket, in the **Buckets** list, select the bucket.
2. Choose **Delete**.
3. To confirm deletion, in **Delete bucket**, type the name of the bucket.

Important

Deleting a bucket cannot be undone. Bucket names are unique. If you delete your bucket, another AWS user can use the name. If you want to continue to use the same bucket name, don't delete your bucket. Instead, empty and keep the bucket.

4. To delete your bucket, choose **Delete bucket**.

Next steps

In the preceding examples, you learned how to perform some basic Amazon S3 tasks.

The following topics explain the learning paths that you can use to gain a deeper understanding of Amazon S3 so that you can implement it in your applications.

Topics

- [Understand common use cases \(p. 19\)](#)
- [Control access to your buckets and objects \(p. 19\)](#)
- [Explore training and support \(p. 19\)](#)
- [Manage and monitor your storage \(p. 20\)](#)
- [Develop with Amazon S3 \(p. 20\)](#)

Understand common use cases

You can use Amazon S3 to support your specific use case. The [AWS Solutions Library](#) and [AWS Blog](#) provide use-case specific information and tutorials. The following are some common use cases for Amazon S3:

- **Backup and storage** – Use Amazon S3 storage management features to manage costs, meet regulatory requirements, reduce latency, and save multiple distinct copies of your data for compliance requirements.
- **Application hosting** – Deploy, install, and manage web applications that are reliable, highly scalable, and low-cost. For example, you can configure your Amazon S3 bucket to host a static website. For more information, see [Hosting a static website using Amazon S3 \(p. 116\)](#).
- **Media hosting** – Build a highly available infrastructure that hosts video, photo, or music uploads and downloads.
- **Software delivery** – Host your software applications for customers to download.

Control access to your buckets and objects

Amazon S3 provides a variety of security features and tools. For an overview, see [Access control best practices \(p. 22\)](#).

By default, S3 buckets and the objects in them are private. You have access only to the S3 resources that you create. You can use the following features to grant granular resource permissions that support your specific use case or to audit the permissions of your Amazon S3 resources.

- [S3 Block Public Access](#) – Block public access to S3 buckets and objects. By default, Block Public Access settings are turned on at the account and bucket level.
- [AWS Identity and Access Management \(IAM\)](#) – Create IAM users for your AWS account to manage access to your Amazon S3 resources. For example, you can use IAM with Amazon S3 to control the type of access a user or group of users has to an Amazon S3 bucket that your AWS account owns.
- [Bucket policies](#) – Use IAM-based policy language to configure resource-based permissions for your S3 buckets and the objects in them.
- [Access control lists \(ACLs\)](#) – Grant read and write permissions for individual buckets and objects to authorized users. As a general rule, we recommend using S3 resource-based policies (bucket policies and access point policies) or IAM policies for access control instead of ACLs. ACLs are an access control mechanism that predates resource-based policies and IAM. For more information about when you'd use ACLs instead of resource-based policies or IAM policies, see [Access policy guidelines \(p. 400\)](#).
- [S3 Object Ownership](#) – Disable ACLs and take ownership of every object in your bucket, simplifying access management for data stored in Amazon S3. You, as the bucket owner, automatically own and have full control over every object in your bucket, and access control for your data is based on policies.
- [Access Analyzer for S3](#) – Evaluate and monitor your S3 bucket access policies, ensuring that the policies provide only the intended access to your S3 resources.

Explore training and support

You can learn from AWS experts to advance your skills and get expert assistance achieving your objectives.

- **Training** – Training resources provide a hands-on approach to learning Amazon S3. For more information, see [AWS training and certification](#) and [AWS online tech talks](#).
- **Discussion Forums** – On the forum, you can review posts to understand what you can and can't do with Amazon S3. You can also post your questions. For more information, see [Discussion Forums](#).

- **Technical Support** – If you have further questions, you can contact [Technical Support](#).

Manage and monitor your storage

- [Managing your storage \(p. 638\)](#) – After you create buckets and upload objects in Amazon S3, you can manage your object storage. For example, you can use S3 Versioning and Amazon S3 Replication for disaster recovery, S3 Lifecycle to manage storage costs, and S3 Object Lock to meet compliance requirements.
- [Monitoring your storage \(p. 959\)](#) – Monitoring is an important part of maintaining the reliability, availability, and performance of Amazon S3 and your AWS solutions. You can monitor storage activity and costs. Also, we recommend that you collect monitoring data from all the parts of your AWS solution so that you can more easily debug a multipoint failure if one occurs.
- [Analytics and insights \(p. 1048\)](#) – You can also use analytics and insights in Amazon S3 to understand, analyze, and optimize your storage usage. For example, use [Amazon S3 Storage Lens \(p. 1053\)](#) to understand, analyze, and optimize your storage. S3 Storage Lens provides 29+ usage and activity metrics and interactive dashboards to aggregate data for your entire organization, specific accounts, Regions, buckets, or prefixes. Use [Storage Class Analysis \(p. 1048\)](#) to analyze storage access patterns to decide when it's time to move your data to a more cost-effective storage class.

Develop with Amazon S3

Amazon S3 is a REST service. You can send requests to Amazon S3 using the REST API or the AWS SDK libraries, which wrap the underlying Amazon S3 REST API, simplifying your programming tasks. You can also use the AWS Command Line Interface (AWS CLI) to make Amazon S3 API calls. For more information, see [Making requests \(p. 1138\)](#).

The Amazon S3 REST API is an HTTP interface to Amazon S3. With the REST API, you use standard HTTP requests to create, fetch, and delete buckets and objects. To use the REST API, you can use any toolkit that supports HTTP. You can even use a browser to fetch objects, as long as they are anonymously readable. For more information, see [Developing with Amazon S3 using the REST API \(p. 1197\)](#).

To help you build applications using the language of your choice, we provide the following resources.

AWS CLI

You can access the features of Amazon S3 using the AWS CLI. To download and configure the AWS CLI, see [Developing with Amazon S3 using the AWS CLI \(p. 1183\)](#).

The AWS CLI provides two tiers of commands for accessing Amazon S3: High-level (`s3`) commands and API-level (`s3api` and `s3control`) commands. The high-level S3 commands simplify performing common tasks, such as creating, manipulating, and deleting objects and buckets. The `s3api` and `s3control` commands expose direct access to all Amazon S3 API operations, which you can use to carry out advanced operations that might not be possible with the high-level commands alone.

For a list of Amazon S3 AWS CLI commands, see `s3`, `s3api`, and `s3control`.

AWS SDKs and Explorers

You can use the AWS SDKs when developing applications with Amazon S3. The AWS SDKs simplify your programming tasks by wrapping the underlying REST API. The AWS Mobile SDKs and the Amplify JavaScript library are also available for building connected mobile and web applications using AWS.

In addition to the AWS SDKs, AWS Explorers are available for Visual Studio and Eclipse for Java IDE. In this case, the SDKs and the explorers are bundled together as AWS Toolkits.

For more information, see [Developing with Amazon S3 using the AWS SDKs, and explorers \(p. 1184\)](#).

Sample Code and Libraries

The [AWS Developer Center](#) and [AWS Code Sample Catalog](#) have sample code and libraries written especially for Amazon S3. You can use these code samples to understand how to implement the Amazon S3 API. You can also view the [Amazon Simple Storage Service API Reference](#) to understand the Amazon S3 API operations in detail.

Access control best practices

Amazon S3 provides a variety of security features and tools. The following scenarios should serve as a guide to what tools and settings you might want to use when performing certain tasks or operating in specific environments. Proper application of these tools can help maintain the integrity of your data and help ensure that your resources are accessible to the intended users.

Topics

- [Creating a new bucket \(p. 22\)](#)
- [Storing and sharing data \(p. 23\)](#)
- [Sharing resources \(p. 24\)](#)
- [Protecting data \(p. 24\)](#)

Creating a new bucket

When creating a new bucket, you should apply the following tools and settings to help ensure that your Amazon S3 resources are protected.

S3 Object Ownership for simplifying access control

S3 Object Ownership is an Amazon S3 bucket-level setting that you can use to disable access control lists (ACLs) and take ownership of every object in your bucket, simplifying access management for data stored in Amazon S3. By default, when another AWS account uploads an object to your S3 bucket, that account (the object writer) owns the object, has access to it, and can grant other users access to it through ACLs. You can use Object Ownership to change this default behavior so that ACLs are disabled and you, as the bucket owner, automatically own every object in your bucket. As a result, access control for your data is based on policies, such as IAM policies, S3 bucket policies, virtual private cloud (VPC) endpoint policies, and AWS Organizations service control policies (SCPs).

Object Ownership has three settings that you can use to control ownership of objects uploaded in your bucket and disable or enable ACLs:

ACLs disabled

- **Bucket owner enforced (recommended)** – ACLs are disabled, and the bucket owner automatically owns and has full control over every object in the bucket. ACLs no longer affect permissions to data in the S3 bucket. The bucket uses policies exclusively to define access control.

ACLs enabled

- **Bucket owner preferred** – The bucket owner owns and has full control over new objects that other accounts write to the bucket with the `bucket-owner-full-control` canned ACL.
- **Object writer (default)** – The AWS account that uploads an object owns the object, has full control over it, and can grant other users access to it through ACLs.

For more information, see [Controlling ownership of objects and disabling ACLs for your bucket \(p. 601\)](#).

Block Public Access

S3 Block Public Access provides four settings to help you avoid inadvertently exposing your S3 resources. You can apply these settings in any combination to individual access points, buckets, or entire AWS accounts. If you apply a setting to an account, it applies to all buckets and access points that are owned by that account. By default, the **Block all public access** setting is applied to new buckets created in the Amazon S3 console.

For more information, see [The meaning of "public" \(p. 586\)](#).

If the S3 Block Public Access settings are too restrictive, you can use AWS Identity and Access Management (IAM) identities to grant access to specific users rather than disabling all Block Public Access settings. Using Block Public Access with IAM identities helps ensure that any operation that is blocked by a Block Public Access setting is rejected unless the requesting user has been given specific permission.

For more information, see [Block public access settings \(p. 585\)](#).

Grant access with IAM identities

When setting up accounts for new team members who require S3 access, use IAM users and roles to ensure least privileges. You can also implement a form of IAM multi-factor authentication (MFA) to support a strong identity foundation. Using IAM identities, you can grant unique permissions to users and specify what resources they can access and what actions they can take. IAM identities provide increased capabilities, including the ability to require users to enter login credentials before accessing shared resources and apply permission hierarchies to different objects within a single bucket.

For more information, see [Example 1: Bucket owner granting its users bucket permissions \(p. 527\)](#).

Bucket policies

With bucket policies, you can personalize bucket access to help ensure that only those users you have approved can access resources and perform actions within them. In addition to bucket policies, you should use bucket-level Block Public Access settings to further limit public access to your data.

For more information, see [Using bucket policies \(p. 487\)](#).

When creating policies, avoid the use of wildcard characters in the `Principal` element because it effectively allows anyone to access your Amazon S3 resources. It's better to explicitly list users or groups that are allowed to access the bucket. Rather than including a wildcard for their actions, grant them specific permissions when applicable.

To further maintain the practice of least privileges, `Deny` statements in the `Effect` element should be as broad as possible and `Allow` statements should be as narrow as possible. `Deny` effects paired with the `"s3 : "*` action are another good way to implement opt-in best practices for the users included in policy condition statements.

For more information about specifying conditions for when a policy is in effect, see [Amazon S3 condition key examples \(p. 420\)](#).

Buckets in a VPC setting

When adding users in a corporate setting, you can use a virtual private cloud (VPC) endpoint to allow any users in your virtual network to access your Amazon S3 resources. VPC endpoints enable developers to provide specific access and permissions to groups of users based on the network the user is connected to. Rather than adding each user to an IAM role or group, you can use VPC endpoints to deny bucket access if the request doesn't originate from the specified endpoint.

For more information, see [Controlling access from VPC endpoints with bucket policies \(p. 489\)](#).

Storing and sharing data

Use the following tools and best practices to store and share your Amazon S3 data.

Versioning and Object Lock for data integrity

If you use the Amazon S3 console to manage buckets and objects, you should implement S3 Versioning and S3 Object Lock. These features help prevent accidental changes to critical data and enable you to

roll back unintended actions. This capability is particularly useful when there are multiple users with full write and execute permissions accessing the Amazon S3 console.

For information about S3 Versioning, see [Using versioning in S3 buckets \(p. 638\)](#). For information about Object Lock, see [Using S3 Object Lock \(p. 680\)](#).

Object lifecycle management for cost efficiency

To manage your objects so that they are stored cost effectively throughout their lifecycle, you can pair lifecycle policies with object versioning. Lifecycle policies define actions that you want S3 to take during an object's lifetime. For example, you can create a lifecycle policy that will transition objects to another storage class, archive them, or delete them after a specified period of time. You can define a lifecycle policy for all objects or a subset of objects in the bucket by using a shared prefix or tag.

For more information, see [Managing your storage lifecycle \(p. 701\)](#).

Cross-Region Replication for multiple office locations

When creating buckets that are accessed by different office locations, you should consider implementing S3 Cross-Region Replication. Cross-Region Replication helps ensure that all users have access to the resources they need and increases operational efficiency. Cross-Region Replication offers increased availability by copying objects across S3 buckets in different AWS Regions. However, the use of this tool increases storage costs.

For more information, see [Replicating objects \(p. 753\)](#).

Permissions for secure static website hosting

When configuring a bucket to be used as a publicly accessed static website, you must disable all Block Public Access settings. It is important to only provide `s3:GetObject` actions and not `ListObject` or `PutObject` permissions when writing the bucket policy for your static website. This helps ensure that users cannot view all the objects in your bucket or add their own content.

For more information, see [Setting permissions for website access \(p. 1126\)](#).

Amazon CloudFront provides the capabilities required to set up a secure static website. Amazon S3 static websites only support HTTP endpoints. CloudFront uses the durable storage of Amazon S3 while providing additional security headers like HTTPS. HTTPS adds security by encrypting a normal HTTP request and protecting against common cyberattacks.

For more information, see [Getting started with a secure static website](#) in the *Amazon CloudFront Developer Guide*.

Sharing resources

There are several different ways that you can share resources with a specific group of users. You can use the following tools to share a set of documents or other resources to a single group of users, department, or an office. Although they can all be used to accomplish the same goal, some tools might pair better than others with your existing settings.

S3 Object Ownership

S3 Object Ownership is an Amazon S3 bucket-level setting that you can use to disable ACLs and take ownership of every object in your bucket, simplifying access management for data stored in Amazon S3. By default, when another AWS account uploads an object to your S3 bucket, that account (the object writer) owns the object, has access to it, and can grant other users access to it through ACLs. You can use Object Ownership to change this default behavior so that ACLs are disabled and you, as the bucket owner, automatically own every object in your bucket. As a result, access control for your data is based on policies. For more information, see [Controlling ownership of objects and disabling ACLs for your bucket \(p. 601\)](#).

User policies

You can share resources with a limited group of people using IAM groups and user policies. When creating a new IAM user, you are prompted to create and add them to a group. However, you can create and add users to groups at any point. If the individuals you intend to share these resources with are already set up within IAM, you can add them to a common group and share the bucket with their group within the user policy. You can also use IAM user policies to share individual objects within a bucket.

For more information, see [Allowing an IAM user access to one of your buckets \(p. 516\)](#).

Access control lists

As a general rule, we recommend that you use S3 bucket policies or IAM policies for access control. Amazon S3 ACLs are the original access control mechanism in Amazon S3 that predates IAM. If you already use S3 ACLs and you find them sufficient, there is no need to change. However, certain access control scenarios require the use of ACLs. For example, when a bucket owner wants to grant permission to objects, but not all objects are owned by the bucket owner, the object owner must first grant permission to the bucket owner. This is done using an object ACL.

A majority of modern use cases in Amazon S3 no longer require the use of ACLs, and we recommend that you disable ACLs except in unusual circumstances where you must control access for each object individually. With Object Ownership, you can disable ACLs and rely on policies for access control. When you disable ACLs, you can easily maintain a bucket with objects uploaded by different AWS accounts. You, as the bucket owner, own all the objects in the bucket and can manage access to them using policies.

Important

If your bucket uses the bucket owner enforced setting for S3 Object Ownership, you must use policies to grant access to your bucket and the objects in it. Requests to set ACLs or update ACLs fail and return the `AccessControlListNotSupported` error code. Requests to read ACLs are still supported.

For more information about using ACLs, see [Example 3: Bucket owner granting permissions to objects it does not own \(p. 536\)](#).

Prefixes

When trying to share specific resources from a bucket, you can replicate folder-level permissions using prefixes. The Amazon S3 console supports the folder concept as a means of grouping objects by using a shared name prefix for objects. You can then specify a prefix within the conditions of an IAM user's policy to grant them explicit permission to access the resources associated with that prefix.

For more information, see [Organizing objects in the Amazon S3 console using folders \(p. 254\)](#).

Tagging

If you use object tagging to categorize storage, you can share objects that have been tagged with a specific value with specified users. Resource tagging allows you to control access to objects based on the tags associated with the resource that a user is trying to access. To do this, use the `ResourceTag/key-name` condition within an IAM user policy to allow access to the tagged resources.

For more information, see [Controlling access to AWS resources using resource tags in the IAM User Guide](#).

Protecting data

Use the following tools to help protect data in transit and at rest, both of which are crucial in maintaining the integrity and accessibility of your data.

Object encryption

Amazon S3 offers several object encryption options that protect data in transit and at rest. Server-side encryption encrypts your object before saving it on disks in its data centers and then decrypts it when you download the objects. As long as you authenticate your request and you have access permissions, there is no difference in the way you access encrypted or unencrypted objects. When setting up server-side encryption, you have three mutually exclusive options:

- 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)

For more information, see [Protecting data using server-side encryption \(p. 338\)](#).

Client-side encryption is the act of encrypting data before sending it to Amazon S3. For more information, see [Protecting data using client-side encryption \(p. 381\)](#).

Signing methods

Signature Version 4 is the process of adding authentication information to AWS requests sent by HTTP. For security, most requests to AWS must be signed with an access key, which consists of an access key ID and secret access key. These two keys are commonly referred to as your security credentials.

For more information, see [Authenticating Requests \(AWS Signature Version 4\)](#) and [Signature Version 4 signing process](#).

Logging and monitoring

Monitoring is an important part of maintaining the reliability, availability, and performance of your Amazon S3 solutions so that you can more easily debug a multi-point failure if one occurs. Logging can provide insight into any errors users are receiving, and when and what requests are made. AWS provides several tools for monitoring your Amazon S3 resources:

- Amazon CloudWatch
- AWS CloudTrail
- Amazon S3 Access Logs
- AWS Trusted Advisor

For more information, see [Logging and monitoring in Amazon S3 \(p. 626\)](#).

Amazon S3 is integrated with AWS CloudTrail, a service that provides a record of actions taken by a user, a role, or an AWS service in Amazon S3. This feature can be paired with Amazon GuardDuty, which monitors threats against your Amazon S3 resources by analyzing CloudTrail management events and CloudTrail S3 data events. These data sources monitor different kinds of activity. For example, S3 related CloudTrail management events include operations that list or configure S3 projects. GuardDuty analyzes S3 data events from all of your S3 buckets and monitors them for malicious and suspicious activity.

For more information, see [Amazon S3 protection in Amazon GuardDuty](#) in the *Amazon GuardDuty User Guide*.

Tutorials

The following tutorials present complete end-to-end procedures for common Amazon S3 tasks. These tutorials are intended for a lab-type environment, and they use fictitious company names, user names, and so on. Their purpose is to provide general guidance. They are not intended for direct use in a production environment without careful review and adaptation to meet the unique needs of your organization's environment.

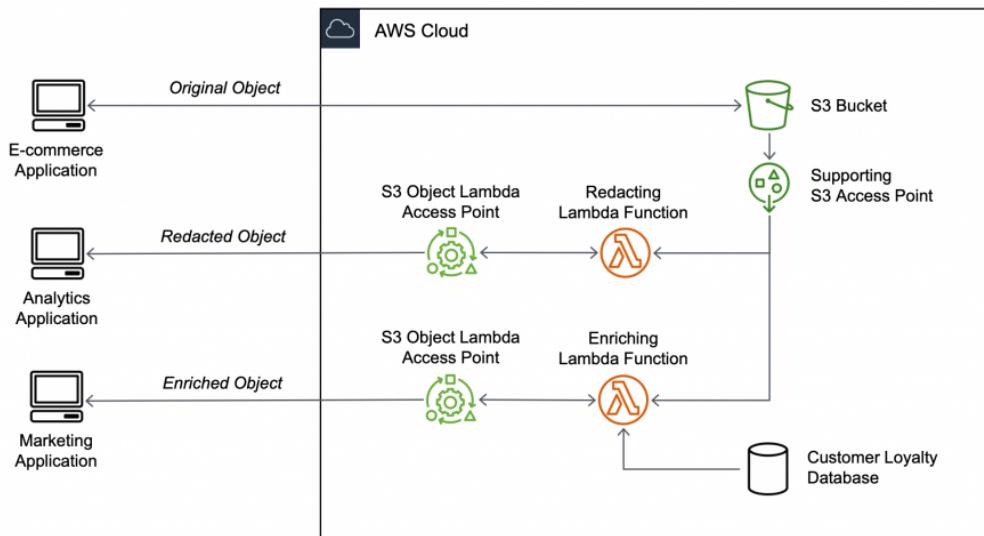
Topics

- [Tutorial: Transforming data for your application with S3 Object Lambda \(p. 27\)](#)
- [Tutorial: Detecting and redacting PII data with S3 Object Lambda and Amazon Comprehend \(p. 41\)](#)
- [Tutorial: Hosting on-demand streaming video with Amazon S3, Amazon CloudFront, and Amazon Route 53 \(p. 51\)](#)
- [Tutorial: Batch-transcoding videos with S3 Batch Operations, AWS Lambda, and AWS Elemental MediaConvert \(p. 64\)](#)
- [Tutorial: Configuring a static website on Amazon S3 \(p. 92\)](#)
- [Configuring a static website using a custom domain registered with Route 53 \(p. 97\)](#)

Tutorial: Transforming data for your application with S3 Object Lambda

When you store data in Amazon S3, you can easily share it for use by multiple applications. However, each application might have unique data format requirements, and might need modification or processing of your data for a specific use case. For example, a dataset created by an ecommerce application might include personally identifiable information (PII). When the same data is processed for analytics, this PII is not needed and should be redacted. However, if the same dataset is used for a marketing campaign, you might need to enrich the data with additional details, such as information from the customer loyalty database.

With [S3 Object Lambda](#), you can add your own code to process data retrieved from S3 before returning it to an application. Specifically, you can configure an AWS Lambda function and attach it to an S3 Object Lambda access point. When an application sends [standard S3 GET requests](#) through the S3 Object Lambda access point, the specified Lambda function is invoked to process any data retrieved from an S3 bucket through the supporting S3 access point. Then, the S3 Object Lambda access point returns the transformed result back to the application. You can author and execute your own custom Lambda functions, tailoring the S3 Object Lambda data transformation to your specific use case, all with no changes required to your applications.



Objective

In this tutorial, you learn how to add custom code to standard S3 GET requests to modify the requested object retrieved from S3 so that the object suit the needs of the requesting client or application. Specifically, you learn how to transform all the text in the original object stored in S3 to uppercase through S3 Object Lambda.

Topics

- [Prerequisites \(p. 28\)](#)
- [Step 1: Create an S3 bucket \(p. 30\)](#)
- [Step 2: Upload a file to the S3 bucket \(p. 30\)](#)
- [Step 3: Create an S3 access point \(p. 31\)](#)
- [Step 4: Create a Lambda function \(p. 31\)](#)
- [Step 5: Configure an IAM policy for your Lambda function's execution role \(p. 36\)](#)
- [Step 6: Create an S3 Object Lambda access point \(p. 36\)](#)
- [Step 7: View the transformed data \(p. 37\)](#)
- [Step 8: Clean up \(p. 39\)](#)
- [Next steps \(p. 41\)](#)

Prerequisites

Before you start this tutorial, you must have an AWS account that you can sign in to as an AWS Identity and Access Management (IAM) user with correct permissions. You also must install Python version 3.8 or later.

Substeps

- [Create an IAM user with permissions in your AWS account \(console\) \(p. 29\)](#)
- [Install Python 3.8 or later on your local machine \(p. 29\)](#)

Create an IAM user with permissions in your AWS account (console)

You can create an IAM user for the tutorial, or you can add permissions to an existing IAM user. To complete this tutorial, your IAM user must attach the following IAM policies to access relevant AWS resources and perform specific actions.

Your IAM user requires the following policies:

- [AmazonS3FullAccess](#) – Grants permissions to all Amazon S3 actions, including permissions to create and use an Object Lambda access point.
- [AWSLambda_FullAccess](#) – Grants permissions to all Lambda actions.
- [IAMFullAccess](#) – Grants permissions to all IAM actions.
- [IAMAccessAnalyzerReadOnlyAccess](#) – Grants permissions to read all access information provided by IAM Access Analyzer.

Note

For simplicity, this tutorial uses full-access AWS managed policies. For production use, we recommend that you instead grant only the minimum permissions necessary for your use case, in accordance with [security best practices \(p. 633\)](#).

For more information about how to create an IAM user, see [Creating IAM users \(console\)](#) in the *IAM User Guide*.

Install Python 3.8 or later on your local machine

Use the following procedure to install Python 3.8 or later on your local machine. For more installation instructions, see the [Downloading Python](#) page in the *Python Beginners Guide*.

1. Open your local terminal or shell and run the following command to determine whether Python is already installed, and if so, which version is installed.

```
python --version
```

2. If you don't have Python 3.8 or later, download the [official installer](#) of Python 3.8 or later that's suitable for your local machine.
3. Run the installer by double-clicking the downloaded file, and follow the steps to complete the installation.

For **Windows users**, choose **Add Python 3.X to PATH** in the installation wizard before choosing **Install Now**.

4. Restart your terminal by closing and reopening it.
5. Run the following command to verify that Python 3.8 or later is installed correctly.

For **macOS users**, run this command:

```
python3 --version
```

For **Windows users**, run this command:

```
python --version
```

6. Run the following command to verify that the pip3 package manager is installed. If you see a pip version number and python 3.8 or later in the command response, that means the pip3 package manager is installed successfully.

```
pip --version
```

Step 1: Create an S3 bucket

Create a bucket to store the original data that you plan to transform.

To create a bucket

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. Choose **Create bucket**.

The **Create bucket** page opens.

4. For **Bucket name**, enter a name (for example, `tutorial-bucket`) for your bucket.

For more information about naming buckets in Amazon S3, see [Bucket naming rules \(p. 118\)](#).

5. For **Region**, choose the AWS Region where you want the bucket to reside.

For more information about the bucket Region, see [Buckets overview \(p. 114\)](#).

6. For **Block Public Access settings for this bucket**, keep the default settings (**Block all public access** is enabled).

We recommend that you keep all Block Public Access settings enabled unless you need to turn off one or more of them for your use case. For more information about blocking public access, see [Blocking public access to your Amazon S3 storage \(p. 584\)](#).

7. For the remaining settings, keep the defaults.

(Optional) If you want to configure additional bucket settings for your specific use case, see [Creating a bucket \(p. 119\)](#).

8. Choose **Create bucket**.

Step 2: Upload a file to the S3 bucket

Upload a text file to the S3 bucket. This text file contains the original data that you will transform to uppercase later in this tutorial.

For example, you can upload a `tutorial.txt` file that contains the following text:

```
Amazon S3 Object Lambda Tutorial:  
You can add your own code to process data retrieved from S3 before  
returning it to an application.
```

To upload a file to a bucket

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. In the **Buckets** list, choose the name of the bucket that you created in [Step 1 \(p. 30\)](#) (for example, `tutorial-bucket`) to upload your file to.

4. On the **Objects** tab for your bucket, choose **Upload**.
5. On the **Upload** page, under **Files and folders**, choose **Add files**.
6. Choose a file to upload, and then choose **Open**. For example, you can upload the `tutorial.txt` file example mentioned earlier.
7. Choose **Upload**.

Step 3: Create an S3 access point

To use an S3 Object Lambda access point to access and transform the original data, you must create an S3 access point and associate it with the S3 bucket that you created in [Step 1 \(p. 30\)](#). The access point must be in the same AWS Region as the objects that you want to transform.

Later in this tutorial, you'll use this access point as a supporting access point for your Object Lambda access point.

To create an access point

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 **Access Points**.
3. On the **Access Points** page, choose **Create access point**.
4. In the **Access point name** field, enter the name (for example, `tutorial-access-point`) for the access point.

For more information about naming access points, see [Rules for naming Amazon S3 access points \(p. 306\)](#).

5. In the **Bucket name** field, enter the name of the bucket that you created in [Step 1 \(p. 30\)](#) (for example, `tutorial-bucket`). S3 attaches the access point to this bucket.

(Optional) You can choose **Browse S3** to browse and search the buckets in your account. If you choose **Browse S3**, choose the desired bucket, and then choose **Choose path** to populate the **Bucket name** field with that bucket's name.

6. For **Network origin**, choose **Internet**.

For more information about network origins for access points, see [Creating access points restricted to a virtual private cloud \(p. 308\)](#).

7. By default, all Block Public Access settings are turned on for your access point. We recommend that you keep **Block all public access** enabled.

For more information, see [Managing public access to access points \(p. 309\)](#).

8. For all other access point settings, keep the default settings.

(Optional) You can modify the access point settings to support your use case. For this tutorial, we recommend keeping the default settings.

(Optional) If you need to manage access to your access point, you can specify an access point policy. For more information, see [Access point policy examples \(p. 303\)](#).

9. Choose **Create access point**.

Step 4: Create a Lambda function

To transform original data, create a Lambda function for use with your S3 Object Lambda access point.

Substeps

- [Write Lambda function code and create a deployment package with a virtual environment \(p. 32\)](#)
- [Create a Lambda function with an execution role \(console\) \(p. 35\)](#)
- [Deploy your Lambda function code with .zip file archives and configure the Lambda function \(console\) \(p. 35\)](#)

Write Lambda function code and create a deployment package with a virtual environment

1. On your local machine, create a folder with the folder name `object-lambda` for the virtual environment to use later in this tutorial.
2. In the `object-lambda` folder, create a file with a Lambda function that changes all text in the original object to uppercase. For example, you can use the following function written in Python. Save this function in a file named `transform.py`.

```
import boto3
import requests

# This function capitalizes all text in the original object
def lambda_handler(event, context):
    object_context = event["getObjectContext"]
    # Get the presigned URL to fetch the requested original object
    # from S3
    s3_url = object_context["inputS3Url"]
    # Extract the route and request token from the input context
    request_route = object_context["outputRoute"]
    request_token = object_context["outputToken"]

    # Get the original S3 object using the presigned URL
    response = requests.get(s3_url)
    original_object = response.content.decode("utf-8")

    # Transform all text in the original object to uppercase
    # You can replace it with your custom code based on your use case
    transformed_object = original_object.upper()

    # Write object back to S3 Object Lambda
    s3 = boto3.client('s3')
    # The WriteGetObjectResponse API sends the transformed data
    # back to S3 Object Lambda and then to the user
    s3.write_get_object_response(
        Body=transformed_object,
        RequestRoute=request_route,
        RequestToken=request_token)

    # Exit the Lambda function: return the status code
    return {'status_code': 200}
```

Note

The preceding example Lambda function loads the entire requested object into memory before transforming it and returning it to the client. Alternatively, you can stream the object from S3 to avoid loading the entire object into memory. This approach can be useful when working with large objects. For more information about streaming responses with Object Lambda access points, see the streaming examples in [Working with WriteGetObjectResponse \(p. 284\)](#).

When you're writing a Lambda function for use with an S3 Object Lambda access point, the function is based on the input event context that S3 Object Lambda provides to the Lambda function. The

event context provides information about the request being made in the event passed from S3 Object Lambda to Lambda. It contains the parameters that you use to create the Lambda function.

The fields used to create the preceding Lambda function are as follows:

The field of `getObjectContext` means the input and output details for connections to Amazon S3 and S3 Object Lambda. It has the following fields:

- `inputS3Url` – A presigned URL that the Lambda function can use to download the original object from the supporting access point. By using a presigned URL, the Lambda function doesn't need to have Amazon S3 read permissions to retrieve the original object and can only access the object processed by each invocation.
- `outputRoute` – A routing token that is added to the S3 Object Lambda URL when the Lambda function calls `WriteGetObjectResponse` to send back the transformed object.
- `outputToken` – A token used by S3 Object Lambda to match the `WriteGetObjectResponse` call with the original caller when sending back the transformed object.

For more information about all the fields in the event context, see [Event context format and usage \(p. 295\)](#) and [Writing and debugging AWS Lambda functions for Amazon S3 Object Lambda Access Points \(p. 284\)](#).

3. In your local terminal, enter the following command to install the `virtualenv` package:

```
python -m pip install virtualenv
```

4. In your local terminal, open the `object-lambda` folder that you created earlier, and then enter the following command to create and initialize a virtual environment called `venv`.

```
python -m virtualenv venv
```

5. To activate the virtual environment, enter the following command to execute the `activate` file from the environment's folder:

For **macOS users**, run this command:

```
source venv/bin/activate
```

For **Windows users**, run this command:

```
.\venv\Scripts\activate
```

Now, your command prompt changes to show **(venv)**, indicating that the virtual environment is active.

6. To install the required libraries, run the following commands line by line in the `venv` virtual environment.

These commands install updated versions of the dependencies of your `lambda_handler` Lambda function. These dependencies are the AWS SDK for Python (`Boto3`) and the `requests` module.

```
pip3 install boto3
```

```
pip3 install requests
```

7. To deactivate the virtual environment, run the following command:

API Version 2006-03-01

```
deactivate
```

8. To create a deployment package with the installed libraries as a .zip file named `lambda.zip` at the root of the `object-lambda` directory, run the following commands line by line in your local terminal.

Tip

The following commands might need to be adjusted to work in your particular environment. For example, a library might appear in `site-packages` or `dist-packages`, and the first folder might be `lib` or `lib64`. Also, the `python` folder might be named with a different Python version. To locate a specific package, use the `pip show` command.

For **macOS users**, run these commands:

```
cd venv/lib/python3.8/site-packages
```

```
zip -r ../../../../lambda.zip .
```

For **Windows users**, run these commands:

```
cd .\venv\Lib\site-packages\
```

```
powershell Compress-Archive * ../../../../../../lambda.zip
```

The last command saves the deployment package to the root of the `object-lambda` directory.

9. Add the function code file `transform.py` to the root of your deployment package.

For **macOS users**, run these commands:

```
cd ../../../../../../
```

```
zip -g lambda.zip transform.py
```

For **Windows users**, run these commands:

```
cd ..\..\..\..
```

```
powershell Compress-Archive -update transform.py lambda.zip
```

After you complete this step, you should have the following directory structure:

```
lambda.zip$  
# transform.py  
# __pycache__  
| boto3/  
# certifi/  
# pip/  
# requests/  
...
```

Create a Lambda function with an execution role (console)

1. Sign in to the AWS Management Console and open the AWS Lambda console at <https://console.aws.amazon.com/lambda/>.
 2. In the left navigation pane, choose **Functions**.
 3. Choose **Create function**.
 4. Choose **Author from scratch**.
 5. Under **Basic information**, do the following:
 - a. For **Function name**, enter `tutorial-object-lambda-function`.
 - b. For **Runtime**, choose **Python 3.8** or a later version.
 6. Expand the **Change default execution role** section. Under **Execution role**, choose **Create a new role with basic Lambda permissions**.
- In Step 5 (p. 36) later in this tutorial, you attach the **AmazonS3ObjectLambdaExecutionRolePolicy** to this Lambda function's execution role.
7. Keep the remaining settings set to the defaults.
 8. Choose **Create function**.

Deploy your Lambda function code with .zip file archives and configure the Lambda function (console)

1. In the AWS Lambda console at <https://console.aws.amazon.com/lambda/>, choose **Functions** in the left navigation pane.
2. Choose the Lambda function that you created earlier (for example, `tutorial-object-lambda-function`).
3. On the Lambda function's details page, choose the **Code** tab. In the **Code Source** section, choose **Upload from** and then **.zip file**.
4. Choose **Upload** to select your local .zip file.
5. Choose the `lambda.zip` file that you created earlier, and then choose **Open**.
6. Choose **Save**.
7. In the **Runtime settings** section, choose **Edit**.
8. On the **Edit runtime settings** page, confirm that **Runtime** is set to **Python 3.8** or a later version.
9. To tell the Lambda runtime which handler method in your Lambda function code to invoke, enter `transform.lambda_handler` for **Handler**.

When you configure a function in Python, the value of the handler setting is the file name and the name of the handler module, separated by a dot. For example, `transform.lambda_handler` calls the `lambda_handler` method defined in the `transform.py` file.

10. Choose **Save**.
11. (Optional) On your Lambda function's details page, choose the **Configuration** tab. In the left navigation pane, choose **General configuration**, then choose **Edit**. In the **Timeout** field, enter **1 min 0 sec**. Keep the remaining settings set to the defaults, and choose **Save**.

Timeout is the amount of time that Lambda allows a function to run for an invocation before stopping it. The default is 3 seconds. The maximum duration for a Lambda function used by S3 Object Lambda is 60 seconds. Pricing is based on the amount of memory configured and the amount of time that your code runs.

Step 5: Configure an IAM policy for your Lambda function's execution role

To enable your Lambda function to provide customized data and response headers to the `GetObject` caller, your Lambda function's execution role must have IAM permissions to call the `WriteGetObjectResponse` API.

To attach an IAM policy to your Lambda function role

1. In the AWS Lambda console at <https://console.aws.amazon.com/lambda/>, choose **Functions** in the left navigation pane.
2. Choose the function that you created in [Step 4 \(p. 31\)](#) (for example, `tutorial-object-lambda-function`).
3. On your Lambda function's details page, choose the **Configuration** tab, and then choose **Permissions** in the left navigation pane.
4. Under **Execution role**, choose the link of the **Role name**. The IAM console opens.
5. On the IAM console's **Summary** page for your Lambda function's execution role, choose the **Permissions** tab, and then choose **Attach policies**.
6. On the **Attach Permissions** page, enter `AmazonS3ObjectLambdaExecutionRolePolicy` in the search box to filter the list of policies. Select the check box next to the name of the `AmazonS3ObjectLambdaExecutionRolePolicy` policy.
7. Choose **Attach policy**.

Step 6: Create an S3 Object Lambda access point

An S3 Object Lambda access point provides the flexibility to invoke a Lambda function directly from an S3 GET request so that the function can process data retrieved from an S3 access point. When creating and configuring an S3 Object Lambda access point, you must specify the Lambda function to invoke and provide the event context in JSON format as custom parameters for Lambda to use.

To create an S3 Object Lambda access point

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 **Object Lambda Access Points**.
3. On the **Object Lambda Access Points** page, choose **Create Object Lambda Access Point**.
4. For **Object Lambda Access Point name**, enter the name that you want to use for the Object Lambda access point (for example, `tutorial-object-lambda-accesspoint`).
5. For **Supporting Access Point**, enter or browse to the standard access point that you created in [Step 3 \(p. 31\)](#) (for example, `tutorial-access-point`), and then choose **Choose supporting Access Point**.
6. For **Invoke Lambda function**, you can choose either of the following two options for this tutorial.
 - Choose **Choose from functions in your account**, and then choose the Lambda function that you created in [Step 4 \(p. 31\)](#) (for example, `tutorial-object-lambda-function`) from the **Lambda function** dropdown list.
 - Choose **Enter ARN**, and then enter the Amazon Resource Name (ARN) of the Lambda function that you created in [Step 4 \(p. 31\)](#).
7. For **Lambda function version**, choose **\$LATEST** (the latest version of the Lambda function that you created in [Step 4 \(p. 31\)](#)).

8. (Optional) If you need your Lambda function to recognize and process GET requests with range and part number headers, select **Lambda function supports requests using range** and **Lambda function supports requests using part numbers**. Otherwise, clear these two check boxes.

For more information about how to use range or part numbers with S3 Object Lambda, see [Working with Range and partNumber headers \(p. 293\)](#).

9. (Optional) Under **Payload - optional**, add JSON text to provide your Lambda function with additional information.

A payload is optional JSON text that you can provide to your Lambda function as input for all invocations coming from a specific S3 Object Lambda access point. To customize the behaviors for multiple Object Lambda access points that invoke the same Lambda function, you can configure payloads with different parameters, thereby extending the flexibility of your Lambda function.

For more information about payload, see [Event context format and usage \(p. 295\)](#).

10. (Optional) For **Request metrics - optional**, choose **Disable** or **Enable** to add Amazon S3 monitoring to your Object Lambda access point. Request metrics are billed at the standard Amazon CloudWatch rate. For more information, see [CloudWatch pricing](#).

11. Under **Object Lambda Access Point policy - optional**, keep the default setting.

(Optional) You can set a resource policy. This resource policy grants the `GetObject` API permission to use the specified Object Lambda access point.

12. Keep the remaining settings set to the defaults, and choose **Create Object Lambda Access Point**.

Step 7: View the transformed data

Now, S3 Object Lambda is ready to transform your data for your use case. In this tutorial, S3 Object Lambda transforms all the text in your object to uppercase.

Substeps

- [View the transformed data in your S3 Object Lambda access point \(p. 37\)](#)
- [Run a Python script to print the original and transformed data \(p. 38\)](#)

View the transformed data in your S3 Object Lambda access point

When you request to retrieve a file through your S3 Object Lambda access point, you make a `GetObject` API call to S3 Object Lambda. S3 Object Lambda invokes the Lambda function to transform your data, and then returns the transformed data as the response to the standard S3 `GetObject` API call.

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 **Object Lambda Access Points**.
3. On the **Object Lambda Access Points** page, choose the S3 Object Lambda access point that you created in [Step 6 \(p. 36\)](#) (for example, `tutorial-object-lambda-accesspoint`).
4. On the **Objects** tab of your S3 Object Lambda access point, select the file that has the same name (for example, `tutorial.txt`) as the one that you uploaded to the S3 bucket in [Step 2 \(p. 30\)](#).

This file should contain all the transformed data.

5. To view the transformed data, choose **Open** or **Download**.

Run a Python script to print the original and transformed data

You can use S3 Object Lambda with your existing applications. To do so, update your application configuration to use the new S3 Object Lambda access point ARN that you created in [Step 6 \(p. 36\)](#) to retrieve data from S3.

The following example Python script prints both the original data from the S3 bucket and the transformed data from the S3 Object Lambda access point.

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 **Object Lambda Access Points**.
3. On the **Object Lambda Access Points** page, choose the radio button to the left of the S3 Object Lambda access point that you created in [Step 6 \(p. 36\)](#) (for example, `tutorial-object-lambda-accesspoint`).
4. Choose **Copy ARN**.
5. Save the ARN for use later.
6. Write a Python script on your local machine to print both the original data (for example, `tutorial.txt`) from your S3 Bucket and the transformed data (for example, `tutorial.txt`) from your S3 Object Lambda access point. You can use the following example script.

```
import boto3

s3 = boto3.client('s3')

def getObject(bucket, key):
    objectBody = s3.get_object(Bucket = bucket, Key = key)
    print(objectBody["Body"].read().decode("utf-8"))
    print("\n")

print('Original object from the S3 bucket:')
# Replace the two input parameters of getObject() below with
# the S3 bucket name that you created in Step 1 \(p. 30\) and
# the name of the file that you uploaded to the S3 bucket in Step 2 \(p. 30\)
getObject("tutorial-bucket",
          "tutorial.txt")

print('Object transformed by S3 Object Lambda:')
# Replace the two input parameters of getObject() below with
# the ARN of your S3 Object Lambda access point that you saved earlier and
# the name of the file with the transformed data (which in this case is
# the same as the name of the file that you uploaded to the S3 bucket
# in Step 2 \(p. 30\))
getObject("arn:aws:s3-object-lambda:us-west-2:111122223333:accesspoint/tutorial-object-lambda-accesspoint",
          "tutorial.txt")
```

7. Save your Python script with a custom name (for example, `tutorial_print.py`) in the folder (for example, `object-lambda`) that you created in [Step 4 \(p. 31\)](#) on your local machine.
8. In your local terminal, run the following command from the root of the directory (for example, `object-lambda`) that you created in [Step 4 \(p. 31\)](#).

```
python3 tutorial_print.py
```

You should see both the original data and the transformed data (all text as uppercase) through the terminal. For example, you should see something like the following text.

```
Original object from the S3 bucket:
```

Amazon S3 Object Lambda Tutorial:
You can add your own code to process data retrieved from S3 before returning it to an application.

Object transformed by S3 Object Lambda:
AMAZON S3 OBJECT LAMBDA TUTORIAL:
YOU CAN ADD YOUR OWN CODE TO PROCESS DATA RETRIEVED FROM S3 BEFORE RETURNING IT TO AN APPLICATION.

Step 8: Clean up

If you transformed your data through S3 Object Lambda only as a learning exercise, delete the AWS resources that you allocated so that you no longer accrue charges.

Substeps

- [Delete the Object Lambda access point \(p. 39\)](#)
- [Delete the S3 access point \(p. 39\)](#)
- [Delete the execution role for your Lambda function \(p. 40\)](#)
- [Delete the Lambda function \(p. 40\)](#)
- [Delete the CloudWatch log group \(p. 40\)](#)
- [Delete the original file in the S3 source bucket \(p. 40\)](#)
- [Delete the S3 source bucket \(p. 41\)](#)
- [Delete the IAM user \(p. 41\)](#)

Delete the Object Lambda access point

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 **Object Lambda Access Points**.
3. On the **Object Lambda Access Points** page, choose the radio button to the left of the S3 Object Lambda access point that you created in [Step 6 \(p. 36\)](#) (for example, `tutorial-object-lambda-accesspoint`).
4. Choose **Delete**.
5. Confirm that you want to delete your Object Lambda access point by entering its name in the text field that appears, and then choose **Delete**.

Delete the S3 access point

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 **Access Points**.
3. Navigate to the access point that you created in [Step 3 \(p. 31\)](#) (for example, `tutorial-access-point`), and choose the radio button next to the name of the access point.
4. Choose **Delete**.
5. Confirm that you want to delete your access point by entering its name in the text field that appears, and then choose **Delete**.

Delete the execution role for your Lambda function

1. Sign in to the AWS Management Console and open the AWS Lambda console at <https://console.aws.amazon.com/lambda/>.
2. In the left navigation pane, choose **Functions**.
3. Choose the function that you created in [Step 4 \(p. 31\)](#) (for example, `tutorial-object-lambda-function`).
4. On your Lambda function's details page, choose the **Configuration** tab, and then choose **Permissions** in the left navigation pane.
5. Under **Execution role**, choose the link of the **Role name**. The IAM console opens.
6. On the IAM console's **Summary** page of your Lambda function's execution role, choose **Delete role**.
7. In the **Delete role** dialog box, choose **Yes, delete**.

Delete the Lambda function

1. In the AWS Lambda console at <https://console.aws.amazon.com/lambda/>, choose **Functions** in the left navigation pane.
2. Select the check box to the left of the name of the function that you created in [Step 4 \(p. 31\)](#) (for example, `tutorial-object-lambda-function`).
3. Choose **Actions**, and then choose **Delete**.
4. In the **Delete function** dialog box, choose **Delete**.

Delete the CloudWatch log group

1. Open the CloudWatch console at <https://console.aws.amazon.com/cloudwatch/>.
2. In the left navigation pane, choose **Log groups**.
3. Find the log group whose name ends with the Lambda function that you created in [Step 4 \(p. 31\)](#) (for example, `tutorial-object-lambda-function`).
4. Select the check box to the left of the name of the log group.
5. Choose **Actions**, and then choose **Delete log group(s)**.
6. In the **Delete log group(s)** dialog box, choose **Delete**.

Delete the original file in the S3 source bucket

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. In the **Bucket name** list, choose the name of the bucket that you uploaded the original file to in [Step 2 \(p. 30\)](#) (for example, `tutorial-bucket`).
4. Select the check box to the left of the name of the object that you want to delete (for example, `tutorial.txt`).
5. Choose **Delete**.
6. On the **Delete objects** page, in the **Permanently delete objects?** section, confirm that you want to delete this object by entering `permanently delete` in the text box.
7. Choose **Delete objects**.

Delete the S3 source bucket

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. In the **Buckets** list, choose the radio button next to the name of the bucket that you created in [Step 1 \(p. 30\)](#) (for example, **tutorial-bucket**).
4. Choose **Delete**.
5. On the **Delete bucket** page, confirm that you want to delete the bucket by entering the bucket name in the text field, and then choose **Delete bucket**.

Delete the IAM user

1. Sign in to the AWS Management Console and open the IAM console at <https://console.aws.amazon.com/iam/>.
2. In the left navigation pane, choose **Users**, and then select the check box next to the user name that you want to delete.
3. At the top of the page, choose **Delete**.
4. In the **Delete user name?** dialog box, enter the user name in the text input field to confirm the deletion of the user. Choose **Delete**.

Next steps

After completing this tutorial, you can customize the Lambda function for your use case to modify the data returned by standard S3 GET requests.

The following is a list of common use cases for S3 Object Lambda:

- Masking sensitive data for security and compliance.

For more information, see [Tutorial: Detecting and redacting PII data with S3 Object Lambda and Amazon Comprehend \(p. 41\)](#).

- Filtering certain rows of data to deliver specific information.
- Augmenting data with information from other services or databases.
- Converting across data formats, such as converting XML to JSON for application compatibility.
- Compressing or decompressing files as they are being downloaded.
- Resizing and watermarking images.
- Implementing custom authorization rules to access data.

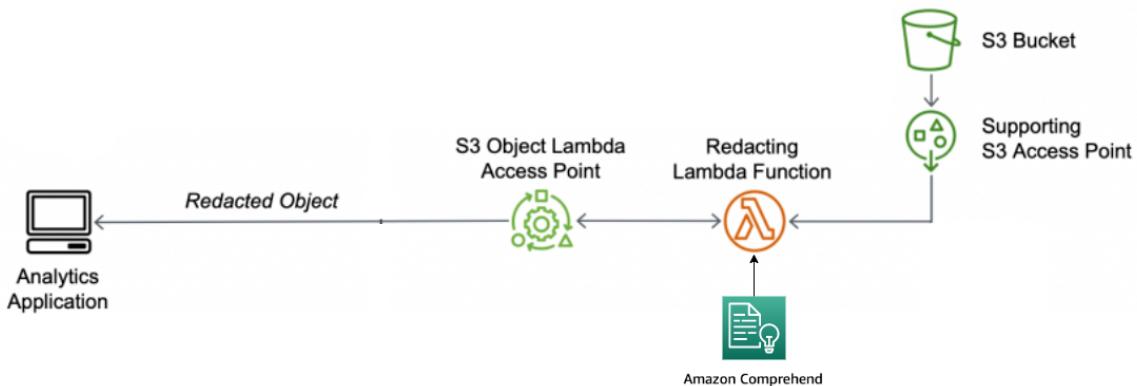
For more information about S3 Object Lambda, see [Transforming objects with S3 Object Lambda \(p. 271\)](#).

Tutorial: Detecting and redacting PII data with S3 Object Lambda and Amazon Comprehend

When you're using Amazon S3 for shared datasets for multiple applications and users to access, it's important to restrict privileged information, such as personally identifiable information (PII), to only

authorized entities. For example, when a marketing application uses some data containing PII, it might need to first mask PII data to meet data privacy requirements. Also, when an analytics application uses a production order inventory dataset, it might need to first redact customer credit card information to prevent unintended data leakage.

With [S3 Object Lambda](#) and a prebuilt AWS Lambda function powered by Amazon Comprehend, you can protect PII data retrieved from S3 before returning it to an application. Specifically, you can use the prebuilt [Lambda function](#) as a redacting function and attach it to an S3 Object Lambda access point. When an application (for example, an analytics application) sends [standard S3 GET requests](#), these requests made through the S3 Object Lambda access point invoke the prebuilt redacting Lambda function to detect and redact PII data retrieved from an S3 bucket through a supporting S3 access point. Then, the S3 Object Lambda access point returns the redacted result back to the application.



In the process, the prebuilt Lambda function uses [Amazon Comprehend](#), a natural language processing (NLP) service, to capture variations in how PII is represented, regardless of how PII exists in text (such as numerically or as a combination of words and numbers). Amazon Comprehend can even use context in the text to understand if a 4-digit number is a PIN, the last four numbers of a Social Security number (SSN), or a year. Amazon Comprehend processes any text file in UTF-8 format and can protect PII at scale without affecting accuracy. For more information, see [What is Amazon Comprehend?](#) in the [Amazon Comprehend Developer Guide](#).

Objective

In this tutorial, you learn how to use S3 Object Lambda with the prebuilt Lambda function `ComprehendPiiRedactionS3ObjectLambda`. This function uses Amazon Comprehend to detect PII entities. It then redacts these entities by replacing them with asterisks. By redacting PII, you conceal sensitive data, which can help with security and compliance.

You also learn how to use and configure a prebuilt AWS Lambda function in the [AWS Serverless Application Repository](#) to work together with S3 Object Lambda for easy deployment.

Topics

- [Prerequisites: Create an IAM user with permissions \(p. 43\)](#)
- [Step 1: Create an S3 bucket \(p. 44\)](#)
- [Step 2: Upload a file to the S3 bucket \(p. 44\)](#)
- [Step 3: Create an S3 access point \(p. 45\)](#)
- [Step 4: Configure and deploy a prebuilt Lambda function \(p. 46\)](#)
- [Step 5: Create an S3 Object Lambda access point \(p. 46\)](#)
- [Step 6: Use the S3 Object Lambda access point to retrieve the redacted file \(p. 48\)](#)
- [Step 7: Clean up \(p. 48\)](#)
- [Next steps \(p. 51\)](#)

Prerequisites: Create an IAM user with permissions

Before you start this tutorial, you must have an AWS account that you can sign in to as an AWS Identity and Access Management (IAM) user with correct permissions.

You can create an IAM user for the tutorial, or you can add permissions to an existing IAM user. To complete this tutorial, your IAM user must attach the following IAM policies to access relevant AWS resources and perform specific actions.

Note

For simplicity, this tutorial uses full-access policies. For production use, we recommend that you instead grant only the minimum permissions necessary for your use case, in accordance with [security best practices \(p. 633\)](#).

Your IAM user requires the following AWS managed policies:

- [AmazonS3FullAccess](#) – Grants permissions to all Amazon S3 actions, including permissions to create and use an Object Lambda access point.
- [AWSLambda_FullAccess](#) – Grants permissions to all Lambda actions.
- [AWSCloudFormationFullAccess](#) – Grants permissions to all AWS CloudFormation actions.
- [IAMFullAccess](#) – Grants permissions to all IAM actions.
- [IAMAccessAnalyzerReadOnlyAccess](#) – Grants permissions to read all access information provided by IAM Access Analyzer.

You can directly attach these existing policies when creating an IAM user. For more information about how to create an IAM user, see [Creating IAM users \(console\)](#) in the *IAM User Guide*.

In addition, your IAM user requires a customer managed policy. To grant the IAM user permissions to all AWS Serverless Application Repository resources and actions, you must create an IAM policy and attach the policy to the IAM user.

To create and attach an IAM policy to your IAM user

1. Sign in to the AWS Management Console and open the IAM console at <https://console.aws.amazon.com/iam/>.
2. In the left navigation pane, choose **Policies**.
3. Choose **Create policy**.
4. On the **Visual editor** tab, for **Service**, choose **Choose a service**. Then, choose **Serverless Application Repository**.
5. For **Actions**, under **Manual actions**, select **All Serverless Application Repository actions (serverlessrepo:*)** for this tutorial.

As a security best practice, you should allow permissions to only those actions and resources that a user needs, based on your use case. For more information, see [Security best practices in IAM](#) in the *IAM User Guide*.

6. For **Resources**, choose **All resources** for this tutorial.

As a best practice, you should define permissions for only specific resources in specific accounts. Alternatively, you can grant least privilege using condition keys. For more information, see [Grant least privilege](#) in the *IAM User Guide*.

7. Choose **Next: Tags**.
8. Choose **Next: Review**.
9. On the **Review policy** page, enter a **Name** (for example, **tutorial-serverless-application-repository**) and a **Description** (optional) for the policy that you are creating. Review the policy

summary to make sure that you have granted the intended permissions, and then choose **Create policy** to save your new policy.

10. In the left navigation pane, choose **Users**. Then, choose the IAM user for this tutorial.
11. On the **Summary** page of the chosen user, choose the **Permissions** tab, and then choose **Add permissions**.
12. Under **Grant permissions**, choose **Attach existing policies directly**.
13. Select the check box next to the policy that you just created (for example, `tutorial-serverless-application-repository`) and then choose **Next: Review**.
14. Under **Permissions summary**, review the summary to make sure that you attached the intended policy. Then, choose **Add permissions**.

Step 1: Create an S3 bucket

Create a bucket to store the original data that you plan to transform.

To create a bucket

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. Choose **Create bucket**.
The **Create bucket** page opens.
4. For **Bucket name**, enter a name (for example, `tutorial-bucket`) for your bucket.
For more information about naming buckets in Amazon S3, see [Bucket naming rules \(p. 118\)](#).
5. For **Region**, choose the AWS Region where you want the bucket to reside.
For more information about the bucket Region, see [Buckets overview \(p. 114\)](#).
6. For **Block Public Access settings for this bucket**, keep the default settings (**Block all public access** is enabled).
We recommend that you keep all Block Public Access settings enabled unless you need to turn off one or more of them for your use case. For more information about blocking public access, see [Blocking public access to your Amazon S3 storage \(p. 584\)](#).
7. For the remaining settings, keep the defaults.
(Optional) If you want to configure additional bucket settings for your specific use case, see [Creating a bucket \(p. 119\)](#).
8. Choose **Create bucket**.

Step 2: Upload a file to the S3 bucket

Upload a text file containing known PII data of various types, such as names, banking information, phone numbers, and SSNs, to the S3 bucket as the original data that you will redact PII from later in this tutorial.

For example, you can upload following the `tutorial.txt` file. This is an example input file from Amazon Comprehend.

```
Hello Zhang Wei, I am John. Your AnyCompany Financial Services,  
LLC credit card account 1111-0000-1111-0008 has a minimum payment
```

of \$24.53 that is due by July 31st. Based on your autopay settings, we will withdraw your payment on the due date from your bank account number XXXXXX1111 with the routing number XXXXX0000.

Your latest statement was mailed to 100 Main Street, Any City, WA 98121.

After your payment is received, you will receive a confirmation text message at 206-555-0100.

If you have questions about your bill, AnyCompany Customer Service is available by phone at 206-555-0199 or email at support@anycompany.com.

To upload a file to a bucket

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. In the **Buckets** list, choose the name of the bucket that you created in [Step 1 \(p. 44\)](#) (for example, **tutorial-bucket**) to upload your file to.
4. On the **Objects** tab for your bucket, choose **Upload**.
5. On the **Upload** page, under **Files and folders**, choose **Add files**.
6. Choose a file to upload, and then choose **Open**. For example, you can upload the `tutorial.txt` file example mentioned earlier.
7. Choose **Upload**.

Step 3: Create an S3 access point

To use an S3 Object Lambda access point to access and transform the original data, you must create an S3 access point and associate it with the S3 bucket that you created in [Step 1 \(p. 44\)](#). The access point must be in the same AWS Region as the objects you want to transform.

Later in this tutorial, you'll use this access point as a supporting access point for your Object Lambda access point.

To create an access point

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 **Access Points**.
3. On the **Access Points** page, choose **Create access point**.
4. In the **Access point name** field, enter the name (for example, **tutorial-pii-access-point**) for the access point.

For more information about naming access points, see [Rules for naming Amazon S3 access points \(p. 306\)](#).

5. In the **Bucket name** field, enter the name of the bucket that you created in [Step 1 \(p. 44\)](#) (for example, **tutorial-bucket**). S3 attaches the access point to this bucket.

(Optional) You can choose **Browse S3** to browse and search the buckets in your account. If you choose **Browse S3**, choose the desired bucket, and then choose **Choose path** to populate the **Bucket name** field with that bucket's name.

6. For **Network origin**, choose **Internet**.

For more information about network origins for access points, see [Creating access points restricted to a virtual private cloud \(p. 308\)](#).

7. By default, all block public access settings are turned on for your access point. We recommend that you keep **Block all public access** enabled. For more information, see [Managing public access to access points \(p. 309\)](#).
8. For all other access point settings, keep the default settings.

(Optional) You can modify the access point settings to support your use case. For this tutorial, we recommend keeping the default settings.

(Optional) If you need to manage access to your access point, you can specify an access point policy. For more information, see [Access point policy examples \(p. 303\)](#).
9. Choose **Create access point**.

Step 4: Configure and deploy a prebuilt Lambda function

To redact PII data, configure and deploy the prebuilt AWS Lambda function `ComprehendPiiRedactionS3ObjectLambda` for use with your S3 Object Lambda access point.

To configure and deploy the Lambda function

1. Sign in to the AWS Management Console and view the `ComprehendPiiRedactionS3ObjectLambda` function in the AWS Serverless Application Repository.
2. For **Application settings**, under **Application name**, keep the default value (`ComprehendPiiRedactionS3ObjectLambda`) for this tutorial.

(Optional) You can enter the name that you want to give to this application. You might want to do this if you plan to configure multiple Lambda functions for different access needs for the same shared dataset.
3. For **MaskCharacter**, keep the default value (*). The mask character replaces each character in the redacted PII entity.
4. For **MaskMode**, keep the default value (**MASK**). The **MaskMode** value specifies whether the PII entity is redacted with the MASK character or the `PII_ENTITY_TYPE` value.
5. To redact the specified types of data, for **PiiEntityTypes**, keep the default value **ALL**. The **PiiEntityTypes** value specifies the PII entity types to be considered for redaction.

For more information about the list of supported PII entity types, see [Detect Personally Identifiable Information \(PII\) in the Amazon Comprehend Developer Guide](#).
6. Keep the remaining settings set to the defaults.

(Optional) If you want to configure additional settings for your specific use case, see the **Readme file** section on the left side of the page.
7. Select the check box next to **I acknowledge that this app creates custom IAM roles**.
8. Choose **Deploy**.
9. On the new application's page, under **Resources**, choose the **Logical ID** of the Lambda function that you deployed to review the function on the Lambda function page.

Step 5: Create an S3 Object Lambda access point

An S3 Object Lambda access point provides the flexibility to invoke a Lambda function directly from an S3 GET request so that the function can redact PII data retrieved from an S3 access point. When creating

and configuring an S3 Object Lambda access point, you must specify the redacting Lambda function to invoke and provide the event context in JSON format as custom parameters for Lambda to use.

The event context provides information about the request being made in the event passed from S3 Object Lambda to Lambda. For more information about all the fields in the event context, see [Event context format and usage \(p. 295\)](#).

To create an S3 Object Lambda access point

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 **Object Lambda Access Points**.
3. On the **Object Lambda Access Points** page, choose **Create Object Lambda Access Point**.
4. For **Object Lambda Access Point name**, enter the name that you want to use for the Object Lambda access point (for example, `tutorial-pii-object-lambda-accesspoint`).
5. For **Supporting Access Point**, enter or browse to the standard access point that you created in [Step 3 \(p. 45\)](#) (for example, `tutorial-pii-access-point`), and then choose **Choose supporting Access Point**.
6. For **Invoke Lambda function**, you can choose either of the following two options for this tutorial.
 - Choose **Choose from functions in your account** and choose the Lambda function that you deployed in [Step 4 \(p. 46\)](#) (for example, `serverlessrepo-ComprehendPiiRedactionS3ObjectLambda`) from the **Lambda function** dropdown list.
 - Choose **Enter ARN**, and then enter the Amazon Resource Name (ARN) of the Lambda function that you created in [Step 4 \(p. 46\)](#).
7. For **Lambda function version**, choose **\$LATEST** (the latest version of the Lambda function that you deployed in [Step 4 \(p. 46\)](#)).
8. (Optional) If you need your Lambda function to recognize and process GET requests with range and part number headers, select **Lambda function supports requests using range** and **Lambda function supports requests using part numbers**. Otherwise, clear these two check boxes.

For more information about how to use range or part numbers with S3 Object Lambda, see [Working with Range and partNumber headers \(p. 293\)](#).

9. (Optional) Under **Payload - optional**, add JSON text to provide your Lambda function with additional information.

A payload is optional JSON text that you can provide to your Lambda function as input for all invocations coming from a specific S3 Object Lambda access point. To customize the behaviors for multiple Object Lambda access points that invoke the same Lambda function, you can configure payloads with different parameters, thereby extending the flexibility of your Lambda function.

For more information about payload, see [Event context format and usage \(p. 295\)](#).

10. (Optional) For **Request metrics - optional**, choose **Disable** or **Enable** to add Amazon S3 monitoring to your Object Lambda access point. Request metrics are billed at the standard Amazon CloudWatch rate. For more information, see [CloudWatch pricing](#).
11. Under **Object Lambda Access Point policy - optional**, keep the default setting.

(Optional) You can set a resource policy. This resource policy grants the `GetObject` API permission to use the specified Object Lambda access point.
12. Keep the remaining settings set to the defaults, and choose **Create Object Lambda Access Point**.

Step 6: Use the S3 Object Lambda access point to retrieve the redacted file

Now, S3 Object Lambda is ready to redact PII data from your original file.

To use the S3 Object Lambda access point to retrieve the redacted file

When you request to retrieve a file through your S3 Object Lambda access point, you make a `GetObject` API call to S3 Object Lambda. S3 Object Lambda invokes the Lambda function to redact your PII data and returns the transformed data as the response to the standard S3 `GetObject` API call.

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 **Object Lambda Access Points**.
3. On the **Object Lambda Access Points** page, choose the S3 Object Lambda access point that you created in [Step 5 \(p. 46\)](#) (for example, `tutorial-pii-object-lambda-accesspoint`).
4. On the **Objects** tab of your S3 Object Lambda access point, select the file that has the same name (for example, `tutorial.txt`) as the one that you uploaded to the S3 bucket in [Step 2 \(p. 44\)](#).

This file should contain all the transformed data.

5. To view the transformed data, choose **Open or Download**.

You should be able to see the redacted file, as shown in the following example.

```
Hello *****. Your AnyCompany Financial Services,  
LLC credit card account ***** has a minimum payment  
of $24.53 that is due by *****. Based on your autopay settings,  
we will withdraw your payment on the due date from your  
bank account ***** with the routing number *****.  
  
Your latest statement was mailed to *****.  
After your payment is received, you will receive a confirmation  
text message at *****.  
If you have questions about your bill, AnyCompany Customer Service  
is available by phone at ***** or  
email at *****.
```

Step 7: Clean up

If you redacted your data through S3 Object Lambda only as a learning exercise, delete the AWS resources that you allocated so that you no longer accrue charges.

Substeps

- [Delete the Object Lambda access point \(p. 49\)](#)
- [Delete the S3 access point \(p. 49\)](#)
- [Delete the Lambda function \(p. 49\)](#)
- [Delete the CloudWatch log group \(p. 49\)](#)
- [Delete the original file in the S3 source bucket \(p. 49\)](#)
- [Delete the S3 source bucket \(p. 50\)](#)
- [Delete the IAM role for your Lambda function \(p. 50\)](#)
- [Delete the customer managed policy for your IAM user \(p. 50\)](#)
- [Delete the IAM user \(p. 50\)](#)

Delete the Object Lambda access point

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 **Object Lambda Access Points**.
3. On the **Object Lambda Access Points** page, choose the option button to the left of the S3 Object Lambda access point that you created in [Step 5 \(p. 46\)](#) (for example, `tutorial-pii-object-lambda-accesspoint`).
4. Choose **Delete**.
5. Confirm that you want to delete your Object Lambda access point by entering its name in the text field that appears, and then choose **Delete**.

Delete the S3 access point

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 **Access Points**.
3. Navigate to the access point that you created in [Step 3 \(p. 45\)](#) (for example, `tutorial-pii-access-point`), and choose the option button next to the name of the access point.
4. Choose **Delete**.
5. Confirm that you want to delete your access point by entering its name in the text field that appears, and then choose **Delete**.

Delete the Lambda function

1. In the AWS Lambda console at <https://console.aws.amazon.com/lambda/>, choose **Functions** in the left navigation pane.
2. Choose the function that you created in [Step 4 \(p. 46\)](#) (for example, `serverlessrepo-ComprehendPiiRedactionS3ObjectLambda`).
3. Choose **Actions**, and then choose **Delete**.
4. In the **Delete function** dialog box, choose **Delete**.

Delete the CloudWatch log group

1. Open the CloudWatch console at <https://console.aws.amazon.com/cloudwatch/>.
2. In the left navigation pane, choose **Log groups**.
3. Find the log group whose name ends with the Lambda function that you created in [Step 4 \(p. 46\)](#) (for example, `serverlessrepo-ComprehendPiiRedactionS3ObjectLambda`).
4. Choose **Actions**, and then choose **Delete log group(s)**.
5. In the **Delete log group(s)** dialog box, choose **Delete**.

Delete the original file in the S3 source bucket

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. In the **Bucket name** list, choose the name of the bucket that you uploaded the original file to in [Step 2 \(p. 44\)](#) (for example, `tutorial-bucket`).

4. Select the check box to the left of the name of the object that you want to delete (for example, `tutorial.txt`).
5. Choose **Delete**.
6. On the **Delete objects** page, in the **Permanently delete objects?** section, confirm that you want to delete this object by entering `permanently delete` in the text box.
7. Choose **Delete objects**.

Delete the S3 source bucket

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. In the **Buckets** list, choose the option button next to the name of the bucket that you created in [Step 1 \(p. 44\)](#) (for example, `tutorial-bucket`).
4. Choose **Delete**.
5. On the **Delete bucket** page, confirm that you want to delete the bucket by entering the bucket name in the text field, and then choose **Delete bucket**.

Delete the IAM role for your Lambda function

1. Sign in to the AWS Management Console and open the IAM console at <https://console.aws.amazon.com/iam/>.
2. In the left navigation pane, choose **Roles**, and then select the check box next to the role name that you want to delete. The role name starts with the name of the Lambda function that you deployed in [Step 4 \(p. 46\)](#) (for example, `serverlessrepo-ComprehendPiiRedactionS3ObjectLambda`).
3. Choose **Delete**.
4. In the **Delete** dialog box, enter the role name in the text input field to confirm deletion. Then, choose **Delete**.

Delete the customer managed policy for your IAM user

1. Sign in to the AWS Management Console and open the IAM console at <https://console.aws.amazon.com/iam/>.
2. In the left navigation pane, choose **Policies**.
3. On the **Policies** page, enter the name of the customer managed policy that you created in the [Prerequisites \(p. 43\)](#) (for example, `tutorial-serverless-application-repository`) in the search box to filter the list of policies. Select the option button next to the name of the policy that you want to delete.
4. Choose **Actions**, and then choose **Delete**.
5. Confirm that you want to delete this policy by entering its name in the text field that appears, and then choose **Delete**.

Delete the IAM user

1. Sign in to the AWS Management Console and open the IAM console at <https://console.aws.amazon.com/iam/>.
2. In the left navigation pane, choose **Users**, and then select the check box next to the user name that you want to delete.

3. At the top of the page, choose **Delete**.
4. In the **Delete user name?** dialog box, enter the user name in the text input field to confirm the deletion of the user. Choose **Delete**.

Next steps

After completing this tutorial, you can further explore the following related use cases:

- You can create multiple S3 Object Lambda access points and enable them with prebuilt Lambda functions that are configured differently to redact specific types of PII depending on the data accessors' business needs.

Each type of user assumes an IAM role and only has access to one S3 Object Lambda access point (managed through IAM policies). Then, you attach each `ComprehendPiiRedactionS3ObjectLambda` Lambda function configured for a different redaction use case to a different S3 Object Lambda access point. For each S3 Object Lambda access point, you can have a supporting S3 access point to read data from an S3 bucket that stores the shared dataset.

For more information about how to create an S3 bucket policy that allows users to read from the bucket only through S3 access points, see [Configuring IAM policies for using access points \(p. 301\)](#).

For more information about how to grant a user permission to access the Lambda function, the S3 access point, and the S3 Object Lambda access point, see [Configuring IAM policies for Object Lambda access points \(p. 281\)](#).

- You can build your own Lambda function and use S3 Object Lambda with your customized Lambda function to meet your specific data needs.

For example, to explore various data values, you can use S3 Object Lambda and your own Lambda function that uses additional [Amazon Comprehend features](#), such as entity recognition, key phrase recognition, sentiment analysis, and document classification, to process data. You can also use S3 Object Lambda together with [Amazon Comprehend Medical](#), a HIPAA-eligible NLP service, to analyze and extract data in a context-aware manner.

For more information about how to transform data with S3 Object Lambda and your own Lambda function, see [Tutorial: Transforming data for your application with S3 Object Lambda \(p. 27\)](#).

Tutorial: Hosting on-demand streaming video with Amazon S3, Amazon CloudFront, and Amazon Route 53

You can use Amazon S3 with Amazon CloudFront to host videos for on-demand viewing in a secure and scalable way. Video on demand (VOD) streaming means that your video content is stored on a server and viewers can watch it at any time.

CloudFront is a fast, highly secure, and programmable content delivery network (CDN) service. CloudFront can deliver your content securely over HTTPS from all of the CloudFront edge locations around the globe. For more information about CloudFront, see [What is Amazon CloudFront?](#) in the [Amazon CloudFront Developer Guide](#).

CloudFront caching reduces the number of requests that your origin server must respond to directly. When a viewer (end user) requests a video that you serve with CloudFront, the request is routed to a nearby edge location closer to where the viewer is located. CloudFront serves the video from its cache,

retrieving it from the S3 bucket only if it is not already cached. This caching management feature accelerates the delivery of your video to viewers globally with low latency, high throughput, and high transfer speeds. For more information about CloudFront caching management, see [Optimizing caching and availability](#) in the *Amazon CloudFront Developer Guide*.



Objective

In this tutorial, you configure an S3 bucket to host on-demand video streaming using CloudFront for delivery and Amazon Route 53 for Domain Name System (DNS) and custom domain management.

Topics

- [Prerequisites: Register and configure a custom domain with Route 53 \(p. 52\)](#)
- [Step 1: Create an S3 bucket \(p. 53\)](#)
- [Step 2: Upload a video to the S3 bucket \(p. 54\)](#)
- [Step 3: Create a CloudFront origin access identity \(p. 54\)](#)
- [Step 4: Create a CloudFront distribution \(p. 55\)](#)
- [Step 5: Access the video through the CloudFront distribution \(p. 56\)](#)
- [Step 6: Configure your CloudFront distribution to use your custom domain name \(p. 57\)](#)
- [Step 7: Access the S3 video through the CloudFront distribution with the custom domain name \(p. 60\)](#)
- [\(Optional\) Step 8: View data about requests received by your CloudFront distribution \(p. 61\)](#)
- [Step 9: Clean up \(p. 61\)](#)
- [Next steps \(p. 64\)](#)

Prerequisites: Register and configure a custom domain with Route 53

Before you start this tutorial, you must register and configure a custom domain (for example, **example.com**) with Route 53 so that you can configure your CloudFront distribution to use a custom domain name later.

Without a custom domain name, your S3 video is publicly accessible and hosted through CloudFront at a URL that looks similar to the following:

`https://CloudFront distribution domain name/Path to an S3 video`

For example, `https://d111111abcdef8.cloudfront.net/sample.mp4`.

After you configure your CloudFront distribution to use a custom domain name configured with Route 53, your S3 video is publicly accessible and hosted through CloudFront at a URL that looks similar to the following:

`https://CloudFront distribution alternate domain name/Path to an S3 video`

For example, `https://www.example.com/sample.mp4`. A custom domain name is simpler and more intuitive for your viewers to use.

To register a custom domain, see [Registering a new domain using Route 53](#) in the *Amazon Route 53 Developer Guide*.

When you register a domain name with Route 53, Route 53 creates the hosted zone for you, which you will use later in this tutorial. This hosted zone is where you store information about how to route traffic for your domain, for example, to an Amazon EC2 instance or a CloudFront distribution.

There are fees associated with domain registration, your hosted zone, and DNS queries received by your domain. For more information, see [Amazon Route 53 Pricing](#).

Note

When you register a domain, it costs money immediately and it's irreversible. You can choose not to auto-renew the domain, but you pay up front and own it for the year. For more information, see [Registering a new domain](#) in the *Amazon Route 53 Developer Guide*.

Step 1: Create an S3 bucket

Create a bucket to store the original video that you plan to stream.

To create a bucket

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. Choose **Create bucket**.

The **Create bucket** page opens.

4. For **Bucket name**, enter a name for your bucket (for example, **tutorial-bucket**).
For more information about naming buckets in Amazon S3, see [Bucket naming rules \(p. 118\)](#).
5. For **Region**, choose the AWS Region where you want the bucket to reside.
If possible, you should pick the Region that is closest to the majority of your viewers. For more information about the bucket Region, see [Buckets overview \(p. 114\)](#).
6. For **Block Public Access settings for this bucket**, keep the default settings (**Block all public access** is enabled).

Even with **Block all public access** enabled, viewers can still access the uploaded video through CloudFront. This feature is a major advantage of using CloudFront to host a video stored in S3.

We recommend that you keep all settings enabled unless you need to turn off one or more of them for your use case. For more information about blocking public access, see [Blocking public access to your Amazon S3 storage \(p. 584\)](#).

7. For the remaining settings, keep the defaults.
(Optional) If you want to configure additional bucket settings for your specific use case, see [Creating a bucket \(p. 119\)](#).
8. Choose **Create bucket**.

Step 2: Upload a video to the S3 bucket

The following procedure describes how to upload a video file to an S3 bucket by using the console. If you're uploading many large video files to S3, you might want to use [Amazon S3 Transfer Acceleration](#) to configure fast and secure file transfers. Transfer Acceleration can speed up video uploading to your S3 bucket for long-distance transfer of larger videos. For more information, see [Configuring fast, secure file transfers using Amazon S3 Transfer Acceleration \(p. 136\)](#).

To upload a file to the bucket

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. In the **Buckets** list, choose the name of the bucket that you created in [Step 1 \(p. 53\)](#) (for example, **tutorial-bucket**) to upload your file to.
4. On the **Objects** tab for your bucket, choose **Upload**.
5. On the **Upload** page, under **Files and folders**, choose **Add files**.
6. Choose a file to upload, and then choose **Open**.
For example, you can upload a video file named `sample.mp4`.
7. Choose **Upload**.

Step 3: Create a CloudFront origin access identity

To restrict direct access to the video from your S3 bucket, create a special CloudFront user called an origin access identity (OAI). You will associate the OAI with your distribution later in this tutorial. By using an OAI, you make sure that viewers can't bypass CloudFront and get the video directly from the S3 bucket. Only the CloudFront OAI can access the file in the S3 bucket. For more information, see [Restricting access to Amazon S3 content by using an OAI](#) in the *Amazon CloudFront Developer Guide*.

To create a CloudFront OAI

1. Sign in to the AWS Management Console and open the CloudFront console at <https://console.aws.amazon.com/cloudfront/v3/home>.
2. In the left navigation pane, under the **Security** section, choose **Origin access identities**.
3. Choose **Create origin access identity**.
4. Enter a name (for example, **s3-OAI**) for the new origin access identity.
5. Choose **Create**.

Step 4: Create a CloudFront distribution

To use CloudFront to serve and distribute the video in your S3 bucket, you must create a CloudFront distribution.

Substeps

- [Create a CloudFront distribution \(p. 55\)](#)
- [Review the bucket policy \(p. 55\)](#)

Create a CloudFront distribution

1. Sign in to the AWS Management Console and open the CloudFront console at <https://console.aws.amazon.com/cloudfront/v3/home>.
2. In the left navigation pane, choose **Distributions**.
3. Choose **Create distribution**.
4. In the **Origin** section, for **Origin domain**, choose the domain name of your S3 origin, which starts with the name of the S3 bucket that you created in [Step 1 \(p. 53\)](#) (for example, **tutorial-bucket**).
5. For **S3 bucket access**, choose **Yes use OAI (bucket can restrict access to only CloudFront)**.
6. Under **Origin access identity**, choose the origin access identity that you created in [Step 3 \(p. 54\)](#) (for example, **s3-OAI**).
7. Under **Bucket policy**, choose **Yes, update the bucket policy**.
8. In the **Default cache behavior** section, under **Viewer protocol policy**, choose **Redirect HTTP to HTTPS**.

When you choose this feature, HTTP requests are automatically redirected to HTTPS to secure your website and protect your viewers' data.

9. For the other settings in the **Default cache behaviors** section, keep the default values.

(Optional) You can control how long your file stays in a CloudFront cache before CloudFront forwards another request to your origin. Reducing the duration allows you to serve dynamic content. Increasing the duration means that your viewers get better performance because your files are more likely to be served directly from the edge cache. A longer duration also reduces the load on your origin. For more information, see [Managing how long content stays in the cache \(expiration\)](#) in the *Amazon CloudFront Developer Guide*.

10. For the other sections, keep the remaining settings set to the defaults.

For more information about the different settings options, see [Values That You Specify When You Create or Update a Distribution](#) in the *Amazon CloudFront Developer Guide*.

11. At the bottom of the page, choose **Create distribution**.
12. On the **General** tab for your CloudFront distribution, under **Details**, the value of the **Last modified** column for your distribution changes from **Deploying** to the timestamp when the distribution was last modified. This process typically takes a few minutes.

Review the bucket policy

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. In the **Buckets** list, choose the name of the bucket that you used earlier as the origin of your CloudFront distribution (for example, **tutorial-bucket**).

4. Choose the **Permissions** tab.
5. In the **Bucket policy** section, confirm that you see a statement similar to the following in the bucket policy text:

```
{  
    "Version": "2008-10-17",  
    "Id": "PolicyForCloudFrontPrivateContent",  
    "Statement": [  
        {  
            "Sid": "1",  
            "Effect": "Allow",  
            "Principal": {  
                "AWS": "arn:aws:iam::cloudfront:user/CloudFront Origin Access  
Identity EH1HDMB1FH2TC"  
            },  
            "Action": "s3:GetObject",  
            "Resource": "arn:aws:s3:::tutorial-bucket/*"  
        }  
    ]  
}
```

This is the statement that your CloudFront distribution added to your bucket policy when you chose **Yes, update the bucket policy** earlier.

This bucket policy update indicates that you successfully configured the CloudFront distribution to restrict access to the S3 bucket. Because of this restriction, objects in the bucket can be accessed only through your CloudFront distribution.

Step 5: Access the video through the CloudFront distribution

Now, CloudFront can serve the video stored in your S3 bucket. To access your video through CloudFront, you must combine your CloudFront distribution domain name with the path to the video in the S3 bucket.

To create a URL to the S3 video using the CloudFront distribution domain name

1. Sign in to the AWS Management Console and open the CloudFront console at <https://console.aws.amazon.com/cloudfront/v3/home>.
2. In the left navigation pane, choose **Distributions**.
3. To get the distribution domain name, do the following:
 - a. In the **Origins** column, find the correct CloudFront distribution by looking for its origin name, which starts with the S3 bucket that you created in [Step 1 \(p. 53\)](#) (for example, **tutorial-bucket**).
 - b. After finding the distribution in the list, widen the **Domain name** column to copy the domain name value for your CloudFront distribution.
4. In a new browser tab, paste the distribution domain name that you copied.
5. Return to the previous browser tab, and open the S3 console at <https://console.aws.amazon.com/s3/>.
6. In the left navigation pane, choose **Buckets**.
7. In the **Buckets** list, choose the name of the bucket that you created in [Step 1 \(p. 53\)](#) (for example, **tutorial-bucket**).

8. In the **Objects** list, choose the name of the video that you uploaded in [Step 2 \(p. 54\)](#) (for example, sample.mp4).
9. On the object detail page, in the **Object overview** section, copy the value of the **Key**. This value is the path to the uploaded video object in the S3 bucket.
10. Return to the browser tab where you previously pasted the distribution domain name, enter a forward slash (/) after the distribution domain name, and then paste the path to the video that you copied earlier (for example, sample.mp4).

Now, your S3 video is publicly accessible and hosted through CloudFront at a URL that looks similar to the following:

```
https://CloudFront distribution domain name/Path to the S3 video
```

Replace *CloudFront distribution domain name* and *Path to the S3 video* with the appropriate values. An example URL is <https://d111111abcdef8.cloudfront.net/sample.mp4>.

Step 6: Configure your CloudFront distribution to use your custom domain name

To use your own domain name instead of the CloudFront domain name in the URL to access the S3 video, add an alternate domain name to your CloudFront distribution.

Substeps

- [Request an SSL certificate \(p. 57\)](#)
- [Add the alternate domain name to your CloudFront distribution \(p. 58\)](#)
- [Create a DNS record to route traffic from your alternate domain name to your CloudFront distribution's domain name \(p. 59\)](#)
- [Check whether IPv6 is enabled for your distribution and create another DNS record if needed \(p. 59\)](#)

Request an SSL certificate

To allow your viewers to use HTTPS and your custom domain name in the URL for your video streaming, use AWS Certificate Manager (ACM) to request a Secure Sockets Layer (SSL) certificate. The SSL certificate establishes an encrypted network connection to the website.

1. Sign in to the AWS Management Console and open the ACM console at <https://console.aws.amazon.com/acm/>.
2. If the introductory page appears, under **Provision certificates**, choose **Get Started**.
3. On the **Request a certificate** page, choose **Request a public certificate**, and then choose **Request a certificate**.
4. On the **Add domain names** page, enter the fully qualified domain name (FQDN) of the site that you want to secure with an SSL/TLS certificate. You can use an asterisk (*) to request a wildcard certificate to protect several site names in the same domain. For this tutorial, enter * and the custom domain name that you configured in [Prerequisites \(p. 52\)](#). For example, enter *.example.com, and then choose **Next**.

For more information, see [To request an ACM public certificate \(console\)](#) in the *AWS Certificate Manager User Guide*.

5. On the **Select validation method** page, choose **DNS validation**. Then, choose **Next**.

If you are able to edit your DNS configuration, we recommend that you use DNS domain validation rather than email validation. DNS validation has multiple benefits over email validation. For more information, see [Option 1: DNS validation](#) in the *AWS Certificate Manager User Guide*.

6. (Optional) On the **Add tags** page, tag your certificate with metadata.
7. Choose **Review**.
8. On the **Review** page, verify that the information under **Domain name** and **Validation method** are correct. Then, choose **Confirm and request**.

The **Validation** page shows that your request is being processed and that the certificate domain is being validated. The certificate awaiting validation is in the **Pending validation** status.

9. On the **Validation** page, choose the down arrow to the left of your custom domain name, and then choose **Create record in Route 53** to validate your domain ownership through DNS.

Doing this adds a CNAME record provided by AWS Certificate Manager to your DNS configuration.

10. In the **Create record in Route 53** dialog box, choose **Create**.

The **Validation** page should display a status notification of **Success** at the bottom.

11. Choose **Continue** to view the **Certificates** list page.

The **Status** for your new certificate changes from **Pending validation** to **Issued** within 30 minutes.

Add the alternate domain name to your CloudFront distribution

1. Sign in to the AWS Management Console and open the CloudFront console at <https://console.aws.amazon.com/cloudfront/v3/home>.
2. In the left navigation pane, choose **Distributions**.
3. Choose the ID for the distribution that you created in [Step 4 \(p. 54\)](#).
4. On the **General** tab, go to the **Settings** section, and choose **Edit**.
5. On the **Edit settings** page, for **Alternate domain name (CNAME) - optional**, choose **Add item** to add the custom domain names that you want to use in the URL for the S3 video served by this CloudFront distribution.

In this tutorial, for example, if you want to route traffic for a subdomain, such as `www.example.com`, enter the subdomain name (`www`) with the domain name (`example.com`). Specifically, enter `www.example.com`.

Note

The alternate domain name (CNAME) that you add must be covered by the SSL certificate that you previously attached to your CloudFront distribution.

6. For **Custom SSL certificate - optional**, choose the SSL certificate that you requested earlier (for example, `*.example.com`).

Note

If you don't see the SSL certificate immediately after you request it, wait 30 minutes, and then refresh the list until the SSL certificate is available for you to select.

7. Keep the remaining settings set to the defaults. Choose **Save changes**.
8. On the **General** tab for the distribution, wait for the value of **Last modified** to change from **Deploying** to the timestamp when the distribution was last modified.

Create a DNS record to route traffic from your alternate domain name to your CloudFront distribution's domain name

1. Sign in to the AWS Management Console and open the Route 53 console at <https://console.aws.amazon.com/route53/>.
2. In the left navigation pane, choose **Hosted zones**.
3. On the **Hosted zones** page, choose the name of the hosted zone that Route 53 created for you in [Prerequisites \(p. 52\)](#) (for example, `example.com`).
4. Choose **Create record**, and then use the **Quick create record** method.
5. For **Record name**, keep the value for the record name the same as the alternate domain name of the CloudFront distribution that you added earlier.

In this tutorial, to route traffic to a subdomain, such as `www.example.com`, enter the subdomain name without the domain name. For example, enter only `www` in the text field before your custom domain name.

6. For **Record type**, choose **A - Routes traffic to an IPv4 address and some AWS resources**.
7. For **Value**, choose the **Alias** toggle to enable the alias resource.
8. Under **Route traffic to**, choose **Alias to CloudFront distribution** from the dropdown list.
9. In the search box that says **Choose distribution**, choose the domain name of the CloudFront distribution that you created in [Step 4 \(p. 55\)](#).

To find the domain name of your CloudFront distribution, do the following:

- a. In a new browser tab, sign in to the AWS Management Console and open the CloudFront console at <https://console.aws.amazon.com/cloudfront/v3/home>.
 - b. In the left navigation pane, choose **Distributions**.
 - c. In the **Origins** column, find the correct CloudFront distribution by looking for its origin name, which starts with the S3 bucket that you created in [Step 1 \(p. 53\)](#) (for example, `tutorial-bucket`).
 - d. After finding the distribution in the list, widen the **Domain name** column to see the domain name value for your CloudFront distribution.
10. On the **Create record** page in the Route 53 console, for the remaining settings, keep the defaults.
 11. Choose **Create records**.

Check whether IPv6 is enabled for your distribution and create another DNS record if needed

If IPv6 is enabled for your distribution, you must create another DNS record.

1. To check whether IPv6 is enabled for your distribution, do the following:
 - a. Sign in to the AWS Management Console and open the CloudFront console at <https://console.aws.amazon.com/cloudfront/v3/home>.
 - b. In the left navigation pane, choose **Distributions**.
 - c. Choose the ID of the CloudFront distribution that you created in [Step 4 \(p. 55\)](#).
 - d. On the **General** tab, under **Settings**, check whether **IPv6** is set to **Enabled**.

If IPv6 is enabled for your distribution, you must create another DNS record.

2. If IPv6 is enabled for your distribution, do the following to create a DNS record:

- a. Sign in to the AWS Management Console and open the Route 53 console at <https://console.aws.amazon.com/route53/>.
- b. In the left navigation pane, choose **Hosted zones**.
- c. On the **Hosted zones** page, choose the name of the hosted zone that Route 53 created for you in [Prerequisites \(p. 52\)](#) (for example, `example.com`).
- d. Choose **Create record**, and then use the **Quick create record** method.
- e. For **Record name**, in the text field before your custom domain name, type the same value that you typed when you created the IPv4 DNS record earlier. For example, in this tutorial, to route traffic for the subdomain `www.example.com`, enter only `www`.
- f. For **Record type**, choose **AAAA - Routes traffic to an IPv6 address and some AWS resources**.
- g. For **Value**, choose the **Alias** toggle to enable the alias resource.
- h. Under **Route traffic to**, choose **Alias to CloudFront distribution** from the dropdown list.
- i. In the search box that says **Choose distribution**, choose the domain name of the CloudFront distribution that you created in [Step 4 \(p. 55\)](#).
- j. For the remaining settings, keep the defaults.
- k. Choose **Create records**.

Step 7: Access the S3 video through the CloudFront distribution with the custom domain name

To access the S3 video using the custom URL, you must combine your alternate domain name with the path to the video in the S3 bucket.

To create a custom URL to access the S3 video through the CloudFront distribution

1. Sign in to the AWS Management Console and open the CloudFront console at <https://console.aws.amazon.com/cloudfront/v3/home>.
2. In the left navigation pane, choose **Distributions**.
3. To get the alternate domain name of your CloudFront distribution, do the following:
 - a. In the **Origins** column, find the correct CloudFront distribution by looking for its origin name, which starts with the S3 bucket name for the bucket that you created in [Step 1 \(p. 53\)](#) (for example, `tutorial-bucket`).
 - b. After finding the distribution in the list, widen the **Alternate domain names** column to copy the value of the alternate domain name of your CloudFront distribution.
4. In a new browser tab, paste the alternate domain name of the CloudFront distribution.
5. Return to the previous browser tab, and open the Amazon S3 console at <https://console.aws.amazon.com/s3/>.
6. Find the path to your S3 video, as explained in [Step 5 \(p. 56\)](#).
7. Return to the browser tab where you previously pasted the alternate domain name, enter a forward slash (/), and then paste the path to your S3 video (for example, `sample.mp4`).

Now, your S3 video is publicly accessible and hosted through CloudFront at a custom URL that looks similar to the following:

`https://CloudFront distribution alternate domain name/Path to the S3 video`

Replace `CloudFront distribution alternate domain name` and `Path to the S3 video` with the appropriate values. An example URL is <https://www.example.com/sample.mp4>.

(Optional) Step 8: View data about requests received by your CloudFront distribution

To view data about requests received by your CloudFront distribution

1. Sign in to the AWS Management Console and open the CloudFront console at <https://console.aws.amazon.com/cloudfront/v3/home>.
2. In the left navigation pane, under **Reports & analytics**, choose the reports from the console, ranging from **Cache statistics**, **Popular Objects**, **Top Referrers**, **Usage**, and **Viewers**.
You can filter each report dashboard. For more information, see [CloudFront Reports in the Console in the Amazon CloudFront Developer Guide](#).
3. To filter data, choose the ID of the CloudFront distribution that you created in [Step 4 \(p. 55\)](#).

Step 9: Clean up

If you hosted an S3 streaming video using CloudFront and Route 53 only as a learning exercise, delete the AWS resources that you allocated so that you no longer accrue charges.

Note

When you register a domain, it costs money immediately and it's irreversible. You can choose not to auto-renew the domain, but you pay up front and own it for the year. For more information, see [Registering a new domain](#) in the *Amazon Route 53 Developer Guide*.

Substeps

- [Delete the CloudFront distribution \(p. 61\)](#)
- [Delete the DNS record \(p. 62\)](#)
- [Delete the public hosted zone for your custom domain \(p. 62\)](#)
- [Delete the custom domain name from Route 53 \(p. 63\)](#)
- [Delete the original video in the S3 source bucket \(p. 63\)](#)
- [Delete the S3 source bucket \(p. 64\)](#)

Delete the CloudFront distribution

1. Sign in to the AWS Management Console and open the CloudFront console at <https://console.aws.amazon.com/cloudfront/v3/home>.
2. In the left navigation pane, choose **Distributions**.
3. In the **Origins** column, find the correct CloudFront distribution by looking for its origin name, which starts with the S3 bucket name for the bucket that you created in [Step 1 \(p. 53\)](#) (for example, **tutorial-bucket**).
4. To delete the CloudFront distribution, you must disable it first.
 - If the value of the **Status** column is **Enabled** and the value of **Last modified** is the timestamp when the distribution was last modified, continue to disable the distribution before deleting it.
 - If the value of **Status** is **Enabled** and the value of **Last modified** is **Deploying**, wait until the value of **Status** changes to the timestamp when the distribution was last modified. Then continue to disable the distribution before deleting it.
5. To disable the CloudFront distribution, do the following:
 - a. In the **Distributions** list, select the check box next to the ID for the distribution that you want to delete.

- b. To disable the distribution, choose **Disable**, and then choose **Delete** to confirm.

If you disable a distribution that has an alternate domain name associated with it, CloudFront stops accepting traffic for that domain name (such as `www.example.com`), even if another distribution has an alternate domain name with a wildcard (*) that matches the same domain (such as `*.example.com`).

- c. The value of **Status** immediately changes to **Disabled**. Wait until the value of **Last modified** changes from **Deploying** to the timestamp when the distribution was last modified.

Because CloudFront must propagate this change to all edge locations, it might take a few minutes before the update is complete and the **Delete** option is available for you to delete the distribution.

6. To delete the disabled distribution, do the following:

- a. Choose the check box next to the ID for the distribution that you want to delete.
- b. Choose **Delete**, and then choose **Delete** to confirm.

Delete the DNS record

If you want to delete the public hosted zone for the domain (including the DNS record), see [Delete the public hosted zone for your custom domain \(p. 62\)](#) in the *Amazon Route 53 Developer Guide*. If you only want to delete the DNS record created in [Step 6 \(p. 57\)](#), do the following:

1. Sign in to the AWS Management Console and open the Route 53 console at <https://console.aws.amazon.com/route53/>.
2. In the left navigation pane, choose **Hosted zones**.
3. On the **Hosted zones** page, choose the name of the hosted zone that Route 53 created for you in [Prerequisites \(p. 52\)](#) (for example, `example.com`).
4. In the list of records, select the check box next to the records that you want to delete (the records that you created in [Step 6 \(p. 57\)](#)).

Note

You can't delete records that have a **Type** value of **NS** or **SOA**.

5. Choose **Delete records**.
6. To confirm the deletion, choose **Delete**.

Changes to records take time to propagate to the Route 53 DNS servers. Currently, the only way to verify that your changes have propagated is to use the [GetChange API action](#). Changes usually propagate to all Route 53 name servers within 60 seconds.

Delete the public hosted zone for your custom domain

Warning

If you want to keep your domain registration but stop routing internet traffic to your website or web application, we recommend that you delete records in the hosted zone (as described in the prior section) instead of deleting the hosted zone.

If you delete a hosted zone, someone else can use the domain and route traffic to their own resources using your domain name.

In addition, if you delete a hosted zone, you can't undelete it. You must create a new hosted zone and update the name servers for your domain registration, which can take up to 48 hours to take effect.

If you want to make the domain unavailable on the internet, you can first transfer your DNS service to a free DNS service and then delete the Route 53 hosted zone. This prevents future DNS queries from possibly being misrouted.

1. If the domain is registered with Route 53, see [Adding or changing name servers and glue records for a domain](#) in the *Amazon Route 53 Developer Guide* for information about how to replace Route 53 name servers with name servers for the new DNS service.
2. If the domain is registered with another registrar, use the method provided by the registrar to change name servers for the domain.

Note

If you're deleting a hosted zone for a subdomain (`www.example.com`), you don't need to change name servers for the domain (`example.com`).

1. Sign in to the AWS Management Console and open the Route 53 console at <https://console.aws.amazon.com/route53/>.
2. In the left navigation pane, choose **Hosted zones**.
3. On the **Hosted zones** page, choose the name of the hosted zone that you want to delete.
4. On the **Records** tab for your hosted zone, confirm that the hosted zone that you want to delete contains only an **NS** and an **SOA** record.

If it contains additional records, delete them first.

If you created any NS records for subdomains in the hosted zone, delete those records too.

5. On the **DNSSEC signing** tab for your hosted zone, disable DNSSSEC signing if it was enabled. For more information, see [Disabling DNSSEC signing](#) in the *Amazon Route 53 Developer Guide*.
6. At the top of the details page of the hosted zone, choose **Delete zone**.
7. To confirm the deletion, enter **delete**, and then choose **Delete**.

Delete the custom domain name from Route 53

For most top-level domains (TLDs), you can delete the registration if you no longer want it. If you delete a domain name registration from Route 53 before the registration is scheduled to expire, AWS does not refund the registration fee. For more information, see [Deleting a domain name registration](#) in the *Amazon Route 53 Developer Guide*.

Important

If you want to transfer the domain between AWS accounts or transfer the domain to another registrar, don't delete the domain and expect to immediately reregister it. Instead, see the applicable documentation in the *Amazon Route 53 Developer Guide*:

- [Transferring a domain to a different AWS account](#)
- [Transferring a domain from Amazon Route 53 to another registrar](#)

Delete the original video in the S3 source bucket

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. In the **Bucket name** list, choose the name of the bucket that you uploaded the video to in [Step 2 \(p. 54\)](#) (for example, `tutorial-bucket`).
4. On the **Objects** tab, select the check box next to the name of the object that you want to delete (for example, `sample.mp4`).
5. Choose **Delete**.
6. Under **Permanently delete objects?**, enter `permanently delete` to confirm that you want to delete this object.

7. Choose **Delete objects**.

Delete the S3 source bucket

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. In the **Buckets** list, select the option button next to the name of the bucket that you created in [Step 1 \(p. 53\)](#) (for example, `tutorial-bucket`).
4. Choose **Delete**.
5. On the **Delete bucket** page, confirm that you want to delete the bucket by entering the bucket name in the text field, and then choose **Delete bucket**.

Next steps

After you complete this tutorial, you can further explore the following related use cases:

- Transcode S3 videos into streaming formats needed by a particular television or connected device before hosting these videos with a CloudFront distribution.

To use Amazon S3 Batch Operations, AWS Lambda and AWS Elemental MediaConvert to batch-transcode a collection of videos to a variety of output media formats, see [Tutorial: Batch-transcoding videos with S3 Batch Operations, AWS Lambda, and AWS Elemental MediaConvert \(p. 64\)](#).

- Host other objects stored in S3, such as images, audio, motion graphics, style sheets, HTML, JavaScript, React apps, and so on, using CloudFront and Route 53.

For example, see [Configuring a static website using a custom domain registered with Route 53 \(p. 97\)](#) and [Speeding up your website with Amazon CloudFront \(p. 109\)](#).

- Use [Amazon S3 Transfer Acceleration](#) to configure fast and secure file transfers. Transfer Acceleration can speed up video uploading to your S3 bucket for long-distance transfer of larger videos. Transfer Acceleration improves transfer performance by routing traffic through the CloudFront globally distributed edge locations and over the AWS backbone networks. It also uses network protocol optimizations. For more information, see [Configuring fast, secure file transfers using Amazon S3 Transfer Acceleration \(p. 136\)](#).

Tutorial: Batch-transcoding videos with S3 Batch Operations, AWS Lambda, and AWS Elemental MediaConvert

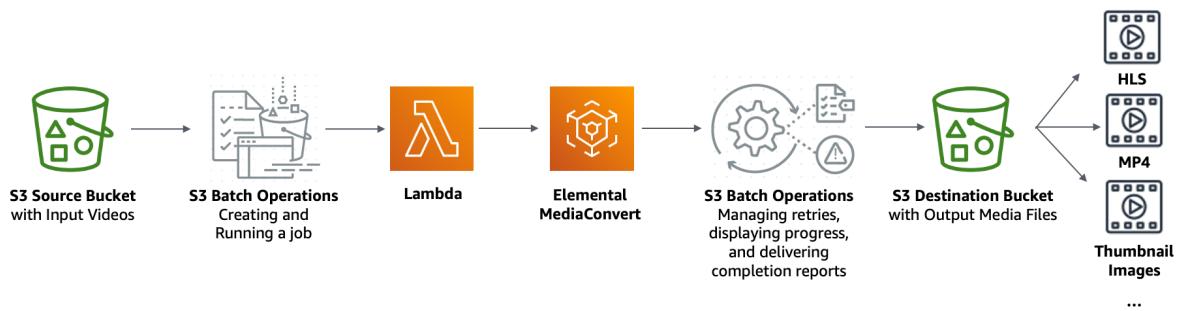
Video consumers use devices of all shapes, sizes, and vintages to enjoy media content. This wide array of devices presents a challenge for content creators and distributors. Instead of being in a one-size-fits-all format, videos must be converted so that they can span a broad range of sizes, formats, and bitrates. This conversion task is even more challenging when you have a large number of videos that must be converted.

AWS offers you a method to build a scalable, distributed architecture that does the following:

- Ingests input videos
- Processes the videos for playback on a wide range of devices

- Stores the transcoded media files
- Delivers the output media files to meet demand

When you have extensive video repositories stored in Amazon S3, you can transcode these videos from their source formats into multiple file types in the size, resolution, and format needed by a particular video player or device. Specifically, [S3 Batch Operations](#) provides you with a solution to invoke AWS Lambda functions for existing input videos in an S3 source bucket. Then, the Lambda functions call [AWS Elemental MediaConvert](#) to perform large-scale video transcoding tasks. The converted output media files are stored in an S3 destination bucket.



Objective

In this tutorial, you learn how to set up S3 Batch Operations to invoke a Lambda function for batch-transcoding of videos stored in an S3 source bucket. The Lambda function calls MediaConvert to transcode the videos. The outputs for each video in the S3 source bucket are as follows:

- An [HTTP Live Streaming \(HLS\)](#) adaptive bitrate stream for playback on devices of multiple sizes and varying bandwidths
- An MP4 video file
- Thumbnail images collected at intervals

Topics

- [Prerequisites \(p. 65\)](#)
- [Step 1: Create an S3 bucket for the output media files \(p. 66\)](#)
- [Step 2: Create an IAM role for MediaConvert \(p. 67\)](#)
- [Step 3: Create an IAM role for your Lambda function \(p. 68\)](#)
- [Step 4: Create a Lambda function for video transcoding \(p. 69\)](#)
- [Step 5: Configure Amazon S3 Inventory for your S3 source bucket \(p. 81\)](#)
- [Step 6: Create an IAM role for S3 Batch Operations \(p. 84\)](#)
- [Step 7: Create and run an S3 Batch Operations job \(p. 86\)](#)
- [Step 8: Check the output media files from your S3 destination bucket \(p. 89\)](#)
- [Step 9: Clean up \(p. 90\)](#)
- [Next steps \(p. 92\)](#)

Prerequisites

Before you start this tutorial, you must have an Amazon S3 source bucket (for example, `tutorial-bucket-1`) with videos to be transcoded already stored in it.

You can give the bucket another name if you want. For more information about bucket names in Amazon S3, see [Bucket naming rules \(p. 118\)](#).

For the S3 source bucket, keep the settings related to **Block Public Access settings for this bucket** set to the defaults (**Block all public access** is enabled). For more information, see [Creating a bucket \(p. 119\)](#).

For more information about uploading videos to the S3 source bucket, see [Uploading objects \(p. 158\)](#). If you're uploading many large video files to S3, you might want to use [Amazon S3 Transfer Acceleration](#) to configure fast and secure file transfers. Transfer Acceleration can speed up video uploading to your S3 bucket for long-distance transfer of larger videos. For more information, see [Configuring fast, secure file transfers using Amazon S3 Transfer Acceleration \(p. 136\)](#).

Step 1: Create an S3 bucket for the output media files

In this step, you create an S3 destination bucket to store the converted output media files. You also create a Cross Origin Resource Sharing (CORS) configuration to allow cross-origin access to the transcoded media files stored in your S3 destination bucket.

Substeps

- [Create a bucket for the output media files \(p. 66\)](#)
- [Add a CORS configuration to the S3 output bucket \(p. 66\)](#)

Create a bucket for the output media files

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. Choose **Create bucket**.
4. For **Bucket name**, enter a name for your bucket (for example, `tutorial-bucket-2`).
5. For **Region**, choose the AWS Region where you want the bucket to reside.
6. To ensure public access to your output media files, in **Block Public Access settings for this bucket**, clear **Block all public access**.

Warning

Before you complete this step, review [Blocking public access to your Amazon S3 storage \(p. 584\)](#) to ensure that you understand and accept the risks involved with allowing public access. When you turn off Block Public Access settings to make your bucket public, anyone on the internet can access your bucket. We recommend that you block all public access to your buckets.

If you don't want to clear the Block Public Access settings, you can use Amazon CloudFront to deliver the transcoded media files to viewers (end users). For more information, see [Tutorial: Hosting on-demand streaming video with Amazon S3, Amazon CloudFront, and Amazon Route 53 \(p. 51\)](#).

7. Select the check box next to **I acknowledge that the current settings might result in this bucket and the objects within becoming public**.
8. Keep the remaining settings set to the defaults.
9. Choose **Create bucket**.

Add a CORS configuration to the S3 output bucket

A JSON CORS configuration defines a way for client web applications (video players in this context) that are loaded in one domain to play transcoded output media files in a different domain.

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. In the **Buckets** list, choose the name of the bucket that you created earlier (for example, `tutorial-bucket-2`).
4. Choose the **Permissions** tab.
5. In the **Cross-origin resource sharing (CORS)** section, choose **Edit**.
6. In the CORS configuration text box, copy and paste the following CORS configuration.

The CORS configuration must be in JSON format. In this example, the `AllowedOrigins` attribute uses the wildcard character (*) to specify all origins. If you know your specific origin, you can restrict the `AllowedOrigins` attribute to your specific player URL. For more information about configuring this and other attributes, see [CORS configuration \(p. 574\)](#).

```
[  
  {  
    "AllowedOrigins": [  
      "*"  
    ],  
    "AllowedMethods": [  
      "GET"  
    ],  
    "AllowedHeaders": [  
      "*"  
    ],  
    "ExposeHeaders": []  
  }  
]
```

7. Choose **Save changes**.

Step 2: Create an IAM role for MediaConvert

To use AWS Elemental MediaConvert to transcode input videos stored in your S3 bucket, you must have an AWS Identity and Access Management (IAM) service role to grant MediaConvert permissions to read and write video files from and to your S3 source and destination buckets. When you run transcoding jobs, the MediaConvert console uses this role.

To create an IAM role for MediaConvert

1. Create an IAM role with a role name that you choose (for example, `tutorial-mediaconvert-role`). To create this role, follow the steps in [Create your MediaConvert role in IAM \(console\)](#) in the *AWS Elemental MediaConvert User Guide*.
2. After you create the IAM role for MediaConvert, in the list of **Roles**, choose the name of the role for MediaConvert that you created (for example, `tutorial-mediaconvert-role`).
3. On the **Summary** page, copy the **Role ARN** (which starts with `arn:aws:iam::`), and save the ARN for use later.

For more information about ARNs, see [Amazon Resource Names \(ARNs\)](#) in the *AWS General Reference*.

Step 3: Create an IAM role for your Lambda function

To batch-transcode videos with MediaConvert and S3 Batch Operations, you use a Lambda function to connect these two services to convert videos. This Lambda function must have an IAM role that grants the Lambda function permissions to access MediaConvert and S3 Batch Operations.

Substeps

- [Create an IAM role for your Lambda function \(p. 68\)](#)
- [Embed an inline policy for the IAM role of your Lambda function \(p. 68\)](#)

Create an IAM role for your Lambda function

1. Sign in to the AWS Management Console and open the IAM console at <https://console.aws.amazon.com/iam/>.
2. In the left navigation pane, choose **Roles**, and then choose **Create role**.
3. Choose the **AWS service** role type, and then under **Common use cases**, choose **Lambda**.
4. Choose **Next: Permissions**.
5. On the **Attach permissions policies** page, enter **AWSLambdaBasicExecutionRole** in the **Filter policies** box. To attach the managed policy **AWSLambdaBasicExecutionRole** to this role to grant write permissions to Amazon CloudWatch Logs, select the check box next to **AWSLambdaBasicExecutionRole**.
6. Choose **Next: Tags**.
7. (Optional) Add tags to the managed policy.
8. Choose **Next: Review**.
9. For **Role name**, enter **tutorial-lambda-transcode-role**.
10. Choose **Create role**.

Embed an inline policy for the IAM role of your Lambda function

To grant permissions to the MediaConvert resource that's needed for the Lambda function to execute, you must use an inline policy.

1. Sign in to the AWS Management Console and open the IAM console at <https://console.aws.amazon.com/iam/>.
2. In the left navigation pane, choose **Roles**.
3. In the **Roles** list, choose the name of the IAM role that you created earlier for your Lambda function (for example, **tutorial-lambda-transcode-role**).
4. Choose the **Permissions** tab.
5. Choose **Add inline policy**.
6. Choose the **JSON** tab, and then copy and paste the following JSON policy.

In the JSON policy, replace the example ARN value of **Resource** with the role ARN of the IAM role for MediaConvert that you created in [Step 2 \(p. 67\)](#) (for example, **tutorial-mediaconvert-role**).

```
{  
    "Version": "2012-10-17",  
    "Statement": [  
        {
```

```
        "Action": [
            "logs>CreateLogGroup",
            "logs>CreateLogStream",
            "logs>PutLogEvents"
        ],
        "Resource": "*",
        "Effect": "Allow",
        "Sid": "Logging"
    },
    {
        "Action": [
            "iam>PassRole"
        ],
        "Resource": [
            "arn:aws:iam::111122223333:role/tutorial-mediainfo-role"
        ],
        "Effect": "Allow",
        "Sid": "PassRole"
    },
    {
        "Action": [
            "mediainfo:*"
        ],
        "Resource": [
            "*"
        ],
        "Effect": "Allow",
        "Sid": "MediaInfoService"
    },
    {
        "Action": [
            "s3:)"
        ],
        "Resource": [
            "*"
        ],
        "Effect": "Allow",
        "Sid": "S3Service"
    }
]
```

7. Choose **Review Policy**.
8. For **Name**, enter **tutorial-lambda-policy**.
9. Choose **Create Policy**.

After you create an inline policy, it is automatically embedded in the IAM role of your Lambda function.

Step 4: Create a Lambda function for video transcoding

In this section of the tutorial, you build a Lambda function using the SDK for Python to integrate with S3 Batch Operations and MediaConvert. To start transcoding the videos already stored in your S3 source bucket, you run an S3 Batch Operations job that directly invokes the Lambda function for each video in the S3 source bucket. Then, the Lambda function submits a transcoding job for each video to MediaConvert.

Substeps

- [Write Lambda function code and create a deployment package \(p. 70\)](#)

- [Create a Lambda function with an execution role \(console\) \(p. 80\)](#)
- [Deploy your Lambda function with .zip file archives and configure the Lambda function \(console\) \(p. 80\)](#)

Write Lambda function code and create a deployment package

1. On your local machine, create a folder named `batch-transcode`.
2. In the `batch-transcode` folder, create a file with JSON job settings. For example, you can use the settings provided in this section, and name the file `job.json`.

A `job.json` file specifies the following:

- Which files to transcode
- How you want to transcode your input videos
- What output media files you want to create
- What to name the transcoded files
- Where to save the transcoded files
- Which advanced features to apply, and so on

In this tutorial, we use the following `job.json` file to create the following outputs for each video in the S3 source bucket:

- An HTTP Live Streaming (HLS) adaptive bitrate stream for playback on multiple devices of differing sizes and varying bandwidths
- An MP4 video file
- Thumbnail images collected at intervals

This example `job.json` file uses Quality-Defined Variable Bitrate (QVBR) to optimize video quality. The HLS output is Apple-compliant (audio unmixed from video, segment duration of 6 seconds, and optimized video quality through auto QVBR).

If you don't want to use the example settings provided here, you can generate a `job.json` specification based on your use case. To ensure consistency across your outputs, make sure that your input files have similar video and audio configurations. For any input files with different video and audio configurations, create separate automations (unique `job.json` settings). For more information, see [Example AWS Elemental MediaConvert job settings in JSON](#) in the *AWS Elemental MediaConvert User Guide*.

```
{  
    "OutputGroups": [  
        {  
            "CustomName": "HLS",  
            "Name": "Apple HLS",  
            "Outputs": [  
                {  
                    "ContainerSettings": {  
                        "Container": "M3U8",  
                        "M3u8Settings": {  
                            "AudioFramesPerPes": 4,  
                            "PcrControl": "PCR_EVERY_PES_PACKET",  
                            "PmtPid": 480,  
                            "PrivateMetadataPid": 503,  
                            "ProgramNumber": 1,  
                            "PatInterval": 0,  
                            "PmtInterval": 0,  
                            "SegmentDuration": 6,  
                            "SegmentStartOffset": 0  
                        }  
                    }  
                }  
            ]  
        }  
    ]  
}
```

```
"TimedMetadata": "NONE",
"VideoPid": 481,
"AudioPids": [
    482,
    483,
    484,
    485,
    486,
    487,
    488,
    489,
    490,
    491,
    492
],
},
"VideoDescription": {
    "Width": 640,
    "ScalingBehavior": "DEFAULT",
    "Height": 360,
    "TimecodeInsertion": "DISABLED",
    "AntiAlias": "ENABLED",
    "Sharpness": 50,
    "CodecSettings": {
        "Codec": "H_264",
        "H264Settings": {
            "InterlaceMode": "PROGRESSIVE",
            "NumberReferenceFrames": 3,
            "Syntax": "DEFAULT",
            "Softness": 0,
            "GopClosedCadence": 1,
            "GopSize": 2,
            "Slices": 1,
            "GopBReference": "DISABLED",
            "MaxBitrate": 1200000,
            "SlowPal": "DISABLED",
            "SpatialAdaptiveQuantization": "ENABLED",
            "TemporalAdaptiveQuantization": "ENABLED",
            "FlickerAdaptiveQuantization": "DISABLED",
            "EntropyEncoding": "CABAC",
            "FramerateControl": "INITIALIZE_FROM_SOURCE",
            "RateControlMode": "QVBR",
            "CodecProfile": "MAIN",
            "Telecine": "NONE",
            "MinIInterval": 0,
            "AdaptiveQuantization": "HIGH",
            "CodecLevel": "AUTO",
            "FieldEncoding": "PAFF",
            "SceneChangeDetect": "TRANSITION_DETECTION",
            "QualityTuningLevel": "SINGLE_PASS_HQ",
            "FramerateConversionAlgorithm": "DUPLICATE_DROP",
            "UnregisteredSeiTTimecode": "DISABLED",
            "GopSizeUnits": "SECONDS",
            "ParControl": "INITIALIZE_FROM_SOURCE",
            "NumberBFramesBetweenReferenceFrames": 2,
            "RepeatPps": "DISABLED"
        }
    },
    "AfdSignaling": "NONE",
    "DropFrameTimecode": "ENABLED",
    "RespondToAfd": "NONE",
    "ColorMetadata": "INSERT"
},
"OutputSettings": {
    "HlsSettings": {
```

```
        "AudioGroupId": "program_audio",
        "AudioRenditionSets": "program_audio",
        "SegmentModifier": "$dt$",
        "IFrameOnlyManifest": "EXCLUDE"
    }
},
"NameModifier": "_360"
},
{
"ContainerSettings": {
    "Container": "M3U8",
    "M3u8Settings": {
        "AudioFramesPerPes": 4,
        "PcrControl": "PCR_EVERY_PES_PACKET",
        "PmtPid": 480,
        "PrivateMetadataPid": 503,
        "ProgramNumber": 1,
        "PatInterval": 0,
        "PmtInterval": 0,
        "TimedMetadata": "NONE",
        "TimedMetadataPid": 502,
        "VideoPid": 481,
        "AudioPids": [
            482,
            483,
            484,
            485,
            486,
            487,
            488,
            489,
            490,
            491,
            492
        ]
    }
},
"VideoDescription": {
    "Width": 960,
    "ScalingBehavior": "DEFAULT",
    "Height": 540,
    "TimecodeInsertion": "DISABLED",
    "AntiAlias": "ENABLED",
    "Sharpness": 50,
    "CodecSettings": {
        "Codec": "H_264",
        "H264Settings": {
            "InterlaceMode": "PROGRESSIVE",
            "NumberReferenceFrames": 3,
            "Syntax": "DEFAULT",
            "Softness": 0,
            "GopClosedCadence": 1,
            "GopSize": 2,
            "Slices": 1,
            "GopBReference": "DISABLED",
            "MaxBitrate": 3500000,
            "SlowPal": "DISABLED",
            "SpatialAdaptiveQuantization": "ENABLED",
            "TemporalAdaptiveQuantization": "ENABLED",
            "FlickerAdaptiveQuantization": "DISABLED",
            "EntropyEncoding": "CABAC",
            "FramerateControl": "INITIALIZE_FROM_SOURCE",
            "RateControlMode": "QVBR",
            "CodecProfile": "MAIN",
            "Telecine": "NONE",
            "MinIInterval": 0,
            "MaxBReference": 1
        }
    }
}
```

```
        "AdaptiveQuantization": "HIGH",
        "CodecLevel": "AUTO",
        "FieldEncoding": "PAFF",
        "SceneChangeDetect": "TRANSITION_DETECTION",
        "QualityTuningLevel": "SINGLE_PASS_HQ",
        "FramerateConversionAlgorithm": "DUPLICATE_DROP",
        "UnregisteredSeiTimecode": "DISABLED",
        "GopSizeUnits": "SECONDS",
        "ParControl": "INITIALIZE_FROM_SOURCE",
        "NumberBFramesBetweenReferenceFrames": 2,
        "RepeatPps": "DISABLED"
    }
},
"AfdSignaling": "NONE",
"DropFrameTimecode": "ENABLED",
"RespondToAfd": "NONE",
"ColorMetadata": "INSERT"
},
"OutputSettings": {
    "HlsSettings": {
        "AudioGroupId": "program_audio",
        "AudioRenditionSets": "program_audio",
        "SegmentModifier": "$dt$",
        "IFrameOnlyManifest": "EXCLUDE"
    }
},
"NameModifier": "_540"
},
{
    "ContainerSettings": {
        "Container": "M3U8",
        "M3u8Settings": {
            "AudioFramesPerPes": 4,
            "PcrControl": "PCR_EVERY_PES_PACKET",
            "PmtPid": 480,
            "PrivateMetadataPid": 503,
            "ProgramNumber": 1,
            "PatInterval": 0,
            "PmtInterval": 0,
            "TimedMetadata": "NONE",
            "VideoPid": 481,
            "AudioPids": [
                482,
                483,
                484,
                485,
                486,
                487,
                488,
                489,
                490,
                491,
                492
            ]
        }
    },
    "VideoDescription": {
        "Width": 1280,
        "ScalingBehavior": "DEFAULT",
        "Height": 720,
        "TimecodeInsertion": "DISABLED",
        "AntiAlias": "ENABLED",
        "Sharpness": 50,
        "CodecSettings": {
            "Codec": "H_264",
            "H264Settings": {
```

```
        "InterlaceMode": "PROGRESSIVE",
        "NumberReferenceFrames": 3,
        "Syntax": "DEFAULT",
        "Softness": 0,
        "GopClosedCadence": 1,
        "GopSize": 2,
        "Slices": 1,
        "GopBReference": "DISABLED",
        "MaxBitrate": 5000000,
        "SlowPal": "DISABLED",
        "SpatialAdaptiveQuantization": "ENABLED",
        "TemporalAdaptiveQuantization": "ENABLED",
        "FlickerAdaptiveQuantization": "DISABLED",
        "EntropyEncoding": "CABAC",
        "FramerateControl": "INITIALIZE_FROM_SOURCE",
        "RateControlMode": "QVBR",
        "CodecProfile": "MAIN",
        "Telecine": "NONE",
        "MinIInterval": 0,
        "AdaptiveQuantization": "HIGH",
        "CodecLevel": "AUTO",
        "FieldEncoding": "PAFF",
        "SceneChangeDetect": "TRANSITION_DETECTION",
        "QualityTuningLevel": "SINGLE_PASS_HQ",
        "FramerateConversionAlgorithm": "DUPLICATE_DROP",
        "UnregisteredSeiTimecode": "DISABLED",
        "GopSizeUnits": "SECONDS",
        "ParControl": "INITIALIZE_FROM_SOURCE",
        "NumberBFramesBetweenReferenceFrames": 2,
        "RepeatPps": "DISABLED"
    },
},
"AfdSignaling": "NONE",
"DropFrameTimecode": "ENABLED",
"RespondToAfd": "NONE",
"ColorMetadata": "INSERT"
},
"OutputSettings": {
    "HlsSettings": {
        "AudioGroupId": "program_audio",
        "AudioRenditionSets": "program_audio",
        "SegmentModifier": "$dt$",
        "IFrameOnlyManifest": "EXCLUDE"
    }
},
"NameModifier": "_720"
},
{
    "ContainerSettings": {
        "Container": "M3U8",
        "M3u8Settings": {}
    },
    "AudioDescriptions": [
        {
            " AudioSourceName": "Audio Selector 1",
            " CodecSettings": {
                "Codec": "AAC",
                "AacSettings": {
                    "Bitrate": 96000,
                    "CodingMode": "CODING_MODE_2_0",
                    "SampleRate": 48000
                }
            }
        }
    ],
    "OutputSettings": {
```

```

        "HlsSettings": {
            "AudioGroupId": "program_audio",
            "AudioTrackType": "ALTERNATE_AUDIO_AUTO_SELECT_DEFAULT"
        },
        "NameModifier": "_audio"
    }
],
"OutputGroupSettings": {
    "Type": "HLS_GROUP_SETTINGS",
    "HlsGroupSettings": {
        "ManifestDurationFormat": "INTEGER",
        "SegmentLength": 6,
        "TimedMetadataId3Period": 10,
        "CaptionLanguageSetting": "OMIT",
        "Destination": "s3://EXAMPLE-BUCKET/HLS/",
        "DestinationSettings": {
            "S3Settings": {
                "AccessControl": {
                    "CannedAcl": "PUBLIC_READ"
                }
            }
        },
        "TimedMetadataId3Frame": "PRIV",
        "CodecSpecification": "RFC_4281",
        "OutputSelection": "MANIFESTS_AND_SEGMENTS",
        "ProgramDateTimePeriod": 600,
        "MinSegmentLength": 0,
        "DirectoryStructure": "SINGLE_DIRECTORY",
        "ProgramDateTime": "EXCLUDE",
        "SegmentControl": "SEGMENTED_FILES",
        "ManifestCompression": "NONE",
        "ClientCache": "ENABLED",
        "StreamInfResolution": "INCLUDE"
    }
},
{
    "CustomName": "MP4",
    "Name": "File Group",
    "Outputs": [
        {
            "ContainerSettings": {
                "Container": "MP4",
                "Mp4Settings": {
                    "CsIgAtom": "INCLUDE",
                    "FreeSpaceBox": "EXCLUDE",
                    "MoovPlacement": "PROGRESSIVE_DOWNLOAD"
                }
            },
            "VideoDescription": {
                "Width": 1280,
                "ScalingBehavior": "DEFAULT",
                "Height": 720,
                "TimecodeInsertion": "DISABLED",
                "AntiAlias": "ENABLED",
                "Sharpness": 100,
                "CodecSettings": {
                    "Codec": "H_264",
                    "H264Settings": {
                        "InterlaceMode": "PROGRESSIVE",
                        "ParNumerator": 1,
                        "NumberReferenceFrames": 3,
                        "Syntax": "DEFAULT",
                        "Softness": 0,
                        "GopClosedCadence": 1,
                    }
                }
            }
        }
    ]
}

```

```
        "HrdBufferInitialFillPercentage": 90,
        "GopSize": 2,
        "Slices": 2,
        "GopBReference": "ENABLED",
        "HrdBufferSize": 10000000,
        "MaxBitrate": 5000000,
        "ParDenominator": 1,
        "EntropyEncoding": "CABAC",
        "RateControlMode": "QVBR",
        "CodecProfile": "HIGH",
        "MinIInterval": 0,
        "AdaptiveQuantization": "AUTO",
        "CodecLevel": "AUTO",
        "FieldEncoding": "PAFF",
        "SceneChangeDetect": "ENABLED",
        "QualityTuningLevel": "SINGLE_PASS_HQ",
        "UnregisteredSeiTimecode": "DISABLED",
        "GopSizeUnits": "SECONDS",
        "ParControl": "SPECIFIED",
        "NumberBFramesBetweenReferenceFrames": 3,
        "RepeatPps": "DISABLED",
        "DynamicSubGop": "ADAPTIVE"
    },
},
"AfdSignaling": "NONE",
"DropFrameTimecode": "ENABLED",
"RespondToAfd": "NONE",
"ColorMetadata": "INSERT"
},
"AudioDescriptions": [
{
    "AudioTypeControl": "FOLLOW_INPUT",
    "AudioSourceName": "Audio Selector 1",
    "CodecSettings": {
        "Codec": "AAC",
        "AacSettings": {
            "AudioDescriptionBroadcasterMix": "NORMAL",
            "Bitrate": 160000,
            "RateControlMode": "CBR",
            "CodecProfile": "LC",
            "CodingMode": "CODING_MODE_2_0",
            "RawFormat": "NONE",
            "SampleRate": 48000,
            "Specification": "MPEG4"
        }
    },
    "LanguageCodeControl": "FOLLOW_INPUT",
    "AudioType": 0
}
]
],
"OutputGroupSettings": {
    "Type": "FILE_GROUP_SETTINGS",
    "FileGroupSettings": {
        "Destination": "s3://EXAMPLE-BUCKET/MP4/",
        "DestinationSettings": {
            "S3Settings": {
                "AccessControl": {
                    "CannedAcl": "PUBLIC_READ"
                }
            }
        }
    }
}
},
```

```
{  
    "CustomName": "Thumbnails",  
    "Name": "File Group",  
    "Outputs": [  
        {  
            "ContainerSettings": {  
                "Container": "RAW"  
            },  
            "VideoDescription": {  
                "Width": 1280,  
                "ScalingBehavior": "DEFAULT",  
                "Height": 720,  
                "TimecodeInsertion": "DISABLED",  
                "AntiAlias": "ENABLED",  
                "Sharpness": 50,  
                "CodecSettings": {  
                    "Codec": "FRAME_CAPTURE",  
                    "FrameCaptureSettings": {  
                        "FramerateNumerator": 1,  
                        "FramerateDenominator": 5,  
                        "MaxCaptures": 500,  
                        "Quality": 80  
                    }  
                },  
                "AfdSignaling": "NONE",  
                "DropFrameTimecode": "ENABLED",  
                "RespondToAfd": "NONE",  
                "ColorMetadata": "INSERT"  
            }  
        }  
    ],  
    "OutputGroupSettings": {  
        "Type": "FILE_GROUP_SETTINGS",  
        "FileGroupSettings": {  
            "Destination": "s3://EXAMPLE-BUCKET/Thumbnails/",  
            "DestinationSettings": {  
                "S3Settings": {  
                    "AccessControl": {  
                        "CannedAcl": "PUBLIC_READ"  
                    }  
                }  
            }  
        }  
    }  
],  
    "AdAvailOffset": 0,  
    "Inputs": [  
        {  
            "AudioSelectors": {  
                "Audio Selector 1": {  
                    "Offset": 0,  
                    "DefaultSelection": "DEFAULT",  
                    "ProgramSelection": 1  
                }  
            },  
            "VideoSelector": {  
                "ColorSpace": "FOLLOW"  
            },  
            "FilterEnable": "AUTO",  
            "PsiControl": "USE_PSI",  
            "FilterStrength": 0,  
            "DeblockFilter": "DISABLED",  
            "DenoiseFilter": "DISABLED",  
            "TimecodeSource": "EMBEDDED",  
            "FileInput": "s3://EXAMPLE-INPUT-BUCKET/input.mp4"  
        }  
    ]  
}
```

```
    }
}
```

3. In the batch-transcode folder, create a file with a Lambda function. You can use the following Python example and name the file `convert.py`.

S3 Batch Operations sends specific task data to a Lambda function and requires result data back. For request and response examples for the Lambda function, information about response and result codes, and example Lambda functions for S3 Batch Operations, see [Invoke AWS Lambda function \(p. 907\)](#).

```
import json
import os
from urllib.parse import urlparse
import uuid
import boto3

"""
When you run an S3 Batch Operations job, your job
invokes this Lambda function. Specifically, the Lambda function is
invoked on each video object listed in the manifest that you specify
for the S3 Batch Operations job in Step 5 \(p. 81\).
"""

Input parameter "event": The S3 Batch Operations event as a request
for the Lambda function.

Input parameter "context": Context about the event.

Output: A result structure that Amazon S3 uses to interpret the result
of the operation. It is a job response returned back to S3 Batch Operations.
"""

def handler(event, context):

    invocation_schema_version = event['invocationSchemaVersion']
    invocation_id = event['invocationId']
    task_id = event['tasks'][0]['taskId']

    source_s3_key = event['tasks'][0]['s3Key']
    source_s3_bucket = event['tasks'][0]['s3BucketArn'].split(':::')[ -1]
    source_s3 = 's3://' + source_s3_bucket + '/' + source_s3_key

    result_list = []
    result_code = 'Succeeded'
    result_string = 'The input video object was converted successfully.'

    # The type of output group determines which media players can play
    # the files transcoded by MediaConvert.
    # For more information, see Creating outputs with AWS Elemental MediaConvert.
    output_group_type_dict = {
        'HLS_GROUP_SETTINGS': 'HlsGroupSettings',
        'FILE_GROUP_SETTINGS': 'FileGroupSettings',
        'CMAF_GROUP_SETTINGS': 'CmafGroupSettings',
        'DASH_ISO_GROUP_SETTINGS': 'DashIsoGroupSettings',
        'MS_SMOOTH_GROUP_SETTINGS': 'MsSmoothGroupSettings'
    }

    try:
        job_name = 'Default'
        with open('job.json') as file:
            job_settings = json.load(file)

        job_settings['Inputs'][0]['FileInput'] = source_s3
```

```

# The path of each output video is constructed based on the values of
# the attributes in each object of OutputGroups in the job.json file.
destination_s3 = 's3://{}//{}//{}' \
    .format(os.environ['DestinationBucket'],
            os.path.splitext(os.path.basename(source_s3_key))[0],
            os.path.splitext(os.path.basename(job_name))[0])

for output_group in job_settings['OutputGroups']:
    output_group_type = output_group['OutputGroupSettings']['Type']
    if output_group_type in output_group_type_dict.keys():
        output_group_type = output_group_type_dict[output_group_type]
        output_group['OutputGroupSettings'][output_group_type]['Destination'] =
    \'{0}{1}'.format(destination_s3,
                     urlparse(output_group['OutputGroupSettings']\
[output_group_type]['Destination']).path)
    else:
        raise ValueError("Exception: Unknown Output Group Type {}."
                         .format(output_group_type))

job_metadata_dict = {
    'assetID': str(uuid.uuid4()),
    'application': os.environ['Application'],
    'input': source_s3,
    'settings': job_name
}

region = os.environ['AWS_DEFAULT_REGION']
endpoints = boto3.client('mediaconvert', region_name=region) \
    .describe_endpoints()
client = boto3.client('mediaconvert', region_name=region,
                      endpoint_url=endpoints['Endpoints'][0]['Url'],
                      verify=False)

try:
    client.create_job(Role=os.environ['MediaConvertRole'],
                       UserMetadata=job_metadata_dict,
                       Settings=job_settings)
# You can customize error handling based on different error codes that
# MediaConvert can return.
# For more information, see MediaConvert error codes.
# When the result_code is TemporaryFailure, S3 Batch Operations retries
# the task before the job is completed. If this is the final retry,
# the error message is included in the final report.
except Exception as error:
    result_code = 'TemporaryFailure'
    raise

except Exception as error:
    if result_code != 'TemporaryFailure':
        result_code = 'PermanentFailure'
    result_string = str(error)

finally:
    result_list.append({
        'taskId': task_id,
        'resultCode': result_code,
        'resultString': result_string,
    })

return {
    'invocationSchemaVersion': invocation_schema_version,
    'treatMissingKeyAs': 'PermanentFailure',
    'invocationId': invocation_id,
    'results': result_list
}

```

```
}
```

4. To create a deployment package with `convert.py` and `job.json` as a `.zip` file named `lambda.zip`, in your local terminal, open the `batch-transcode` folder that you created earlier, and run the following command.

For **macOS users**, run the following command:

```
zip -r lambda.zip convert.py job.json
```

For **Windows users**, run the following commands:

```
powershell Compress-Archive convert.py lambda.zip
```

```
powershell Compress-Archive -update job.json lambda.zip
```

Create a Lambda function with an execution role (console)

1. Open the AWS Lambda console at <https://console.aws.amazon.com/lambda/>.
2. In the left navigation pane, choose **Functions**.
3. Choose **Create function**.
4. Choose **Author from scratch**.
5. Under **Basic information**, do the following:
 - a. For **Function name**, enter `tutorial-lambda-convert`.
 - b. For **Runtime**, choose **Python 3.8** or a later version of Python.
6. Choose **Change default execution role**, and under **Execution role**, choose **Use an existing role**.
7. Under **Existing role**, choose the name of the IAM role that you created for your Lambda function in [Step 3 \(p. 68\)](#) (for example, `tutorial-lambda-transcode-role`).
8. For the remaining settings, keep the defaults.
9. Choose **Create function**.

Deploy your Lambda function with .zip file archives and configure the Lambda function (console)

1. In the **Code Source** section of the page for the Lambda function that you created (for example, `tutorial-lambda-convert`), choose **Upload from** and then **.zip file**.
2. Choose **Upload** to select your local `.zip` file.
3. Choose the `lambda.zip` file that you created earlier, and choose **Open**.
4. Choose **Save**.
5. In the **Runtime settings** section, choose **Edit**.
6. To tell the Lambda runtime which handler method in your Lambda function code to invoke, enter `convert.handler` in the **Handler** field.

When you configure a function in Python, the value of the handler setting is the file name and the name of the handler module, separated by a dot (.). For example, `convert.handler` calls the `handler` method defined in the `convert.py` file.

7. Choose **Save**.

8. On your Lambda function page, choose the **Configuration** tab. In the left navigation pane on the **Configuration** tab, choose **Environment variables**, and then choose **Edit**.
9. Choose **Add environment variable**. Then, enter the specified **Key** and **Value** for each of the following environment variables:

- **Key: DestinationBucket Value: tutorial-bucket-2**

This value is the S3 bucket for output media files that you created in [Step 1 \(p. 66\)](#).

- **Key: MediaConvertRole Value: arn:aws:iam::111122223333:role/tutorial-mediaconvert-role**

This value is the ARN of the IAM role for MediaConvert that you created in [Step 2 \(p. 67\)](#). Make sure to replace this ARN with the actual ARN of your IAM role.

- **Key: Application Value: Batch-Transcoding**

This value is the name of the application.

10. Choose **Save**.

11. (Optional) On the **Configuration** tab, in the **General configuration** section of the left navigation pane, choose **Edit**. In the **Timeout** field, enter **2 min 0 sec**. Then, choose **Save**.

Timeout is the amount of time that Lambda allows a function to run for an invocation before stopping it. The default is 3 seconds. Pricing is based on the amount of memory configured and the amount of time that your code runs. For more information, see [AWS Lambda pricing](#).

Step 5: Configure Amazon S3 Inventory for your S3 source bucket

After setting up the transcoding Lambda function, create an S3 Batch Operations job to transcode a set of videos. First, you need a list of input video objects that you want S3 Batch Operations to run the specified transcoding action on. To get a list of input video objects, you can generate an S3 Inventory report for your S3 source bucket (for example, **tutorial-bucket-1**).

Substeps

- [Create and configure a bucket for S3 Inventory reports for input videos \(p. 81\)](#)
- [Configure Amazon S3 Inventory for your S3 video source bucket \(p. 82\)](#)
- [Check the inventory report for your S3 video source bucket \(p. 83\)](#)

Create and configure a bucket for S3 Inventory reports for input videos

To store an S3 Inventory report that lists the objects of the S3 source bucket, create an S3 Inventory destination bucket, and then configure a bucket policy for the bucket to write inventory files to the S3 source bucket.

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. Choose **Create bucket**.
4. For **Bucket name**, enter a name for your bucket (for example, **tutorial-bucket-3**).
5. For **AWS Region**, choose the AWS Region where you want the bucket to reside.

The inventory destination bucket must be in the same AWS Region as the source bucket where you are setting up S3 Inventory. The inventory destination bucket can be in a different AWS account.

6. In **Block Public Access settings for this bucket**, keep the default settings (**Block all public access** is enabled).
7. For the remaining settings, keep the defaults.
8. Choose **Create bucket**.
9. In the **Buckets** list, choose the name of the bucket that you just created (for example, `tutorial-bucket-3`).
10. To grant Amazon S3 permission to write data for the inventory reports to the S3 Inventory destination bucket, choose the **Permissions** tab.
11. Scroll down to the **Bucket policy** section, and choose **Edit**. The **Bucket policy** page opens.
12. To grant permissions for S3 Inventory, in the **Policy** field, paste the following bucket policy.

Replace the three example values with the following values:

- The name of the bucket that you created to store the inventory reports (for example, `tutorial-bucket-3`).
- The name of the source bucket that stores the input videos (for example, `tutorial-bucket-1`).
- The AWS account ID that you used to create the S3 video source bucket (for example, `111122223333`).

```
{  
    "Version": "2012-10-17",  
    "Statement": [  
        {  
            "Sid": "InventoryAndAnalyticsExamplePolicy",  
            "Effect": "Allow",  
            "Principal": {"Service": "s3.amazonaws.com"},  
            "Action": "s3:PutObject",  
            "Resource": ["arn:aws:s3:::tutorial-bucket-3/*"],  
            "Condition": {  
                "ArnLike": {  
                    "aws:SourceArn": "arn:aws:s3:::tutorial-bucket-1"  
                },  
                "StringEquals": {  
                    "aws:SourceAccount": "111122223333",  
                    "s3:x-amz-acl": "bucket-owner-full-control"  
                }  
            }  
        }  
    ]  
}
```

13. Choose **Save changes**.

Configure Amazon S3 Inventory for your S3 video source bucket

To generate a flat file list of video objects and metadata, you must configure S3 Inventory for your S3 video source bucket. These scheduled inventory reports can include all the objects in the bucket or objects grouped by a shared prefix. In this tutorial, the S3 Inventory report includes all the video objects in your S3 source bucket.

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. To configure an S3 Inventory report of the input videos in your S3 source bucket, in the **Buckets** list, choose the name of the S3 source bucket (for example, **tutorial-bucket-1**).
4. Choose the **Management** tab.
5. Scroll down to the **Inventory configurations** section, and choose **Create inventory configuration**.
6. For **Inventory configuration name**, enter a name (for example, **tutorial-inventory-config**).
7. Under **Inventory scope**, choose **Current version only** for **Object versions** and keep the other **Inventory scope** settings set to the defaults for this tutorial.
8. In the **Report details** section, for **Destination bucket**, choose **This account**.
9. For **Destination**, choose **Browse S3**, and choose the destination bucket that you created earlier to save the inventory reports to (for example, **tutorial-bucket-3**). Then choose **Choose path**.

The inventory destination bucket must be in the same AWS Region as the source bucket where you are setting up S3 Inventory. The inventory destination bucket can be in a different AWS account.

Under the **Destination** bucket field, the **Destination bucket permission** is added to the inventory destination bucket policy, allowing Amazon S3 to place data in the inventory destination bucket. For more information, see [Creating a destination bucket policy \(p. 742\)](#).

10. For **Frequency**, choose **Daily**.
11. For **Output format**, choose **CSV**.
12. For **Status**, choose **Enable**.
13. In the **Server-side encryption** section, choose **Disable** for this tutorial.

For more information, see [Configuring inventory using the S3 console \(p. 744\)](#) and [Granting Amazon S3 permission to use your AWS KMS key for encryption \(p. 743\)](#).

14. In the **Additional fields - optional** section, select **Size**, **Last modified**, and **Storage class**.
15. Choose **Create**.

For more information, see [Configuring inventory using the S3 console \(p. 744\)](#).

Check the inventory report for your S3 video source bucket

When an inventory report is published, the manifest files are sent to the S3 Inventory destination bucket.

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. In the **Buckets** list, choose the name of the video source bucket (for example, **tutorial-bucket-1**).
4. Choose **Management**.
5. To see if your S3 Inventory report is ready so that you can create an S3 Batch Operations job in [Step 7 \(p. 86\)](#), under **Inventory configurations**, check whether the **Create job from manifest** button is enabled.

Note

It can take up to 48 hours to deliver the first inventory report. If the **Create job from manifest** button is disabled, the first inventory report has not been delivered. Wait until the first inventory report is delivered and the **Create job from manifest** button is enabled before you create an S3 Batch Operations job in [Step 7 \(p. 86\)](#).

6. To check an S3 Inventory report (`manifest.json`), in the **Destination** column, choose the name of the inventory destination bucket that you created earlier for storing inventory reports (for example, **tutorial-bucket-3**).

7. On the **Objects** tab, choose the existing folder with the name of your S3 source bucket (for example, **tutorial-bucket-1**). Then choose the name that you entered in **Inventory configuration name** when you created the inventory configuration earlier (for example, **tutorial-inventory-config**).
You can see a list of folders with the generation dates of the reports as their names.
8. To check the daily S3 Inventory report for a particular date, choose the folder with the corresponding generation date name, and then choose **manifest.json**.
9. To check the details of the inventory report on a specific date, on the **manifest.json** page, choose **Download** or **Open**.

Step 6: Create an IAM role for S3 Batch Operations

To use S3 Batch Operations to do batch-transcoding, you must first create an IAM role to give Amazon S3 permissions to perform S3 Batch Operations.

Substeps

- [Create an IAM policy for S3 Batch Operations \(p. 84\)](#)
- [Create an S3 Batch Operations IAM role and attach permissions policies \(p. 85\)](#)

Create an IAM policy for S3 Batch Operations

You must create an IAM policy that gives S3 Batch Operations permission to read the input manifest, invoke the Lambda function, and write the S3 Batch Operations job completion report.

1. Sign in to the AWS Management Console and open the IAM console at <https://console.aws.amazon.com/iam/>.
2. In the left navigation pane, choose **Policies**.
3. Choose **Create policy**.
4. Choose the **JSON** tab.
5. In the **JSON** text field, paste the following JSON policy.

In the JSON policy, replace the four example values with the following values:

- The name of the source bucket that stores your input videos (for example, **tutorial-bucket-1**).
- The name of the inventory destination bucket that you created in [Step 5 \(p. 81\)](#) to store **manifest.json** files (for example, **tutorial-bucket-3**).
- The name of the bucket that you created in [Step 1 \(p. 66\)](#) to store output media files (for example, **tutorial-bucket-2**). In this tutorial, we put job completion reports in the destination bucket for output media files.
- The role ARN of the Lambda function that you created in [Step 4 \(p. 69\)](#). To find and copy the role ARN of the Lambda function, do the following:
 - In a new browser tab, open the **Functions** page on the Lambda console at <https://console.aws.amazon.com/lambda/home#/functions>.
 - In **Functions** list, choose the name of the Lambda function that you created in [Step 4 \(p. 69\)](#) (for example, **tutorial-lambda-convert**).
 - Choose **Copy ARN**.

```
{  
    "Version": "2012-10-17",  
    "Statement": [
```

```
{  
    "Sid": "S3Get",  
    "Effect": "Allow",  
    "Action": [  
        "s3:GetObject",  
        "s3:GetObjectVersion"  
    ],  
    "Resource": [  
        "arn:aws:s3:::tutorial-bucket-1/*",  
        "arn:aws:s3:::tutorial-bucket-3/*"  
    ]  
},  
{  
    "Sid": "S3PutJobCompletionReport",  
    "Effect": "Allow",  
    "Action": "s3:PutObject",  
    "Resource": "arn:aws:s3:::tutorial-bucket-2/*"  
},  
{  
    "Sid": "S3BatchOperationsInvokeLambda",  
    "Effect": "Allow",  
    "Action": [  
        "lambda:InvokeFunction"  
    ],  
    "Resource": [  
        "arn:aws:lambda:us-west-2:111122223333:function:tutorial-lambda-  
convert"  
    ]  
}  
]
```

6. Choose **Next: Tags**.
7. Choose **Next: Review**.
8. In the **Name** field, enter **tutorial-s3batch-policy**.
9. Choose **Create policy**.

Create an S3 Batch Operations IAM role and attach permissions policies

1. Sign in to the AWS Management Console and open the IAM console at <https://console.aws.amazon.com/iam/>.
2. In the left navigation pane, choose **Roles**, and then choose **Create role**.
3. Choose the **AWS service** role type, and then choose the **S3** service.
4. Under **Select your use case**, choose **S3 Batch Operations**.
5. Choose **Next: Permissions**.
6. Under **Attach permissions policies**, enter the name of the IAM policy that you created earlier (for example, **tutorial-s3batch-policy**) in the search box to filter the list of policies. Select the check box next to the name of the policy (for example, **tutorial-s3batch-policy**).
7. Choose **Next: Tags**.
8. Choose **Next: Review**.
9. For **Role name**, enter **tutorial-s3batch-role**.
10. Choose **Create role**.

After you create the IAM role for S3 Batch Operations, the following trust policy is automatically attached to the role. This trust policy allows the S3 Batch Operations service principal to assume the IAM role.

```
{  
    "Version": "2012-10-17",  
    "Statement": [  
        {  
            "Effect": "Allow",  
            "Principal": {  
                "Service": "batchoperations.s3.amazonaws.com"  
            },  
            "Action": "sts:AssumeRole"  
        }  
    ]  
}
```

Step 7: Create and run an S3 Batch Operations job

To create an S3 Batch Operations job to process the input videos in your S3 source bucket, you must specify parameters for this particular job.

Note

Before you start creating an S3 Batch Operations job, make sure that the **Create job from manifest** button is enabled. For more information, see [Check the inventory report for your S3 video source bucket \(p. 83\)](#). If the **Create job from manifest** button is disabled, the first inventory report has not been delivered and you must wait until the button is enabled. After you configure Amazon S3 Inventory for your S3 source bucket in [Step 5 \(p. 81\)](#), it can take up to 48 hours to deliver the first inventory report.

Substeps

- [Create an S3 Batch Operations job \(p. 86\)](#)
- [Run the S3 Batch Operations job to invoke your Lambda function \(p. 87\)](#)
- [\(Optional\) Check your completion report \(p. 88\)](#)
- [\(Optional\) Monitor each Lambda invocation in the Lambda console \(p. 88\)](#)
- [\(Optional\) Monitor each MediaConvert video-transcoding job in the MediaConvert console \(p. 89\)](#)

Create an S3 Batch Operations job

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 **Batch Operations**.
3. Choose **Create job**.
4. For **AWS Region**, choose the Region where you want to create your job.

In this tutorial, to use the S3 Batch Operations job to invoke a Lambda function, you must create the job in the same Region as the S3 video source bucket where the objects referenced in the manifest are located.

5. In the **Manifest** section, do the following:
 - a. For **Manifest format**, choose **S3 Inventory report (manifest.json)**.
 - b. For **Manifest object**, choose **Browse S3** to find the bucket that you created in [Step 5 \(p. 81\)](#) for storing inventory reports (for example, `tutorial-bucket-3`). On the **Manifest object** page, navigate through the object names until you find a `manifest.json` file for a specific date. This file lists the information about all the videos that you want to batch-transcode. When you've found the `manifest.json` file that you want to use, choose the option button next to it. Then choose **Choose path**.

- c. (Optional) For **Manifest object version ID - optional**, enter the version ID for the manifest object if you want to use a version other than the most recent.
6. Choose **Next**.
7. To use the Lambda function to transcode all the objects listed in the selected `manifest.json` file, under **Operation type**, choose **Invoke AWS Lambda function**.
8. In the **Invoke Lambda function** section, do the following:
 - a. Choose **Choose from functions in your account**.
 - b. For **Lambda function**, choose the Lambda function that you created in [Step 4 \(p. 69\)](#) (for example, `tutorial-lambda-convert`).
 - c. For **Lambda function version**, keep the default value `$LATEST`.
9. Choose **Next**. The **Configure additional options** page opens.
10. In the **Additional options** section, keep the default settings.

For more information about these options, see [Batch Operations job request elements \(p. 889\)](#).
11. In the **Completion report** section, for **Path to completion report destination**, choose **Browse S3**. Find the bucket that you created for output media files in [Step 1 \(p. 66\)](#) (for example, `tutorial-bucket-2`). Choose the option button next to that bucket's name. Then choose **Choose path**.

For the remaining **Completion report** settings, keep the defaults. For more information about completion report settings, see [Batch Operations job request elements \(p. 889\)](#). A completion report maintains a record of the job's details and the operations performed.
12. In the **Permissions** section, choose **Choose from existing IAM roles**. For **IAM role**, choose the IAM role for your S3 Batch Operations job that you created in [Step 6 \(p. 84\)](#) (for example, `tutorial-s3batch-role`).
13. Choose **Next**.
14. On the **Review** page, review the settings. Then choose **Create job**.

After S3 finishes reading your S3 Batch Operations job's manifest, it sets the **Status** of the job to **Awaiting your confirmation to run**. To see updates to the job's status, refresh the page. You can't run your job until its status is **Awaiting your confirmation to run**.

Run the S3 Batch Operations job to invoke your Lambda function

Run your Batch Operations job to invoke your Lambda function for video transcoding. If your job fails, you can check your completion report to identify the cause.

To run the S3 Batch Operations job

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 **Batch Operations**.
3. In the **Jobs** list, choose the **Job ID** of the job on the first row, which is the S3 Batch Operations job that you created earlier.
4. Choose **Run job**.
5. Review your job parameters again, and confirm that the value for **Total objects listed in manifest** is the same as the number of objects in the manifest. Then choose **Run job**.

Your S3 Batch Operations job page opens.

6. After the job starts running, on your job page, under **Status**, check the progress of your S3 Batch Operations job, such as **Status**, **% Complete**, **Total succeeded (rate)**, **Total failed (rate)**, **Date terminated**, and **Reason for termination**.

When the S3 Batch Operations job completes, view the data on your job page to confirm that the job finished as expected.

If more than 50 percent of an S3 Batch Operations job's object operations fail after more than 1,000 operations have been attempted, the job automatically fails. To check your completion report to identify the cause of the failures, use the following optional procedure.

(Optional) Check your completion report

You can use your completion report to determine which objects failed and the cause of the failures.

To check your completion report for details about failed objects

1. On the page of your S3 Batch Operations job, scroll down to the **Completion report** section, and choose the link under **Completion report destination**.

The S3 output destination bucket's page opens.
2. On the **Objects** tab, choose the folder that has a name ending with the job ID of the S3 Batch Operations job that you created earlier.
3. Choose **results/**.
4. Select the check box next to the **.csv** file.
5. To view the job report, choose **Open** or **Download**.

(Optional) Monitor each Lambda invocation in the Lambda console

After the S3 Batch Operations job starts running, the job invokes the Lambda function for each input video object. S3 writes logs of each Lambda invocation to CloudWatch Logs. You can use the Lambda console's monitoring dashboard to monitor your Lambda function.

1. Open the AWS Lambda console at <https://console.aws.amazon.com/lambda/>.
2. In the left navigation pane, choose **Functions**.
3. In the **Functions** list, choose the name of the Lambda function that you created in [Step 4 \(p. 69\)](#) (for example, **tutorial-lambda-convert**).
4. Choose the **Monitor** tab.
5. Under **Metrics**, see the runtime metrics for your Lambda function.
6. Under **Logs**, view log data for each Lambda invocation through CloudWatch Logs Insights.

Note

When you use S3 Batch Operations with a Lambda function, the Lambda function is invoked on each object. If your S3 Batch Operations job is large, it can invoke multiple Lambda functions at the same time, causing a spike in Lambda concurrency.

Each AWS account has a Lambda concurrency quota per Region. For more information, see [AWS Lambda Function Scaling](#) in the *AWS Lambda Developer Guide*. A best practice for using Lambda functions with S3 Batch Operations is to set a concurrency limit on the Lambda function itself. Setting a concurrency limit keeps your job from consuming most of your Lambda concurrency and potentially throttling other functions in your account. For more

information, see [Managing Lambda reserved concurrency](#) in the *AWS Lambda Developer Guide*.

(Optional) Monitor each MediaConvert video-transcoding job in the MediaConvert console

A MediaConvert job does the work of transcoding a media file. When your S3 Batch Operations job invokes your Lambda function for each video, each Lambda function invocation creates a MediaConvert transcoding job for each input video.

1. Sign in to the AWS Management Console and open the MediaConvert console at <https://console.aws.amazon.com/mediaconvert/>.
2. If the MediaConvert introductory page appears, choose **Get started**.
3. From the list of **Jobs**, view each row to monitor the transcoding task for each input video.
4. Identify the row of a job that you want to check, and choose the **Job ID** link to open the job details page.
5. On the **Job summary** page, under **Outputs**, choose the link for the HLS, MP4, or Thumbnails output, depending on what is supported by your browser, to go to the S3 destination bucket for the output media files.
6. In the corresponding folder (HLS, MP4, or Thumbnails) of your S3 output destination bucket, choose the name of the output media file object.

The object's detail page opens.
7. On the object's detail page, under **Object overview**, choose the link under **Object URL** to watch the transcoded output media file.

Step 8: Check the output media files from your S3 destination bucket

To check the output media files from your S3 destination bucket

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. In the **Buckets** list, choose the name of the S3 destination bucket for output media files that you created in [Step 1 \(p. 66\)](#) (for example, **tutorial-bucket-2**).
4. On the **Objects** tab, each input video has a folder that has the name of the input video. Each folder contains the transcoded output media files for an input video.

To check the output media files for an input video, do the following:

- a. Choose the folder with the name of the input video that you want to check.
- b. Choose the **Default/** folder.
- c. Choose the folder for a transcoded format (HLS, MP4, or thumbnails in this tutorial).
- d. Choose the name of the output media file.
- e. To watch the transcoded file, on the object's details page, choose the link under **Object URL**.

Output media files in the HLS format are split into short segments. To play these videos, embed the object URL of the .m3u8 file in a compatible player.

Step 9: Clean up

If you transcoded videos using S3 Batch Operations, Lambda, and MediaConvert only as a learning exercise, delete the AWS resources that you allocated so that you no longer accrue charges.

Substeps

- [Delete the S3 Inventory configuration for your S3 source bucket \(p. 90\)](#)
- [Delete the Lambda function \(p. 90\)](#)
- [Delete the CloudWatch log group \(p. 90\)](#)
- [Delete the IAM roles together with the inline policies for the IAM roles \(p. 91\)](#)
- [Delete the customer-managed IAM policy \(p. 91\)](#)
- [Empty the S3 buckets \(p. 91\)](#)
- [Delete the S3 buckets \(p. 91\)](#)

Delete the S3 Inventory configuration for your S3 source bucket

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. In the **Buckets** list, choose the name of your source bucket (for example, **tutorial-bucket-1**).
4. Choose the **Management** tab.
5. In the **Inventory configurations** section, choose the option button next to the inventory configuration that you created in [Step 5 \(p. 81\)](#) (for example, **tutorial-inventory-config**).
6. Choose **Delete**, and then choose **Confirm**.

Delete the Lambda function

1. Open the AWS Lambda console at <https://console.aws.amazon.com/lambda/>.
2. In the left navigation pane, choose **Functions**.
3. Select the check box next to the function that you created in [Step 4 \(p. 69\)](#) (for example, **tutorial-lambda-convert**).
4. Choose **Actions**, and then choose **Delete**.
5. In the **Delete function** dialog box, choose **Delete**.

Delete the CloudWatch log group

1. Open the CloudWatch console at <https://console.aws.amazon.com/cloudwatch/>.
2. In the left navigation pane, choose **Logs**, and then choose **Log groups**.
3. Select the check box next to the log group that has a name ending with the Lambda function that you created in [Step 4 \(p. 69\)](#) (for example, **tutorial-lambda-convert**).
4. Choose **Actions**, and then choose **Delete log group(s)**.
5. In the **Delete log group(s)** dialog box, choose **Delete**.

Delete the IAM roles together with the inline policies for the IAM roles

To delete the IAM roles that you created in [Step 2 \(p. 67\)](#), [Step 3 \(p. 68\)](#), and [Step 6 \(p. 84\)](#), do the following:

1. Sign in to the AWS Management Console and open the IAM console at <https://console.aws.amazon.com/iam/>.
2. In the left navigation pane, choose **Roles**, and then select the check boxes next to the role names that you want to delete.
3. At the top of the page, choose **Delete**.
4. In the confirmation dialog box, enter the required response in the text input field based on the prompt, and choose **Delete**.

Delete the customer-managed IAM policy

To delete the customer-managed IAM policy that you created in [Step 6 \(p. 84\)](#), do the following:

1. Sign in to the AWS Management Console and open the IAM console at <https://console.aws.amazon.com/iam/>.
2. In the left navigation pane, choose **Policies**.
3. Choose the option button next to the policy that you created in [Step 6 \(p. 84\)](#) (for example, **tutorial-s3batch-policy**). You can use the search box to filter the list of policies.
4. Choose **Actions**, and then choose **Delete**.
5. Confirm that you want to delete this policy by entering its name in the text field, and then choose **Delete**.

Empty the S3 buckets

To empty the S3 buckets that you created in [Prerequisites \(p. 65\)](#), [Step 1 \(p. 66\)](#), and [Step 5 \(p. 81\)](#), do the following:

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. In the **Buckets** list, choose the option button next to the name of the bucket that you want to empty, and then choose **Empty**.
4. On the **Empty bucket** page, confirm that you want to empty the bucket by entering **permanently delete** in the text field, and then choose **Empty**.

Delete the S3 buckets

To delete the S3 buckets that you created in [Prerequisites \(p. 65\)](#), [Step 1 \(p. 66\)](#), and [Step 5 \(p. 81\)](#), do the following:

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. In the **Buckets** list, choose the option button next to the name of the bucket that you want to delete.

4. Choose **Delete**.
5. On the **Delete bucket** page, confirm that you want to delete the bucket by entering the bucket name in the text field, and then choose **Delete bucket**.

Next steps

After completing this tutorial, you can further explore other relevant use cases:

- You can use Amazon CloudFront to stream the transcoded media files to viewers across the globe. For more information, see [Tutorial: Hosting on-demand streaming video with Amazon S3, Amazon CloudFront, and Amazon Route 53 \(p. 51\)](#).
- You can transcode videos at the moment when you upload them to the S3 source bucket. To do so, you can configure an Amazon S3 event trigger that automatically invokes the Lambda function to transcode new objects in S3 with MediaConvert. For more information, see [Tutorial: Using an Amazon S3 trigger to invoke a Lambda function in the AWS Lambda Developer Guide](#).

Tutorial: Configuring a static website on Amazon S3

You can configure an Amazon S3 bucket to function like a website. This example walks you through the steps of hosting a website on Amazon S3.

Topics

- [Step 1: Create a bucket \(p. 92\)](#)
- [Step 2: Enable static website hosting \(p. 93\)](#)
- [Step 3: Edit Block Public Access settings \(p. 93\)](#)
- [Step 4: Add a bucket policy that makes your bucket content publicly available \(p. 94\)](#)
- [Step 5: Configure an index document \(p. 95\)](#)
- [Step 6: Configure an error document \(p. 96\)](#)
- [Step 7: Test your website endpoint \(p. 97\)](#)
- [Step 8: Clean up \(p. 97\)](#)

Step 1: Create a bucket

The following instructions provide an overview of how to create your buckets for website hosting. For detailed, step-by-step instructions on creating a bucket, see [Creating a bucket \(p. 119\)](#).

To create a bucket

1. Sign in to the AWS Management Console and open the Amazon S3 console at <https://console.aws.amazon.com/s3/>.
2. Choose **Create bucket**.
3. Enter the **Bucket name** (for example, `example.com`).
4. Choose the Region where you want to create the bucket.

Choose a Region that is geographically close to you to minimize latency and costs, or to address regulatory requirements. The Region that you choose determines your Amazon S3 website endpoint. For more information, see [Website endpoints \(p. 1116\)](#).

5. To accept the default settings and create the bucket, choose **Create**.

Step 2: Enable static website hosting

After you create a bucket, you can enable static website hosting for your bucket. You can create a new bucket or use an existing bucket.

To enable static website hosting

1. Sign in to the AWS Management Console and open the Amazon S3 console at <https://console.aws.amazon.com/s3/>.
2. In the **Buckets** list, choose the name of the bucket that you want to enable static website hosting for.
3. Choose **Properties**.
4. Under **Static website hosting**, choose **Edit**.
5. Choose **Use this bucket to host a website**.
6. Under **Static website hosting**, choose **Enable**.
7. In **Index document**, enter the file name of the index document, typically `index.html`.

The index document name is case sensitive and must exactly match the file name of the HTML index document that you plan to upload to your S3 bucket. When you configure a bucket for website hosting, you must specify an index document. Amazon S3 returns this index document when requests are made to the root domain or any of the subfolders. For more information, see [Configuring an index document \(p. 1122\)](#).

8. To provide your own custom error document for 4XX class errors, in **Error document**, enter the custom error document file name.

The error document name is case sensitive and must exactly match the file name of the HTML error document that you plan to upload to your S3 bucket. If you don't specify a custom error document and an error occurs, Amazon S3 returns a default HTML error document. For more information, see [Configuring a custom error document \(p. 1124\)](#).

9. (Optional) If you want to specify advanced redirection rules, in **Redirection rules**, enter JSON to describe the rules.

For example, you can conditionally route requests according to specific object key names or prefixes in the request. For more information, see [Configure redirection rules to use advanced conditional redirects \(p. 1131\)](#).

10. Choose **Save changes**.

Amazon S3 enables static website hosting for your bucket. At the bottom of the page, under **Static website hosting**, you see the website endpoint for your bucket.

11. Under **Static website hosting**, note the **Endpoint**.

The **Endpoint** is the Amazon S3 website endpoint for your bucket. After you finish configuring your bucket as a static website, you can use this endpoint to test your website.

Step 3: Edit Block Public Access settings

By default, Amazon S3 blocks public access to your account and buckets. If you want to use a bucket to host a static website, you can use these steps to edit your block public access settings.

Warning

Before you complete this step, review [Blocking public access to your Amazon S3 storage \(p. 584\)](#) to ensure that you understand and accept the risks involved with allowing public access. When you turn off block public access settings to make your bucket public, anyone

on the internet can access your bucket. We recommend that you block all public access to your buckets.

1. Open the Amazon S3 console at <https://console.aws.amazon.com/s3/>.
2. Choose the name of the bucket that you have configured as a static website.
3. Choose **Permissions**.
4. Under **Block public access (bucket settings)**, choose **Edit**.
5. Clear **Block all public access**, and choose **Save changes**.

Warning

Before you complete this step, review [Blocking public access to your Amazon S3 storage \(p. 584\)](#) to ensure you understand and accept the risks involved with allowing public access. When you turn off block public access settings to make your bucket public, anyone on the internet can access your bucket. We recommend that you block all public access to your buckets.

Block public access (bucket settings)

Public access is granted to buckets and objects through access control lists (ACLs), bucket policies, access point policies, or all. In order to ensure that public access to all your S3 buckets and objects is blocked, turn on Block all public access. These settings apply only to this bucket and its access points. AWS recommends that you turn on Block all public access, but before applying any of these settings, ensure that your applications will work correctly without public access. If you require some level of public access to your buckets or objects within, you can customize the individual settings below to suit your specific storage use cases. [Learn more](#)



Account settings for Block Public Access are currently turned on

Account settings for Block Public Access that are enabled apply even if they are disabled for this bucket.

Block all public access

Turning this setting on is the same as turning on all four settings below. Each of the following settings are independent of one another.

Block public access to buckets and objects granted through new access control lists (ACLs)

S3 will block public access permissions applied to newly added buckets or objects, and prevent the creation of new public access ACLs for existing buckets and objects. This setting doesn't change any existing permissions that allow public access to S3 resources using ACLs.

Block public access to buckets and objects granted through any access control lists (ACLs)

S3 will ignore all ACLs that grant public access to buckets and objects.

Block public access to buckets and objects granted through new public bucket or access point policies

S3 will block new bucket and access point policies that grant public access to buckets and objects. This setting doesn't change any existing policies that allow public access to S3 resources.

Block public and cross-account access to buckets and objects through any public bucket or access point policies

S3 will ignore public and cross-account access for buckets or access points with policies that grant public access to buckets and objects.

Amazon S3 turns off Block Public Access settings for your bucket. To create a public, static website, you might also have to [edit the Block Public Access settings](#) for your account before adding a bucket policy. If account settings for Block Public Access are currently turned on, you see a note under **Block public access (bucket settings)**.

Step 4: Add a bucket policy that makes your bucket content publicly available

After you edit S3 Block Public Access settings, you can add a bucket policy to grant public read access to your bucket. When you grant public read access, anyone on the internet can access your bucket.

Important

The following policy is an example only and allows full access to the contents of your bucket. Before you proceed with this step, review [How can I secure the files in my Amazon S3 bucket?](#) to ensure that you understand the best practices for securing the files in your S3 bucket and risks involved in granting public access.

1. Under **Buckets**, choose the name of your bucket.
2. Choose **Permissions**.
3. Under **Bucket Policy**, choose **Edit**.
4. To grant public read access for your website, copy the following bucket policy, and paste it in the **Bucket policy editor**.

```
{  
    "Version": "2012-10-17",  
    "Statement": [  
        {  
            "Sid": "PublicReadGetObject",  
            "Effect": "Allow",  
            "Principal": "*",  
            "Action": [  
                "s3:GetObject"  
            ],  
            "Resource": [  
                "arn:aws:s3:::Bucket-Name/*"  
            ]  
        }  
    ]  
}
```

5. Update the **Resource** to your bucket name.

In the preceding example bucket policy, **Bucket-Name** is a placeholder for the bucket name. To use this bucket policy with your own bucket, you must update this name to match your bucket name.

6. Choose **Save changes**.

A message appears indicating that the bucket policy has been successfully added.

If you see an error that says `Policy has invalid resource`, confirm that the bucket name in the bucket policy matches your bucket name. For information about adding a bucket policy, see [How do I add an S3 bucket policy?](#)

If you get an error message and cannot save the bucket policy, check your account and bucket Block Public Access settings to confirm that you allow public access to the bucket.

Step 5: Configure an index document

When you enable static website hosting for your bucket, you enter the name of the index document (for example, `index.html`). After you enable static website hosting for the bucket, you upload an HTML file with this index document name to your bucket.

To configure the index document

1. Create an `index.html` file.

If you don't have an `index.html` file, you can use the following HTML to create one:

```
<html xmlns="http://www.w3.org/1999/xhtml" >  
<head>
```

```
<title>My Website Home Page</title>
</head>
<body>
  <h1>Welcome to my website</h1>
  <p>Now hosted on Amazon S3!</p>
</body>
</html>
```

2. Save the index file locally.

The index document file name must exactly match the index document name that you enter in the **Static website hosting** dialog box. The index document name is case sensitive. For example, if you enter `index.html` for the **Index document** name in the **Static website hosting** dialog box, your index document file name must also be `index.html` and not `Index.html`.

3. Sign in to the AWS Management Console and open the Amazon S3 console at <https://console.aws.amazon.com/s3/>.
4. In the **Buckets** list, choose the name of the bucket that you want to use to host a static website.
5. Enable static website hosting for your bucket, and enter the exact name of your index document (for example, `index.html`). For more information, see [Enabling website hosting \(p. 1118\)](#).

After enabling static website hosting, proceed to step 6.

6. To upload the index document to your bucket, do one of the following:
 - Drag and drop the index file into the console bucket listing.
 - Choose **Upload**, and follow the prompts to choose and upload the index file.

For step-by-step instructions, see [Uploading objects \(p. 158\)](#).

7. (Optional) Upload other website content to your bucket.

Step 6: Configure an error document

When you enable static website hosting for your bucket, you enter the name of the error document (for example, `404.html`). After you enable static website hosting for the bucket, you upload an HTML file with this error document name to your bucket.

To configure an error document

1. Create an error document, for example `404.html`.
2. Save the error document file locally.

The error document name is case sensitive and must exactly match the name that you enter when you enable static website hosting. For example, if you enter `404.html` for the **Error document** name in the **Static website hosting** dialog box, your error document file name must also be `404.html`.

3. Sign in to the AWS Management Console and open the Amazon S3 console at <https://console.aws.amazon.com/s3/>.
4. In the **Buckets** list, choose the name of the bucket that you want to use to host a static website.
5. Enable static website hosting for your bucket, and enter the exact name of your error document (for example, `404.html`). For more information, see [Enabling website hosting \(p. 1118\)](#).

After enabling static website hosting, proceed to step 6.

6. To upload the error document to your bucket, do one of the following:
 - Drag and drop the error document file into the console bucket listing.

- Choose **Upload**, and follow the prompts to choose and upload the index file.

For step-by-step instructions, see [Uploading objects \(p. 158\)](#).

Step 7: Test your website endpoint

After you configure static website hosting for your bucket, you can test your website endpoint.

Note

Amazon S3 does not support HTTPS access to the website. If you want to use HTTPS, you can use Amazon CloudFront to serve a static website hosted on Amazon S3.

For more information, see [How do I use CloudFront to serve a static website hosted on Amazon S3?](#) and [Requiring HTTPS for communication between viewers and CloudFront](#).

1. Under **Buckets**, choose the name of your bucket.
2. Choose **Properties**.
3. At the bottom of the page, under **Static website hosting**, choose your **Bucket website endpoint**.

Your index document opens in a separate browser window.

You now have a website hosted on Amazon S3. This website is available at the Amazon S3 website endpoint. However, you might have a domain, such as `example.com`, that you want to use to serve the content from the website you created. You might also want to use Amazon S3 root domain support to serve requests for both `http://www.example.com` and `http://example.com`. This requires additional steps. For an example, see [Configuring a static website using a custom domain registered with Route 53 \(p. 97\)](#).

Step 8: Clean up

If you created your static website only as a learning exercise, delete the AWS resources that you allocated so that you no longer accrue charges. After you delete your AWS resources, your website is no longer available. For more information, see [Deleting a bucket \(p. 129\)](#).

Configuring a static website using a custom domain registered with Route 53

Suppose that you want to host a static website on Amazon S3. You've registered a domain with Amazon Route 53 (for example, `example.com`), and you want requests for `http://www.example.com` and `http://example.com` to be served from your Amazon S3 content. You can use this walkthrough to learn how to host a static website and create redirects on Amazon S3 for a website with a custom domain name that is registered with Route 53. You can work with an existing website that you want to host on Amazon S3, or use this walkthrough to start from scratch.

After you complete this walkthrough, you can optionally use Amazon CloudFront to improve the performance of your website. For more information, see [Speeding up your website with Amazon CloudFront \(p. 109\)](#).

Note

Amazon S3 website endpoints do not support HTTPS or access points. If you want to use HTTPS, you can use Amazon CloudFront to serve a static website hosted on Amazon S3.

For more information, see [How do I use CloudFront to serve a static website hosted on Amazon S3?](#) and [Requiring HTTPS for communication between viewers and CloudFront](#).

Automating static website setup with an AWS CloudFormation template

You can use an AWS CloudFormation template to automate your static website setup. The AWS CloudFormation template sets up the components that you need to host a secure static website so that you can focus more on your website's content and less on configuring components.

The AWS CloudFormation template includes the following components:

- Amazon S3 – Creates an Amazon S3 bucket to host your static website.
- CloudFront – Creates a CloudFront distribution to speed up your static website.
- Lambda@Edge – Uses [Lambda@Edge](#) to add security headers to every server response. Security headers are a group of headers in the web server response that tell web browsers to take extra security precautions. For more information, see the blog post [Adding HTTP security headers using Lambda@Edge and Amazon CloudFront](#).

This AWS CloudFormation template is available for you to download and use. For information and instructions, see [Getting started with a secure static website](#) in the *Amazon CloudFront Developer Guide*.

Topics

- [Before you begin \(p. 98\)](#)
- [Step 1: Register a custom domain with Route 53 \(p. 98\)](#)
- [Step 2: Create two buckets \(p. 99\)](#)
- [Step 3: Configure your root domain bucket for website hosting \(p. 99\)](#)
- [Step 4: Configure your subdomain bucket for website redirect \(p. 100\)](#)
- [Step 5: Configure logging for website traffic \(p. 101\)](#)
- [Step 6: Upload index and website content \(p. 102\)](#)
- [Step 7: Upload an error document \(p. 102\)](#)
- [Step 8: Edit S3 Block Public Access settings \(p. 103\)](#)
- [Step 9: Attach a bucket policy \(p. 104\)](#)
- [Step 10: Test your domain endpoint \(p. 105\)](#)
- [Step 11: Add alias records for your domain and subdomain \(p. 105\)](#)
- [Step 12: Test the website \(p. 108\)](#)
- [Speeding up your website with Amazon CloudFront \(p. 109\)](#)
- [Cleaning up your example resources \(p. 112\)](#)

Before you begin

As you follow the steps in this example, you work with the following services:

Amazon Route 53 – You use Route 53 to register domains and to define where you want to route internet traffic for your domain. The example shows how to create Route 53 alias records that route traffic for your domain (`example.com`) and subdomain (`www.example.com`) to an Amazon S3 bucket that contains an HTML file.

Amazon S3 – You use Amazon S3 to create buckets, upload a sample website page, configure permissions so that everyone can see the content, and then configure the buckets for website hosting.

Step 1: Register a custom domain with Route 53

If you don't already have a registered domain name, such as `example.com`, register one with Route 53. For more information, see [Registering a new domain](#) in the *Amazon Route 53 Developer Guide*. After you register your domain name, you can create and configure your Amazon S3 buckets for website hosting.

Step 2: Create two buckets

To support requests from both the root domain and subdomain, you create two buckets.

- **Domain bucket** – example.com
- **Subdomain bucket** – www.example.com

These bucket names must match your domain name exactly. In this example, the domain name is example.com. You host your content out of the root domain bucket (example.com). You create a redirect request for the subdomain bucket (www.example.com). If someone enters www.example.com in their browser, they are redirected to example.com and see the content that is hosted in the Amazon S3 bucket with that name.

To create your buckets for website hosting

The following instructions provide an overview of how to create your buckets for website hosting. For detailed, step-by-step instructions on creating a bucket, see [Creating a bucket \(p. 119\)](#).

1. Sign in to the AWS Management Console and open the Amazon S3 console at <https://console.aws.amazon.com/s3/>.
2. Create your root domain bucket:
 - a. Choose **Create bucket**.
 - b. Enter the **Bucket name** (for example, **example.com**).
 - c. Choose the Region where you want to create the bucket.

Choose a Region that is geographically close to you to minimize latency and costs, or to address regulatory requirements. The Region that you choose determines your Amazon S3 website endpoint. For more information, see [Website endpoints \(p. 1116\)](#).

- d. To accept the default settings and create the bucket, choose **Create**.
3. Create your subdomain bucket:
 - a. Choose **Create bucket**.
 - b. Enter the **Bucket name** (for example, **www.example.com**).
 - c. Choose the Region where you want to create the bucket.

Choose a Region that is geographically close to you to minimize latency and costs, or to address regulatory requirements. The Region that you choose determines your Amazon S3 website endpoint. For more information, see [Website endpoints \(p. 1116\)](#).

- d. To accept the default settings and create the bucket, choose **Create**.

In the next step, you configure example.com for website hosting.

Step 3: Configure your root domain bucket for website hosting

In this step, you configure your root domain bucket (example.com) as a website. This bucket will contain your website content. When you configure a bucket for website hosting, you can access the website using the [Website endpoints \(p. 1116\)](#).

To enable static website hosting

1. Sign in to the AWS Management Console and open the Amazon S3 console at <https://console.aws.amazon.com/s3/>.
2. In the **Buckets** list, choose the name of the bucket that you want to enable static website hosting for.
3. Choose **Properties**.
4. Under **Static website hosting**, choose **Edit**.
5. Choose **Use this bucket to host a website**.
6. Under **Static website hosting**, choose **Enable**.
7. In **Index document**, enter the file name of the index document, typically `index.html`.

The index document name is case sensitive and must exactly match the file name of the HTML index document that you plan to upload to your S3 bucket. When you configure a bucket for website hosting, you must specify an index document. Amazon S3 returns this index document when requests are made to the root domain or any of the subfolders. For more information, see [Configuring an index document \(p. 1122\)](#).

8. To provide your own custom error document for 4XX class errors, in **Error document**, enter the custom error document file name.

The error document name is case sensitive and must exactly match the file name of the HTML error document that you plan to upload to your S3 bucket. If you don't specify a custom error document and an error occurs, Amazon S3 returns a default HTML error document. For more information, see [Configuring a custom error document \(p. 1124\)](#).

9. (Optional) If you want to specify advanced redirection rules, in **Redirection rules**, enter JSON to describe the rules.

For example, you can conditionally route requests according to specific object key names or prefixes in the request. For more information, see [Configure redirection rules to use advanced conditional redirects \(p. 1131\)](#).

10. Choose **Save changes**.

Amazon S3 enables static website hosting for your bucket. At the bottom of the page, under **Static website hosting**, you see the website endpoint for your bucket.

11. Under **Static website hosting**, note the **Endpoint**.

The **Endpoint** is the Amazon S3 website endpoint for your bucket. After you finish configuring your bucket as a static website, you can use this endpoint to test your website.

After you [edit block public access settings](#) and [add a bucket policy](#) that allows public read access, you can use the website endpoint to access your website.

In the next step, you configure your subdomain (`www.example.com`) to redirect requests to your domain (`example.com`).

Step 4: Configure your subdomain bucket for website redirect

After you configure your root domain bucket for website hosting, you can configure your subdomain bucket to redirect all requests to the domain. In this example, all requests for `www.example.com` are redirected to `example.com`.

To configure a redirect request

1. On the Amazon S3 console, in the **Buckets** list, choose your subdomain bucket name (`www.example.com` in this example).
2. Choose **Properties**.
3. Under **Static website hosting**, choose **Edit**.
4. Choose **Redirect requests for an object**.
5. In the **Target bucket** box, enter your root domain, for example, `example.com`.
6. For **Protocol**, choose `http`.
7. Choose **Save changes**.

Step 5: Configure logging for website traffic

If you want to track the number of visitors accessing your website, you can optionally enable logging for your root domain bucket. For more information, see [Logging requests using server access logging \(p. 978\)](#). If you plan to use Amazon CloudFront to speed up your website, you can also use CloudFront logging.

To enable server access logging for your root domain bucket

1. Open the Amazon S3 console at <https://console.aws.amazon.com/s3/>.
2. In the same Region where you created the bucket that is configured as a static website, create a bucket for logging, for example `logs.example.com`.
3. Create a folder for the server access logging log files (for example, `logs`).
4. (Optional) If you want to use CloudFront to improve your website performance, create a folder for the CloudFront log files (for example, `cfn`).

Important

When you create or update a distribution and enable CloudFront logging, CloudFront updates the bucket access control list (ACL) to give the `awslogsdelivery` account `FULL_CONTROL` permissions to write logs to your bucket. For more information, see [Permissions required to configure standard logging and to access your log files](#) in the *Amazon CloudFront Developer Guide*. If the bucket that stores the logs uses the bucket owner enforced setting for S3 Object Ownership to disable ACLs, CloudFront cannot write logs to the bucket. For more information, see [Controlling ownership of objects and disabling ACLs for your bucket \(p. 601\)](#).

5. In the **Buckets** list, choose your root domain bucket.
6. Choose **Properties**.
7. Under **Server access logging**, choose **Edit**.
8. Choose **Enable**.
9. Under the **Target bucket**, choose the bucket and folder destination for the server access logs:
 - Browse to the folder and bucket location:
 1. Choose **Browse S3**.
 2. Choose the bucket name, and then choose the `logs` folder.
 3. Choose **Choose path**.
 - Enter the S3 bucket path, for example, `s3://logs.example.com/logs/`.
10. Choose **Save changes**.

In your log bucket, you can now access your logs. Amazon S3 writes website access logs to your log bucket every 2 hours.

Step 6: Upload index and website content

In this step, you upload your index document and optional website content to your root domain bucket.

When you enable static website hosting for your bucket, you enter the name of the index document (for example, `index.html`). After you enable static website hosting for the bucket, you upload an HTML file with this index document name to your bucket.

To configure the index document

1. Create an `index.html` file.

If you don't have an `index.html` file, you can use the following HTML to create one:

```
<html xmlns="http://www.w3.org/1999/xhtml" >
<head>
    <title>My Website Home Page</title>
</head>
<body>
    <h1>Welcome to my website</h1>
    <p>Now hosted on Amazon S3!</p>
</body>
</html>
```

2. Save the index file locally.

The index document file name must exactly match the index document name that you enter in the **Static website hosting** dialog box. The index document name is case sensitive. For example, if you enter `index.html` for the **Index document** name in the **Static website hosting** dialog box, your index document file name must also be `index.html` and not `Index.html`.

3. Sign in to the AWS Management Console and open the Amazon S3 console at <https://console.aws.amazon.com/s3/>.
4. In the **Buckets** list, choose the name of the bucket that you want to use to host a static website.
5. Enable static website hosting for your bucket, and enter the exact name of your index document (for example, `index.html`). For more information, see [Enabling website hosting \(p. 1118\)](#).

After enabling static website hosting, proceed to step 6.

6. To upload the index document to your bucket, do one of the following:
 - Drag and drop the index file into the console bucket listing.
 - Choose **Upload**, and follow the prompts to choose and upload the index file.

For step-by-step instructions, see [Uploading objects \(p. 158\)](#).

7. (Optional) Upload other website content to your bucket.

Step 7: Upload an error document

When you enable static website hosting for your bucket, you enter the name of the error document (for example, `404.html`). After you enable static website hosting for the bucket, you upload an HTML file with this error document name to your bucket.

To configure an error document

1. Create an error document, for example `404.html`.

2. Save the error document file locally.

The error document name is case sensitive and must exactly match the name that you enter when you enable static website hosting. For example, if you enter `404.html` for the **Error document** name in the **Static website hosting** dialog box, your error document file name must also be `404.html`.

3. Sign in to the AWS Management Console and open the Amazon S3 console at <https://console.aws.amazon.com/s3/>.
4. In the **Buckets** list, choose the name of the bucket that you want to use to host a static website.
5. Enable static website hosting for your bucket, and enter the exact name of your error document (for example, `404.html`). For more information, see [Enabling website hosting \(p. 1118\)](#).

After enabling static website hosting, proceed to step 6.

6. To upload the error document to your bucket, do one of the following:
 - Drag and drop the error document file into the console bucket listing.
 - Choose **Upload**, and follow the prompts to choose and upload the index file.

For step-by-step instructions, see [Uploading objects \(p. 158\)](#).

Step 8: Edit S3 Block Public Access settings

In this example, you edit block public access settings for the domain bucket (`example.com`) to allow public access.

By default, Amazon S3 blocks public access to your account and buckets. If you want to use a bucket to host a static website, you can use these steps to edit your block public access settings.

Warning

Before you complete this step, review [Blocking public access to your Amazon S3 storage \(p. 584\)](#) to ensure that you understand and accept the risks involved with allowing public access. When you turn off block public access settings to make your bucket public, anyone on the internet can access your bucket. We recommend that you block all public access to your buckets.

1. Open the Amazon S3 console at <https://console.aws.amazon.com/s3/>.
2. Choose the name of the bucket that you have configured as a static website.
3. Choose **Permissions**.
4. Under **Block public access (bucket settings)**, choose **Edit**.
5. Clear **Block all public access**, and choose **Save changes**.

Warning

Before you complete this step, review [Blocking public access to your Amazon S3 storage \(p. 584\)](#) to ensure you understand and accept the risks involved with allowing public access. When you turn off block public access settings to make your bucket public, anyone on the internet can access your bucket. We recommend that you block all public access to your buckets.

Block public access (bucket settings)

Public access is granted to buckets and objects through access control lists (ACLs), bucket policies, access point policies, or all. In order to ensure that public access to all your S3 buckets and objects is blocked, turn on Block all public access. These settings apply only to this bucket and its access points. AWS recommends that you turn on Block all public access, but before applying any of these settings, ensure that your applications will work correctly without public access. If you require some level of public access to your buckets or objects within, you can customize the individual settings below to suit your specific storage use cases. [Learn more](#)



Account settings for Block Public Access are currently turned on

Account settings for Block Public Access that are enabled apply even if they are disabled for this bucket.

Block all public access

Turning this setting on is the same as turning on all four settings below. Each of the following settings are independent of one another.

Block public access to buckets and objects granted through new access control lists (ACLs)

S3 will block public access permissions applied to newly added buckets or objects, and prevent the creation of new public access ACLs for existing buckets and objects. This setting doesn't change any existing permissions that allow public access to S3 resources using ACLs.

Block public access to buckets and objects granted through any access control lists (ACLs)

S3 will ignore all ACLs that grant public access to buckets and objects.

Block public access to buckets and objects granted through new public bucket or access point policies

S3 will block new bucket and access point policies that grant public access to buckets and objects. This setting doesn't change any existing policies that allow public access to S3 resources.

Block public and cross-account access to buckets and objects through any public bucket or access point policies

S3 will ignore public and cross-account access for buckets or access points with policies that grant public access to buckets and objects.

Amazon S3 turns off Block Public Access settings for your bucket. To create a public, static website, you might also have to [edit the Block Public Access settings](#) for your account before adding a bucket policy. If account settings for Block Public Access are currently turned on, you see a note under **Block public access (bucket settings)**.

Step 9: Attach a bucket policy

In this example, you attach a bucket policy to the domain bucket (`example.com`) to allow public read access. You replace the `Bucket-Name` in the example bucket policy with the name of your domain bucket, for example `example.com`.

After you edit S3 Block Public Access settings, you can add a bucket policy to grant public read access to your bucket. When you grant public read access, anyone on the internet can access your bucket.

Important

The following policy is an example only and allows full access to the contents of your bucket. Before you proceed with this step, review [How can I secure the files in my Amazon S3 bucket?](#) to ensure that you understand the best practices for securing the files in your S3 bucket and risks involved in granting public access.

1. Under **Buckets**, choose the name of your bucket.
2. Choose **Permissions**.
3. Under **Bucket Policy**, choose **Edit**.
4. To grant public read access for your website, copy the following bucket policy, and paste it in the **Bucket policy editor**.

```
{  
    "Version": "2012-10-17",  
    "Statement": [
```

```
{  
    "Sid": "PublicReadGetObject",  
    "Effect": "Allow",  
    "Principal": "*",  
    "Action": [  
        "s3:GetObject"  
    ],  
    "Resource": [  
        "arn:aws:s3:::Bucket-Name/*"  
    ]  
}  
}
```

5. Update the **Resource** to your bucket name.

In the preceding example bucket policy, *Bucket-Name* is a placeholder for the bucket name. To use this bucket policy with your own bucket, you must update this name to match your bucket name.

6. Choose **Save changes**.

A message appears indicating that the bucket policy has been successfully added.

If you see an error that says `Policy has invalid resource`, confirm that the bucket name in the bucket policy matches your bucket name. For information about adding a bucket policy, see [How do I add an S3 bucket policy?](#)

If you get an error message and cannot save the bucket policy, check your account and bucket Block Public Access settings to confirm that you allow public access to the bucket.

In the next step, you can figure out your website endpoints and test your domain endpoint.

Step 10: Test your domain endpoint

After you configure your domain bucket to host a public website, you can test your endpoint. For more information, see [Website endpoints \(p. 1116\)](#). You can only test the endpoint for your domain bucket because your subdomain bucket is set up for website redirect and not static website hosting.

Note

Amazon S3 does not support HTTPS access to the website. If you want to use HTTPS, you can use Amazon CloudFront to serve a static website hosted on Amazon S3.

For more information, see [How do I use CloudFront to serve a static website hosted on Amazon S3?](#) and [Requiring HTTPS for communication between viewers and CloudFront](#).

1. Under **Buckets**, choose the name of your bucket.
2. Choose **Properties**.
3. At the bottom of the page, under **Static website hosting**, choose your **Bucket website endpoint**.

Your index document opens in a separate browser window.

In the next step, you use Amazon Route 53 to enable customers to use both of your custom URLs to navigate to your site.

Step 11: Add alias records for your domain and subdomain

In this step, you create the alias records that you add to the hosted zone for your domain maps `example.com` and `www.example.com`. Instead of using IP addresses, the alias records use the Amazon

S3 website endpoints. Amazon Route 53 maintains a mapping between the alias records and the IP addresses where the Amazon S3 buckets reside. You create two alias records, one for your root domain and one for your subdomain.

Add an alias record for your root domain and subdomain

To add an alias record for your root domain (`example.com`)

1. Open the Route 53 console at <https://console.aws.amazon.com/route53/>.

Note

If you don't already use Route 53, see [Step 1: Register a domain](#) in the *Amazon Route 53 Developer Guide*. After completing your setup, you can resume the instructions.

2. Choose **Hosted zones**.
3. In the list of hosted zones, choose the name of the hosted zone that matches your domain name.
4. Choose **Create record**.
5. Choose **Switch to wizard**.

Note

If you want to use quick create to create your alias records, see [Configuring Route 53 to route traffic to an S3 Bucket](#).

6. Choose **Simple routing**, and choose **Next**.
7. Choose **Define simple record**.
8. In **Record name**, accept the default value, which is the name of your hosted zone and your domain.
9. In **Value/Route traffic to**, choose **Alias to S3 website endpoint**.
10. Choose the Region.
11. Choose the S3 bucket.

The bucket name should match the name that appears in the **Name** box. In the **Choose S3 bucket** list, the bucket name appears with the Amazon S3 website endpoint for the Region where the bucket was created, for example, `s3-website-us-west-1.amazonaws.com` (`example.com`).

Choose S3 bucket lists a bucket if:

- You configured the bucket as a static website.
- The bucket name is the same as the name of the record that you're creating.
- The current AWS account created the bucket.

If your bucket does not appear in the **Choose S3 bucket** list, enter the Amazon S3 website endpoint for the Region where the bucket was created, for example, `s3-website-us-west-2.amazonaws.com`. For a complete list of Amazon S3 website endpoints, see [Amazon S3 Website endpoints](#). For more information about the alias target, see [Value/route traffic to](#) in the *Amazon Route 53 Developer Guide*.

12. In **Record type**, choose **A - Routes traffic to an IPv4 address and some AWS resources**.
13. For **Evaluate target health**, choose **No**.
14. Choose **Define simple record**.

To add an alias record for your subdomain (`www.example.com`)

1. Under **Configure records**, choose **Define simple record**.
2. In **Record name** for your subdomain, type `www`.
3. In **Value/Route traffic to**, choose **Alias to S3 website endpoint**.
4. Choose the Region.

5. Choose the S3 bucket, for example, s3-website-us-west-2.amazonaws.com (www.example.com).

If your bucket does not appear in the **Choose S3 bucket** list, enter the Amazon S3 website endpoint for the Region where the bucket was created, for example, **s3-website-us-west-2.amazonaws.com**. For a complete list of Amazon S3 website endpoints, see [Amazon S3 Website endpoints](#). For more information about the alias target, see [Value/route traffic to](#) in the [Amazon Route 53 Developer Guide](#).

6. In **Record type**, choose **A - Routes traffic to an IPv4 address and some AWS resources**.
7. For **Evaluate target health**, choose **No**.
8. Choose **Define simple record**.
9. On the **Configure records** page, choose **Create records**.

Note

Changes generally propagate to all Route 53 servers within 60 seconds. When propagation is done, you can route traffic to your Amazon S3 bucket by using the names of the alias records that you created in this procedure.

Add an alias record for your root domain and subdomain (old Route 53 console)

To add an alias record for your root domain (`example.com`)

The Route 53 console has been redesigned. In the Route 53 console you can temporarily use the old console. If you choose to work with the old Route 53 console, use the procedure below.

1. Open the Route 53 console at <https://console.aws.amazon.com/route53>.

Note

If you don't already use Route 53, see [Step 1: Register a domain in the Amazon Route 53 Developer Guide](#). After completing your setup, you can resume the instructions.

2. Choose **Hosted Zones**.
3. In the list of hosted zones, choose the name of the hosted zone that matches your domain name.
4. Choose **Create Record Set**.
5. Specify the following values:

Name

Accept the default value, which is the name of your hosted zone and your domain.

For the root domain, you don't need to enter any additional information in the **Name** field.

Type

Choose **A – IPv4 address**.

Alias

Choose **Yes**.

Alias Target

In the **S3 website endpoints** section of the list, choose your bucket name.

The bucket name should match the name that appears in the **Name** box. In the **Alias Target** listing, the bucket name is followed by the Amazon S3 website endpoint for the Region where the bucket was created, for example `example.com (s3-website-us-west-2.amazonaws.com)`. **Alias Target** lists a bucket if:

- You configured the bucket as a static website.

- The bucket name is the same as the name of the record that you're creating.
- The current AWS account created the bucket.

If your bucket does not appear in the **Alias Target** listing, enter the Amazon S3 website endpoint for the Region where the bucket was created, for example, s3-website-us-west-2. For a complete list of Amazon S3 website endpoints, see [Amazon S3 Website endpoints](#). For more information about the alias target, see [Value/route traffic to](#) in the [Amazon Route 53 Developer Guide](#).

Routing Policy

Accept the default value of **Simple**.

Evaluate Target Health

Accept the default value of **No**.

6. Choose **Create**.

To add an alias record for your subdomain (`www.example.com`)

1. In the hosted zone for your root domain (example.com), choose **Create Record Set**.
2. Specify the following values:

Name

For the subdomain, enter www in the box.

Type

Choose **A – IPv4 address**.

Alias

Choose **Yes**.

Alias Target

In the **S3 website endpoints** section of the list, choose the same bucket name that appears in the **Name** field—for example, www.example.com (s3-website-us-west-2.amazonaws.com).

Routing Policy

Accept the default value of **Simple**.

Evaluate Target Health

Accept the default value of **No**.

3. Choose **Create**.

Note

Changes generally propagate to all Route 53 servers within 60 seconds. When propagation is done, you can route traffic to your Amazon S3 bucket by using the names of the alias records that you created in this procedure.

Step 12: Test the website

Verify that the website and the redirect work correctly. In your browser, enter your URLs. In this example, you can try the following URLs:

- **Domain** (`http://example.com`) – Displays the index document in the example.com bucket.

- **Subdomain** (`http://www.example.com`) – Redirects your request to `http://example.com`. You see the index document in the `example.com` bucket.

If your website or redirect links don't work, you can try the following:

- **Clear cache** – Clear the cache of your web browser.
- **Check name servers** – If your web page and redirect links don't work after you've cleared your cache, you can compare the name servers for your domain and the name servers for your hosted zone. If the name servers don't match, you might need to update your domain name servers to match those listed under your hosted zone. For more information, see [Adding or changing name servers and glue records for a domain](#).

After you've successfully tested your root domain and subdomain, you can set up an [Amazon CloudFront](#) distribution to improve the performance of your website and provide logs that you can use to review website traffic. For more information, see [Speeding up your website with Amazon CloudFront \(p. 109\)](#).

Speeding up your website with Amazon CloudFront

You can use [Amazon CloudFront](#) to improve the performance of your Amazon S3 website. CloudFront makes your website files (such as HTML, images, and video) available from data centers around the world (known as *edge locations*). When a visitor requests a file from your website, CloudFront automatically redirects the request to a copy of the file at the nearest edge location. This results in faster download times than if the visitor had requested the content from a data center that is located farther away.

CloudFront caches content at edge locations for a period of time that you specify. If a visitor requests content that has been cached for longer than the expiration date, CloudFront checks the origin server to see if a newer version of the content is available. If a newer version is available, CloudFront copies the new version to the edge location. Changes that you make to the original content are replicated to edge locations as visitors request the content.

Automating set up with an AWS CloudFormation template

For more information about using an AWS CloudFormation template to configure a secure static website that creates a CloudFront distribution to serve your website, see [Getting started with a secure static website](#) in the *Amazon CloudFront Developer Guide*.

Topics

- [Step 1: Create a CloudFront distribution \(p. 109\)](#)
- [Step 2: Update the record sets for your domain and subdomain \(p. 111\)](#)
- [\(Optional\) Step 3: Check the log files \(p. 111\)](#)

Step 1: Create a CloudFront distribution

First, you create a CloudFront distribution. This makes your website available from data centers around the world.

To create a distribution with an Amazon S3 origin

1. Open the CloudFront console at <https://console.aws.amazon.com/cloudfront/v3/home>.
2. Choose **Create Distribution**.
3. On the **Select a delivery method for your content** page, under **Web**, choose **Get Started**.
4. On the **Create Distribution** page, in the **Origin Settings** section, for **Origin Domain Name**, enter the Amazon S3 website endpoint for your bucket—for example, `example.com.s3-website.us-west-1.amazonaws.com`.

CloudFront fills in the **Origin ID** for you.

5. For **Default Cache Behavior Settings**, keep the values set to the defaults.

With the default settings for **Viewer Protocol Policy**, you can use HTTPS for your static website. For more information about these configuration options, see [Values that You Specify When You Create or Update a Web Distribution](#) in the *Amazon CloudFront Developer Guide*.

6. For **Distribution Settings**, do the following:

- a. Leave **Price Class** set to **Use All Edge Locations (Best Performance)**.
- b. Set **Alternate Domain Names (CNAMEs)** to the root domain and www subdomain. In this tutorial, these are example.com and www.example.com.

Important

Before you perform this step, note the [requirements for using alternate domain names](#), in particular the need for a valid SSL/TLS certificate.

- c. For **SSL Certificate**, choose **Custom SSL Certificate (example.com)**, and choose the custom certificate that covers the domain and subdomain names.

For more information, see [SSL Certificate](#) in the *Amazon CloudFront Developer Guide*.

- d. In **Default Root Object**, enter the name of your index document, for example, index.html.

If the URL used to access the distribution doesn't contain a file name, the CloudFront distribution returns the index document. The **Default Root Object** should exactly match the name of the index document for your static website. For more information, see [Configuring an index document \(p. 1122\)](#).

- e. Set **Logging to On**.

Important

When you create or update a distribution and enable CloudFront logging, CloudFront updates the bucket access control list (ACL) to give the awslogsdelivery account **FULL_CONTROL** permissions to write logs to your bucket. For more information, see [Permissions required to configure standard logging and to access your log files](#) in the *Amazon CloudFront Developer Guide*. If the bucket that stores the logs uses the bucket owner enforced setting for S3 Object Ownership to disable ACLs, CloudFront cannot write logs to the bucket. For more information, see [Controlling ownership of objects and disabling ACLs for your bucket \(p. 601\)](#).

- f. For **Bucket for Logs**, choose the logging bucket that you created.

For more information about configuring a logging bucket, see [\(Optional\) Logging web traffic \(p. 1130\)](#).

- g. If you want to store the logs that are generated by traffic to the CloudFront distribution in a folder, in **Log Prefix**, enter the folder name.
- h. Keep all other settings at their default values.

7. Choose **Create Distribution**.

8. To see the status of the distribution, find the distribution in the console and check the **Status** column.

A status of **InProgress** indicates that the distribution is not yet fully deployed.

After your distribution is deployed, you can reference your content with the new CloudFront domain name.

9. Record the value of **Domain Name** shown in the CloudFront console, for example, dj4p1rv6mvubz.cloudfront.net.
10. To verify that your CloudFront distribution is working, enter the domain name of the distribution in a web browser.

If your website is visible, the CloudFront distribution works. If your website has a custom domain registered with Amazon Route 53, you will need the CloudFront domain name to update the record set in the next step.

Step 2: Update the record sets for your domain and subdomain

Now that you have successfully created a CloudFront distribution, update the alias record in Route 53 to point to the new CloudFront distribution.

To update the alias record to point to a CloudFront distribution

1. Open the Route 53 console at <https://console.aws.amazon.com/route53/>.
2. In the left navigation, choose **Hosted zones**.
3. On the **Hosted Zones** page, choose the hosted zone that you created for your subdomain, for example, www.example.com.
4. Under **Records**, select the A record that you created for your subdomain.
5. Under **Record details**, choose **Edit record**.
6. Under **Route traffic to**, choose **Alias to CloudFront distribution**.
7. Under **Choose distribution**, choose the CloudFront distribution.
8. Choose **Save**.
9. To redirect the A record for the root domain to the CloudFront distribution, repeat this procedure for the root domain, for example, example.com.

The update to the record sets takes effect within 2–48 hours.

10. To see whether the new A records have taken effect, in a web browser, enter your subdomain URL, for example, <http://www.example.com>.

If the browser no longer redirects you to the root domain (for example, <http://example.com>), the new A records are in place. When the new A record has taken effect, traffic routed by the new A record to the CloudFront distribution is not redirected to the root domain. Any visitors who reference the site by using <http://example.com> or <http://www.example.com> are redirected to the nearest CloudFront edge location, where they benefit from faster download times.

Tip

Browsers can cache redirect settings. If you think the new A record settings should have taken effect, but your browser still redirects <http://www.example.com> to <http://example.com>, try clearing your browser history and cache, closing and reopening your browser application, or using a different web browser.

(Optional) Step 3: Check the log files

The access logs tell you how many people are visiting the website. They also contain valuable business data that you can analyze with other services, such as [Amazon EMR](#).

CloudFront logs are stored in the bucket and folder that you choose when you create a CloudFront distribution and enable logging. CloudFront writes logs to your log bucket within 24 hours from when the corresponding requests are made.

To see the log files for your website

1. Open the Amazon S3 console at <https://console.aws.amazon.com/s3/>.
2. Choose the name of the logging bucket for your website.
3. Choose the CloudFront logs folder.

4. Download the .gzip files written by CloudFront before opening them.

If you created your website only as a learning exercise, you can delete the resources that you allocated so that you no longer accrue charges. To do so, see [Cleaning up your example resources \(p. 112\)](#). After you delete your AWS resources, your website is no longer available.

Cleaning up your example resources

If you created your static website as a learning exercise, you should delete the AWS resources that you allocated so that you no longer accrue charges. After you delete your AWS resources, your website is no longer available.

Tasks

- [Step 1: Delete the Amazon CloudFront distribution \(p. 112\)](#)
- [Step 2: Delete the Route 53 hosted zone \(p. 112\)](#)
- [Step 3: Disable logging and delete your S3 bucket \(p. 113\)](#)

Step 1: Delete the Amazon CloudFront distribution

Before you delete an Amazon CloudFront distribution, you must disable it. A disabled distribution is no longer functional and does not accrue charges. You can enable a disabled distribution at any time. After you delete a disabled distribution, it is no longer available.

To disable and delete a CloudFront distribution

1. Open the CloudFront console at <https://console.aws.amazon.com/cloudfront/v3/home>.
2. Select the distribution that you want to disable, and then choose **Disable**.
3. When prompted for confirmation, choose **Yes, Disable**.
4. Select the disabled distribution, and then choose **Delete**.
5. When prompted for confirmation, choose **Yes, Delete**.

Step 2: Delete the Route 53 hosted zone

Before you delete the hosted zone, you must delete the record sets that you created. You don't need to delete the NS and SOA records; these are automatically deleted when you delete the hosted zone.

To delete the record sets

1. Open the Route 53 console at <https://console.aws.amazon.com/route53/>.
2. In the list of domain names, select your domain name, and then choose **Go to Record Sets**.
3. In the list of record sets, select the A records that you created.

The type of each record set is listed in the **Type** column.
4. Choose **Delete Record Set**.
5. When prompted for confirmation, choose **Confirm**.

To delete a Route 53 hosted zone

1. Continuing from the previous procedure, choose **Back to Hosted Zones**.
2. Select your domain name, and then choose **Delete Hosted Zone**.

3. When prompted for confirmation, choose **Confirm**.

Step 3: Disable logging and delete your S3 bucket

Before you delete your S3 bucket, make sure that logging is disabled for the bucket. Otherwise, AWS continues to write logs to your bucket as you delete it.

To disable logging for a bucket

1. Open the Amazon S3 console at <https://console.aws.amazon.com/s3/>.
2. Under **Buckets**, choose your bucket name, and then choose **Properties**.
3. From **Properties**, choose **Logging**.
4. Clear the **Enabled** check box.
5. Choose **Save**.

Now, you can delete your bucket. For more information, see [Deleting a bucket \(p. 129\)](#).

Creating, configuring, and working with Amazon S3 buckets

To store your data in Amazon S3, you work with resources known as buckets and objects. A *bucket* is a container for objects. An *object* is a file and any metadata that describes that file.

To store an object in Amazon S3, you create a bucket and then upload the object to a bucket. When the object is in the bucket, you can open it, download it, and move it. When you no longer need an object or a bucket, you can clean up your resources.

Note

With Amazon S3, you pay only for what you use. For more information about Amazon S3 features and pricing, see [Amazon S3](#). If you are a new Amazon S3 customer, you can get started with Amazon S3 for free. For more information, see [AWS Free Tier](#).

The topics in this section provide an overview of working with buckets in Amazon S3. They include information about naming, creating, accessing, and deleting buckets. For more information about viewing or listing objects in a bucket, see [Organizing, listing, and working with your objects \(p. 243\)](#).

Topics

- [Buckets overview \(p. 114\)](#)
- [Bucket naming rules \(p. 118\)](#)
- [Creating a bucket \(p. 119\)](#)
- [Viewing the properties for an S3 bucket \(p. 124\)](#)
- [Methods for accessing a bucket \(p. 125\)](#)
- [Emptying a bucket \(p. 127\)](#)
- [Deleting a bucket \(p. 129\)](#)
- [Setting default server-side encryption behavior for Amazon S3 buckets \(p. 132\)](#)
- [Configuring fast, secure file transfers using Amazon S3 Transfer Acceleration \(p. 136\)](#)
- [Using Requester Pays buckets for storage transfers and usage \(p. 144\)](#)
- [Bucket restrictions and limitations \(p. 147\)](#)

Buckets overview

To upload your data (photos, videos, documents, etc.) to Amazon S3, you must first create an S3 bucket in one of the AWS Regions.

A bucket is a container for objects stored in Amazon S3. You can store any number of objects in a bucket and can have up to 100 buckets in your account. To request an increase, visit the [Service Quotas Console](#).

Every object is contained in a bucket. For example, if the object named `photos/puppy.jpg` is stored in the `DOC-EXAMPLE-BUCKET` bucket in the US West (Oregon) Region, then it is addressable using the URL `https://DOC-EXAMPLE-BUCKET.s3.us-west-2.amazonaws.com/photos/puppy.jpg`. For more information, see [Accessing a Bucket \(p. 125\)](#).

In terms of implementation, buckets and objects are AWS resources, and Amazon S3 provides APIs for you to manage them. For example, you can create a bucket and upload objects using the Amazon S3 API. You can also use the Amazon S3 console to perform these operations. The console uses the Amazon S3 APIs to send requests to Amazon S3.

This section describes how to work with buckets. For information about working with objects, see [Amazon S3 objects overview \(p. 149\)](#).

Amazon S3 supports global buckets, which means that each bucket name must be unique across all AWS accounts in all the AWS Regions within a partition. A partition is a grouping of Regions. AWS currently has three partitions: aws (Standard Regions), aws-cn (China Regions), and aws-us-gov (AWS GovCloud (US)).

After a bucket is created, the name of that bucket cannot be used by another AWS account in the same partition until the bucket is deleted. You should not depend on specific bucket naming conventions for availability or security verification purposes. For bucket naming guidelines, see [Bucket naming rules \(p. 118\)](#).

Amazon S3 creates buckets in a Region that you specify. To optimize latency, minimize costs, or address regulatory requirements, choose any AWS Region that is geographically close to you. For example, if you reside in Europe, you might find it advantageous to create buckets in the Europe (Ireland) or Europe (Frankfurt) Regions. For a list of Amazon S3 Regions, see [Regions and Endpoints](#) in the *AWS General Reference*.

Note

Objects that belong to a bucket that you create in a specific AWS Region never leave that Region, unless you explicitly transfer them to another Region. For example, objects that are stored in the Europe (Ireland) Region never leave it.

Topics

- [About permissions \(p. 115\)](#)
- [Managing public access to buckets \(p. 115\)](#)
- [Bucket configuration options \(p. 116\)](#)

About permissions

You can use your AWS account root user credentials to create a bucket and perform any other Amazon S3 operation. However, we recommend that you do not use the root user credentials of your AWS account to make requests, such as to create a bucket. Instead, create an AWS Identity and Access Management (IAM) user, and grant that user full access (users by default have no permissions).

These users are referred to as *administrators*. You can use the administrator user credentials, instead of the root user credentials of your account, to interact with AWS and perform tasks, such as create a bucket, create users, and grant them permissions.

For more information, see [AWS account root user credentials and IAM user credentials](#) in the *AWS General Reference* and [Security best practices in IAM](#) in the *IAM User Guide*.

The AWS account that creates a resource owns that resource. For example, if you create an IAM user in your AWS account and grant the user permission to create a bucket, the user can create a bucket. But the user does not own the bucket; the AWS account that the user belongs to owns the bucket. The user needs additional permission from the resource owner to perform any other bucket operations. For more information about managing permissions for your Amazon S3 resources, see [Identity and access management in Amazon S3 \(p. 394\)](#).

Managing public access to buckets

Public access is granted to buckets and objects through access control lists (ACLs), bucket policies, or both. To help you manage public access to Amazon S3 resources, Amazon S3 provides settings to block public access. Amazon S3 Block Public Access settings can override ACLs and bucket policies so that you can enforce uniform limits on public access to these resources. You can apply Block Public Access settings to individual buckets or to all buckets in your account.

To help ensure that all of your Amazon S3 buckets and objects have their public access blocked, we recommend that you turn on all four settings for Block Public Access for your account. These settings block all public access for all current and future buckets.

Before applying these settings, verify that your applications will work correctly without public access. If you require some level of public access to your buckets or objects—for example, to host a static website as described at [Hosting a static website using Amazon S3 \(p. 1116\)](#)—you can customize the individual settings to suit your storage use cases. For more information, see [Blocking public access to your Amazon S3 storage \(p. 584\)](#).

Note

If you see an `Error` when you list your buckets and their public access settings, you might not have the required permissions. Check to make sure you have the following permissions added to your user or role policy:

```
s3:GetAccountPublicAccessBlock  
s3:GetBucketPublicAccessBlock  
s3:GetBucketPolicyStatus  
s3:GetBucketLocation  
s3:GetBucketAcl  
s3>ListAccessPoints  
s3>ListAllMyBuckets
```

In some rare cases, requests can also fail because of an AWS Region outage.

Bucket configuration options

Amazon S3 supports various options for you to configure your bucket. For example, you can configure your bucket for website hosting, add a configuration to manage the lifecycle of objects in the bucket, and configure the bucket to log all access to the bucket. Amazon S3 supports subresources for you to store and manage the bucket configuration information. You can use the Amazon S3 API to create and manage these subresources. However, you can also use the console or the AWS SDKs.

Note

There are also object-level configurations. For example, you can configure object-level permissions by configuring an access control list (ACL) specific to that object.

These are referred to as subresources because they exist in the context of a specific bucket or object. The following table lists subresources that enable you to manage bucket-specific configurations.

Subresource	Description
<code>cors</code> (cross-origin resource sharing)	You can configure your bucket to allow cross-origin requests. For more information, see Using cross-origin resource sharing (CORS) (p. 573) .
<code>event notification</code>	You can enable your bucket to send you notifications of specified bucket events. For more information, see Amazon S3 Event Notifications (p. 1017) .
<code>lifecycle</code>	You can define lifecycle rules for objects in your bucket that have a well-defined lifecycle. For example, you can define a rule to archive objects one year after creation, or delete an object 10 years after creation. For more information, see Managing your storage lifecycle (p. 701) .
<code>location</code>	When you create a bucket, you specify the AWS Region where you want Amazon S3 to create the bucket. Amazon S3 stores this information in the location subresource and provides an API for you to retrieve this information.

Subresource	Description
<i>logging</i>	<p>Logging enables you to track requests for access to your bucket. Each access log record provides details about a single access request, such as the requester, bucket name, request time, request action, response status, and error code, if any. Access log information can be useful in security and access audits. It can also help you learn about your customer base and understand your Amazon S3 bill.</p> <p>For more information, see Logging requests using server access logging (p. 978).</p>
<i>object locking</i>	<p>To use S3 Object Lock, you must enable it for a bucket. You can also optionally configure a default retention mode and period that applies to new objects that are placed in the bucket.</p> <p>For more information, see Bucket configuration (p. 683).</p>
<i>policy and ACL</i> (access control list)	<p>All your resources (such as buckets and objects) are private by default. Amazon S3 supports both bucket policy and access control list (ACL) options for you to grant and manage bucket-level permissions. Amazon S3 stores the permission information in the <i>policy</i> and <i>acl</i> subresources.</p> <p>For more information, see Identity and access management in Amazon S3 (p. 394).</p>
<i>replication</i>	<p>Replication is the automatic, asynchronous copying of objects across buckets in different or the same AWS Regions. For more information, see Replicating objects (p. 753).</p>
<i>requestPayment</i>	<p>By default, the AWS account that creates the bucket (the bucket owner) pays for downloads from the bucket. Using this subresource, the bucket owner can specify that the person requesting the download will be charged for the download. Amazon S3 provides an API for you to manage this subresource.</p> <p>For more information, see Using Requester Pays buckets for storage transfers and usage (p. 144).</p>
<i>tagging</i>	<p>You can add cost allocation tags to your bucket to categorize and track your AWS costs. Amazon S3 provides the <i>tagging</i> subresource to store and manage tags on a bucket. Using tags you apply to your bucket, AWS generates a cost allocation report with usage and costs aggregated by your tags.</p> <p>For more information, see Billing and usage reporting for S3 buckets (p. 836).</p>
<i>transfer acceleration</i>	<p>Transfer Acceleration enables fast, easy, and secure transfers of files over long distances between your client and an S3 bucket. Transfer Acceleration takes advantage of the globally distributed edge locations of Amazon CloudFront.</p> <p>For more information, see Configuring fast, secure file transfers using Amazon S3 Transfer Acceleration (p. 136).</p>
<i>versioning</i>	<p>Versioning helps you recover accidental overwrites and deletes.</p> <p>We recommend versioning as a best practice to recover objects from being deleted or overwritten by mistake.</p> <p>For more information, see Using versioning in S3 buckets (p. 638).</p>

Subresource	Description
<i>website</i>	You can configure your bucket for static website hosting. Amazon S3 stores this configuration by creating a <i>website</i> subresource. For more information, see Hosting a static website using Amazon S3 (p. 1116) .

Bucket naming rules

The following rules apply for naming buckets in Amazon S3:

- Bucket names must be between 3 (min) and 63 (max) characters long.
- Bucket names can consist only of lowercase letters, numbers, dots (.), and hyphens (-).
- Bucket names must begin and end with a letter or number.
- Bucket names must not contain two adjacent periods.
- Bucket names must not be formatted as an IP address (for example, 192.168.5.4).
- Bucket names must not start with the prefix `xn--`.
- Bucket names must not end with the suffix `-s3alias`. This suffix is reserved for access point alias names. For more information, see [Using a bucket-style alias for your access point \(p. 314\)](#).
- Bucket names must be unique across all AWS accounts in all the AWS Regions within a partition. A partition is a grouping of Regions. AWS currently has three partitions: `aws` (Standard Regions), `aws-cn` (China Regions), and `aws-us-gov` (AWS GovCloud (US)).
- A bucket name cannot be used by another AWS account in the same partition until the bucket is deleted.
- Buckets used with Amazon S3 Transfer Acceleration can't have dots (.) in their names. For more information about Transfer Acceleration, see [Configuring fast, secure file transfers using Amazon S3 Transfer Acceleration \(p. 136\)](#).

For best compatibility, we recommend that you avoid using dots (.) in bucket names, except for buckets that are used only for static website hosting. If you include dots in a bucket's name, you can't use virtual-host-style addressing over HTTPS, unless you perform your own certificate validation. This is because the security certificates used for virtual hosting of buckets don't work for buckets with dots in their names.

This limitation doesn't affect buckets used for static website hosting, because static website hosting is only available over HTTP. For more information about virtual-host-style addressing, see [Virtual hosting of buckets \(p. 1175\)](#). For more information about static website hosting, see [Hosting a static website using Amazon S3 \(p. 1116\)](#).

Note

Before March 1, 2018, buckets created in the US East (N. Virginia) Region could have names that were up to 255 characters long and included uppercase letters and underscores. Beginning March 1, 2018, new buckets in US East (N. Virginia) must conform to the same rules applied in all other Regions.

For information on object key names, see [Creating object key names](#).

Example bucket names

The following example bucket names are valid and follow the recommended naming guidelines:

- `docexamplebucket1`

- log-delivery-march-2020
- my-hosted-content

The following example bucket names are valid but not recommended for uses other than static website hosting:

- docexamplewebsite.com
- www.docexamplewebsite.com
- my.example.s3.bucket

The following example bucket names are *not* valid:

- doc_example_bucket (contains underscores)
- DocExampleBucket (contains uppercase letters)
- doc-example-bucket- (ends with a hyphen)

Creating a bucket

To upload your data to Amazon S3, you must first create an Amazon S3 bucket in one of the AWS Regions. When you create a bucket, you must choose a bucket name and Region. You can optionally choose other storage management options for the bucket. After you create a bucket, you cannot change the bucket name or Region. For information about naming buckets, see [Bucket naming rules \(p. 118\)](#).

The AWS account that creates the bucket owns it. You can upload any number of objects to the bucket. By default, you can create up to 100 buckets in each of your AWS accounts. If you need more buckets, you can increase your account bucket limit to a maximum of 1,000 buckets by submitting a service limit increase. To learn how to submit a bucket limit increase, see [AWS service quotas](#) in the *AWS General Reference*. You can store any number of objects in a bucket.

S3 Object Ownership is an Amazon S3 bucket-level setting that you can use to disable access control lists (ACLs) and take ownership of every object in your bucket, simplifying access management for data stored in Amazon S3. By default, when another AWS account uploads an object to your Amazon S3 bucket, that account (the object writer) owns the object, has access to it, and can grant other users access to it through ACLs. When you create a bucket, you can apply the bucket owner enforced setting for Object Ownership to change this default behavior so that ACLs are disabled and you, as the bucket owner, automatically own every object in your bucket. As a result, access control for your data is based on policies. For more information, see [Controlling ownership of objects and disabling ACLs for your bucket \(p. 601\)](#).

You can use the Amazon S3 console, Amazon S3 APIs, AWS CLI, or AWS SDKs to create a bucket. For more information about the permissions required to create a bucket, see [CreateBucket](#) in the *Amazon Simple Storage Service API Reference*.

Using the S3 console

1. Sign in to the AWS Management Console and open the Amazon S3 console at <https://console.aws.amazon.com/s3/>.
2. Choose **Create bucket**.

The **Create bucket** wizard opens.

3. In **Bucket name**, enter a DNS-compliant name for your bucket.

The bucket name must:

- Be unique across all of Amazon S3.
- Be between 3 and 63 characters long.
- Not contain uppercase characters.
- Start with a lowercase letter or number.

After you create the bucket, you cannot change its name. For information about naming buckets, see [Bucket naming rules \(p. 118\)](#).

Important

Avoid including sensitive information, such as account number, in the bucket name. The bucket name is visible in the URLs that point to the objects in the bucket.

4. In **Region**, choose the AWS Region where you want the bucket to reside.

Choose a Region close to you to minimize latency and costs and address regulatory requirements. Objects stored in a Region never leave that Region unless you explicitly transfer them to another Region. For a list of Amazon S3 AWS Regions, see [AWS service endpoints](#) in the *Amazon Web Services General Reference*.

5. Under **Object Ownership**, to disable or enable ACLs and control ownership of objects uploaded in your bucket, choose one of the following settings:

ACLs disabled

- **Bucket owner enforced** – ACLs are disabled, and the bucket owner automatically owns and has full control over every object in the bucket. ACLs no longer affect permissions to data in the S3 bucket. The bucket uses policies to define access control.

To require that all new buckets are created with ACLs disabled by using IAM or AWS Organizations policies, see [Disabling ACLs for all new buckets \(bucket owner enforced\) \(p. 622\)](#).

ACLs enabled

- **Bucket owner preferred** – The bucket owner owns and has full control over new objects that other accounts write to the bucket with the `bucket-owner-full-control` canned ACL.

If you apply the bucket owner preferred setting, to require all Amazon S3 uploads to include the `bucket-owner-full-control` canned ACL, you can [add a bucket policy \(p. 623\)](#) that only allows object uploads that use this ACL.

- **Object writer** – The AWS account that uploads an object owns the object, has full control over it, and can grant other users access to it through ACLs.

Note

To apply the **Bucket owner enforced** setting or the **Bucket owner preferred** setting, you must have the following permission: `s3:CreateBucket` and `s3:PutBucketOwnershipControls`.

6. In **Bucket settings for Block Public Access**, choose the Block Public Access settings that you want to apply to the bucket.

We recommend that you keep all settings enabled unless you know that you need to turn off one or more of them for your use case, such as to host a public website. Block Public Access settings that you enable for the bucket are also enabled for all access points that you create on the bucket. For more information about blocking public access, see [Blocking public access to your Amazon S3 storage \(p. 584\)](#).

7. (Optional) If you want to enable S3 Object Lock, do the following:

- a. Choose **Advanced settings**.

Important

You can only enable S3 Object Lock for a bucket when you create it. If you enable Object Lock for the bucket, you cannot disable it later. Enabling Object Lock also enables versioning for the bucket. After you enable Object Lock for the bucket, you must configure the Object Lock default retention and legal hold settings to protect new objects from being deleted or overwritten. For more information, see [Configuring S3 Object Lock using the console \(p. 684\)](#).

- b. If you want to enable Object Lock, choose **Enable**, read the warning that appears, and acknowledge it.

For more information about the S3 Object Lock feature, see [Using S3 Object Lock \(p. 680\)](#).

Note

To create an Object Lock enabled bucket, you must have the following permissions: s3:CreateBucket, s3:PutBucketVersioning and s3:PutBucketObjectLockConfiguration.

8. Choose **Create bucket**.

Using the AWS SDKs

When you use the AWS SDKs to create a bucket, you must create a client and then use the client to send a request to create a bucket. As a best practice, you should create your client and bucket in the same AWS Region. If you don't specify a Region when you create a client or a bucket, Amazon S3 uses the default Region US East (N. Virginia).

To create a client to access a dual-stack endpoint, you must specify an AWS Region. For more information, see [Dual-stack endpoints \(p. 1142\)](#). For a list of available AWS Regions, see [Regions and endpoints](#) in the *AWS General Reference*.

When you create a client, the Region maps to the Region-specific endpoint. The client uses this endpoint to communicate with Amazon S3: s3.<region>.amazonaws.com. If your Region launched after March 20, 2019, your client and bucket must be in the same Region. However, you can use a client in the US East (N. Virginia) Region to create a bucket in any Region that launched before March 20, 2019. For more information, see [Legacy endpoints \(p. 1180\)](#).

These AWS SDK code examples perform the following tasks:

- **Create a client by explicitly specifying an AWS Region** — In the example, the client uses the s3.us-west-2.amazonaws.com endpoint to communicate with Amazon S3. You can specify any AWS Region. For a list of AWS Regions, see [Regions and endpoints](#) in the *AWS General Reference*.
- **Send a create bucket request by specifying only a bucket name** — The client sends a request to Amazon S3 to create the bucket in the Region where you created a client.
- **Retrieve information about the location of the bucket** — Amazon S3 stores bucket location information in the *location* subresource that is associated with the bucket.

Java

This example shows how to create an Amazon S3 bucket using the AWS SDK for Java. For instructions on creating and testing a working sample, see [Testing the Amazon S3 Java Code Examples \(p. 1192\)](#).

```
import com.amazonaws.AmazonServiceException;
```

```
import com.amazonaws.SdkClientException;
import com.amazonaws.auth.profile.ProfileCredentialsProvider;
import com.amazonaws.regions.Regions;
import com.amazonaws.services.s3.AmazonS3;
import com.amazonaws.services.s3.AmazonS3ClientBuilder;
import com.amazonaws.services.s3.model.CreateBucketRequest;
import com.amazonaws.services.s3.model.GetBucketLocationRequest;

import java.io.IOException;

public class CreateBucket2 {

    public static void main(String[] args) throws IOException {
        Regions clientRegion = Regions.DEFAULT_REGION;
        String bucketName = "*** Bucket name ***";

        try {
            AmazonS3 s3Client = AmazonS3ClientBuilder.standard()
                .withCredentials(new ProfileCredentialsProvider())
                .withRegion(clientRegion)
                .build();

            if (!s3Client.doesBucketExistV2(bucketName)) {
                // Because the CreateBucketRequest object doesn't specify a region, the
                // bucket is created in the region specified in the client.
                s3Client.createBucket(new CreateBucketRequest(bucketName));

                // Verify that the bucket was created by retrieving it and checking its
                // location.
                String bucketLocation = s3Client.getBucketLocation(new
                    GetBucketLocationRequest(bucketName));
                System.out.println("Bucket location: " + bucketLocation);
            }
        } catch (AmazonServiceException e) {
            // The call was transmitted successfully, but Amazon S3 couldn't process
            // it and returned an error response.
            e.printStackTrace();
        } catch (SdkClientException e) {
            // Amazon S3 couldn't be contacted for a response, or the client
            // couldn't parse the response from Amazon S3.
            e.printStackTrace();
        }
    }
}
```

.NET

For information about how to create and test a working sample, see [Running the Amazon S3 .NET Code Examples \(p. 1193\)](#).

Example

```
using Amazon;
using Amazon.S3;
using Amazon.S3.Model;
using Amazon.S3.Util;
using System;
using System.Threading.Tasks;

namespace Amazon.DocSamples.S3
{
    class CreateBucketTest
    {
```

```
private const string bucketName = "*** bucket name ***";
// Specify your bucket region (an example region is shown).
private static readonly RegionEndpoint bucketRegion = RegionEndpoint.USWest2;
private static IAmazonS3 s3Client;
public static void Main()
{
    s3Client = new AmazonS3Client(bucketRegion);
    CreateBucketAsync().Wait();
}

static async Task CreateBucketAsync()
{
    try
    {
        if (!(await AmazonS3Util.DoesS3BucketExistAsync(s3Client, bucketName)))
        {
            var putBucketRequest = new PutBucketRequest
            {
                BucketName = bucketName,
                UseClientRegion = true
            };

            PutBucketResponse putBucketResponse = await
s3Client.PutBucketAsync(putBucketRequest);
        }
        // Retrieve the bucket location.
        string bucketLocation = await FindBucketLocationAsync(s3Client);
    }
    catch (AmazonS3Exception e)
    {
        Console.WriteLine("Error encountered on server. Message:'{0}' when
writing an object", e.Message);
    }
    catch (Exception e)
    {
        Console.WriteLine("Unknown encountered on server. Message:'{0}' when
writing an object", e.Message);
    }
}
static async Task<string> FindBucketLocationAsync(IAmazonS3 client)
{
    string bucketLocation;
    var request = new GetBucketLocationRequest()
    {
        BucketName = bucketName
    };
    GetBucketLocationResponse response = await
client.GetBucketLocationAsync(request);
    bucketLocation = response.Location.ToString();
    return bucketLocation;
}
```

Ruby

For information about how to create and test a working sample, see [Using the AWS SDK for Ruby - Version 3 \(p. 1194\)](#).

Example

```
require "aws-sdk-s3"

# Wraps Amazon S3 bucket actions.
```

```
class BucketCreateWrapper
  attr_reader :bucket

  # @param bucket [Aws::S3::Bucket] An Amazon S3 bucket initialized with a name. This
  # is a client-side object until
  #                                         create is called.
  #
  def initialize(bucket)
    @bucket = bucket
  end

  # Creates an Amazon S3 bucket in the specified AWS Region.
  #
  # @param region [String] The Region where the bucket is created.
  # @return [Boolean] True when the bucket is created; otherwise, false.
  def create?(region)
    @bucket.create(create_bucket_configuration: { location_constraint: region })
    true
  rescue Aws::Errors::ServiceError => e
    puts "Couldn't create bucket. Here's why: #{e.message}"
    false
  end

  # Gets the Region where the bucket is located.
  #
  # @return [String] The location of the bucket.
  def location
    if @bucket.nil?
      "None. You must create a bucket before you can get it's location!"
    else
      @bucket.client.get_bucket_location(bucket: @bucket.name).location_constraint
    end
  rescue Aws::Errors::ServiceError => e
    "Couldn't get the location of #{@bucket.name}. Here's why: #{e.message}"
  end
end

def run_demo
  region = "us-west-2"
  wrapper = BucketCreateWrapper.new(Aws::S3::Bucket.new("doc-example-bucket-#{Random.uuid}"))
  return unless wrapper.create?(region)

  puts "Created bucket #{wrapper.bucket.name}."
  puts "Your bucket's region is: #{wrapper.location}"
end

run_demo if $PROGRAM_NAME == __FILE__
```

Using the AWS CLI

You can also use the AWS Command Line Interface (AWS CLI) to create an S3 bucket. For more information, see [create-bucket](#) in the *AWS CLI Command Reference*.

For information about the AWS CLI, see [What is the AWS Command Line Interface?](#) in the *AWS Command Line Interface User Guide*.

Viewing the properties for an S3 bucket

You can view and configure the properties for an Amazon S3 bucket, including settings for versioning, tags, default encryption, logging, notifications, and more.

To view the properties for an S3 bucket

1. Sign in to the AWS Management Console and open the Amazon S3 console at <https://console.aws.amazon.com/s3/>.
2. In the **Buckets** list, choose the name of the bucket that you want to view the properties for.
3. Choose **Properties**.
4. On the **Properties** page, you can configure the following properties for the bucket.
 - **Bucket Versioning** – Keep multiple versions of an object in one bucket by using versioning. By default, versioning is disabled for a new bucket. For information about enabling versioning, see [Enabling versioning on buckets \(p. 643\)](#).
 - **Tags** – With AWS cost allocation, you can use bucket tags to annotate billing for your use of a bucket. A tag is a key-value pair that represents a label that you assign to a bucket. To add tags, choose **Tags**, and then choose **Add tag**. For more information, see [Using cost allocation S3 bucket tags \(p. 834\)](#).
 - **Default encryption** – Enabling default encryption provides you with automatic server-side encryption. Amazon S3 encrypts an object before saving it to a disk and decrypts the object when you download it. For more information, see [Setting default server-side encryption behavior for Amazon S3 buckets \(p. 132\)](#).
 - **Server access logging** – Get detailed records for the requests that are made to your bucket with server access logging. By default, Amazon S3 doesn't collect server access logs. For information about enabling server access logging, see [Enabling Amazon S3 server access logging \(p. 980\)](#).
 - **AWS CloudTrail data events** – Use CloudTrail to log data events. By default, trails don't log data events. Additional charges apply for data events. For more information, see [Logging Data Events for Trails](#) in the *AWS CloudTrail User Guide*.
 - **Event notifications** – Enable certain Amazon S3 bucket events to send notification messages to a destination whenever the events occur. To enable events, choose **Create event notification**, and then specify the settings you want to use. For more information, see [Enabling and configuring event notifications using the Amazon S3 console \(p. 1025\)](#).
 - **Transfer acceleration** – Enable fast, easy, and secure transfers of files over long distances between your client and an S3 bucket. For information about enabling transfer acceleration, see [Enabling and using S3 Transfer Acceleration \(p. 139\)](#).
 - **Object Lock** – Use S3 Object Lock to prevent an object from being deleted or overwritten for a fixed amount of time or indefinitely. For more information, see [Using S3 Object Lock \(p. 680\)](#).
 - **Requester Pays** – Enable Requester Pays if you want the requester (instead of the bucket owner) to pay for requests and data transfers. For more information, see [Using Requester Pays buckets for storage transfers and usage \(p. 144\)](#).
 - **Static website hosting** – You can host a static website on Amazon S3. To enable static website hosting, choose **Static website hosting**, and then specify the settings you want to use. For more information, see [Hosting a static website using Amazon S3 \(p. 1116\)](#).

Methods for accessing a bucket

You can access your bucket using the Amazon S3 console. Using the console UI, you can perform almost all bucket operations without having to write any code.

If you access a bucket programmatically, Amazon S3 supports RESTful architecture in which your buckets and objects are resources, each with a resource URI that uniquely identifies the resource.

Amazon S3 supports both virtual-hosted-style and path-style URLs to access a bucket. Because buckets can be accessed using path-style and virtual-hosted-style URLs, we recommend that you create buckets with DNS-compliant bucket names. For more information, see [Bucket restrictions and limitations \(p. 147\)](#).

Note

Virtual-hosted-style and path-style requests use the S3 dot Region endpoint structure (`s3.Region`), for example, `https://my-bucket.s3.us-west-2.amazonaws.com`. However, some older Amazon S3 Regions also support S3 dash Region endpoints `s3-Region`, for example, `https://my-bucket.s3-us-west-2.amazonaws.com`. If your bucket is in one of these Regions, you might see `s3-Region` endpoints in your server access logs or AWS CloudTrail logs. We recommend that you do not use this endpoint structure in your requests.

Virtual-hosted-style access

In a virtual-hosted-style request, the bucket name is part of the domain name in the URL.

Amazon S3 virtual-hosted-style URLs use the following format:

```
https://bucket-name.s3.region-code.amazonaws.com/key-name
```

In this example, `DOC-EXAMPLE-BUCKET1` is the bucket name, US West (Oregon) is the Region, and `puppy.png` is the key name:

```
https://DOC-EXAMPLE-BUCKET1.s3.us-west-2.amazonaws.com/puppy.png
```

For more information about virtual hosted style access, see [Virtual-hosted-style requests \(p. 1176\)](#).

Path-style access

In Amazon S3, path-style URLs use the following format:

```
https://s3.region-code.amazonaws.com/bucket-name/key-name
```

For example, if you create a bucket named `DOC-EXAMPLE-BUCKET1` in the US West (Oregon) Region, and you want to access the `puppy.jpg` object in that bucket, you can use the following path-style URL:

```
https://s3.us-west-2.amazonaws.com/DOC-EXAMPLE-BUCKET1/puppy.jpg
```

For more information, see [Path-style requests \(p. 1176\)](#).

Important

Update (September 23, 2020) – To make sure that customers have the time that they need to transition to virtual-hosted-style URLs, we have decided to delay the deprecation of path-style URLs. For more information, see [Amazon S3 Path Deprecation Plan – The Rest of the Story](#) in the [AWS News Blog](#).

Accessing an S3 bucket over IPv6

Amazon S3 has a set of dual-stack endpoints, which support requests to S3 buckets over both Internet Protocol version 6 (IPv6) and IPv4. For more information, see [Making requests over IPv6 \(p. 1140\)](#).

Accessing a bucket through S3 access points

In addition to accessing a bucket directly, you can access a bucket through an access point. For more information about the S3 access points feature, see [Managing data access with Amazon S3 access points \(p. 301\)](#).

S3 access points only support virtual-host-style addressing. To address a bucket through an access point, use the following format.

```
https://AccessPointName-AccountID.s3-accesspoint.region.amazonaws.com.
```

Note

- If your access point name includes dash (-) characters, include the dashes in the URL and insert another dash before the account ID. For example, to use an access point named finance-docs owned by account 123456789012 in Region us-west-2, the appropriate URL would be `https://finance-docs-123456789012.s3-accesspoint.us-west-2.amazonaws.com`.
- S3 access points don't support access by HTTP, only secure access by HTTPS.

Accessing a bucket using S3://

Some AWS services require specifying an Amazon S3 bucket using `s3://bucket`. The following example shows the correct format. Be aware that when using this format, the bucket name does not include the AWS Region.

```
s3://bucket-name/key-name
```

For example, the following example uses the sample bucket described in the earlier path-style section.

```
s3://mybucket/puppy.jpg
```

Emptying a bucket

You can empty a bucket's contents using the Amazon S3 console, AWS SDKs, or AWS Command Line Interface (AWS CLI). When you empty a bucket, you delete all the objects, but you keep the bucket. After you empty a bucket, it cannot be undone. When you empty a bucket that has S3 Bucket Versioning enabled or suspended, all versions of all the objects in the bucket are deleted. For more information, see [Working with objects in a versioning-enabled bucket \(p. 649\)](#).

You can also specify a lifecycle configuration on a bucket to expire objects so that Amazon S3 can delete them. For more information, see [Setting lifecycle configuration on a bucket \(p. 708\)](#)

Troubleshooting

Objects added to the bucket while the empty bucket action is in progress might be deleted. To prevent new objects from being added to a bucket while the empty bucket action is in progress, you might need to stop your AWS CloudTrail trails from logging events to the bucket. For more information, see [Turning off logging for a trail in the AWS CloudTrail User Guide](#).

Another alternative to stopping CloudTrail trails from being added to the bucket is to add a deny `s3:PutObject` statement to your bucket policy. If you want to store new objects in the bucket, you should remove the deny `s3:PutObject` statement from your bucket policy. For more information, see [Example — Object operations \(p. 415\)](#) and [IAM JSON policy elements: Effect](#) in the [IAM User Guide](#)

Using the S3 console

You can use the Amazon S3 console to empty a bucket, which deletes all of the objects in the bucket without deleting the bucket.

To empty an S3 bucket

1. Sign in to the AWS Management Console and open the Amazon S3 console at <https://console.aws.amazon.com/s3/>.
2. In the **Bucket name** list, select the option next to the name of the bucket that you want to empty, and then choose **Empty**.
3. On the **Empty bucket** page, confirm that you want to empty the bucket by entering the bucket name into the text field, and then choose **Empty**.
4. Monitor the progress of the bucket emptying process on the **Empty bucket: Status** page.

Using the AWS CLI

You can empty a bucket using the AWS CLI only if the bucket does not have Bucket Versioning enabled. If versioning is not enabled, you can use the `rm` (remove) AWS CLI command with the `--recursive` parameter to empty the bucket (or remove a subset of objects with a specific key name prefix).

The following `rm` command removes objects that have the key name prefix `doc`, for example, `doc/doc1` and `doc/doc2`.

```
$ aws s3 rm s3://bucket-name/doc --recursive
```

Use the following command to remove all objects without specifying a prefix.

```
$ aws s3 rm s3://bucket-name --recursive
```

For more information, see [Using high-level S3 commands with the AWS CLI](#) in the *AWS Command Line Interface User Guide*.

Note

You can't remove objects from a bucket that has versioning enabled. Amazon S3 adds a delete marker when you delete an object, which is what this command does. For more information about S3 Bucket Versioning, see [Using versioning in S3 buckets \(p. 638\)](#).

Using the AWS SDKs

You can use the AWS SDKs to empty a bucket or remove a subset of objects that have a specific key name prefix.

For an example of how to empty a bucket using AWS SDK for Java, see [Deleting a bucket \(p. 129\)](#). The code deletes all objects, regardless of whether the bucket has versioning enabled, and then it deletes the bucket. To just empty the bucket, make sure that you remove the statement that deletes the bucket.

For more information about using other AWS SDKs, see [Tools for Amazon Web Services](#).

Using a lifecycle configuration

If you use a lifecycle policy to empty your bucket, the lifecycle policy should include [current versions](#), [non-current versions](#), [delete markers](#), and [incomplete multipart uploads](#).

You can add lifecycle configuration rules to expire all objects or a subset of objects that have a specific key name prefix. For example, to remove all objects in a bucket, you can set a lifecycle rule to expire objects one day after creation.

Amazon S3 supports a bucket lifecycle rule that you can use to stop multipart uploads that don't complete within a specified number of days after being initiated. We recommend that you configure this lifecycle rule to minimize your storage costs. For more information, see [Configuring a bucket lifecycle policy to abort incomplete multipart uploads \(p. 172\)](#).

For more information about using a lifecycle configuration to empty a bucket, see [Setting lifecycle configuration on a bucket \(p. 708\)](#) and [Expiring objects \(p. 707\)](#).

Deleting a bucket

You can delete an empty Amazon S3 bucket. Before deleting a bucket, consider the following:

- Bucket names are unique. If you delete a bucket, another AWS user can use the name.
- If the bucket hosts a static website, and you created and configured an Amazon Route 53 hosted zone as described in [Configuring a static website using a custom domain registered with Route 53 \(p. 97\)](#), you must clean up the Route 53 hosted zone settings that are related to the bucket. For more information, see [Step 2: Delete the Route 53 hosted zone \(p. 112\)](#).
- If the bucket receives log data from Elastic Load Balancing (ELB): We recommend that you stop the delivery of ELB logs to the bucket before deleting it. After you delete the bucket, if another user creates a bucket using the same name, your log data could potentially be delivered to that bucket. For information about ELB access logs, see [Access logs in the User Guide for Classic Load Balancers](#) and [Access logs in the User Guide for Application Load Balancers](#).

Troubleshooting

If you are unable to delete an Amazon S3 bucket, consider the following:

- **Make sure the bucket is empty** – You can only delete buckets that don't have any objects in them. Make sure the bucket is empty.
- **s3:DeleteBucket permissions** – If you cannot delete a bucket, work with your IAM administrator to confirm that you have s3:DeleteBucket permissions in your IAM user policy. For information about how to view or update IAM permissions, see [Changing permissions for an IAM user](#) in the *IAM User Guide*.
- **s3:DeleteBucket deny statement** – If you have s3:DeleteBucket permissions in your IAM policy and you cannot delete a bucket, the bucket policy might include a deny statement for s3:DeleteBucket. Buckets created by ElasticBeanstalk have a policy containing this statement by default. Before you can delete the bucket, you must delete this statement or the bucket policy.

Important

Bucket names are unique. If you delete a bucket, another AWS user can use the name. If you want to continue to use the same bucket name, don't delete the bucket. We recommend that you empty the bucket and keep it.

Using the S3 console

To delete an S3 bucket

1. Sign in to the AWS Management Console and open the Amazon S3 console at <https://console.aws.amazon.com/s3/>.
2. In the **Buckets** list, select the option next to the name of the bucket that you want to delete, and then choose **Delete** at the top of the page.
3. On the **Delete bucket** page, confirm that you want to delete the bucket by entering the bucket name into the text field, and then choose **Delete bucket**.

Note

If the bucket contains any objects, empty the bucket before deleting it by selecting the *empty bucket configuration* link in the **This bucket is not empty** error alert and following

the instructions on the **Empty bucket** page. Then return to the **Delete bucket** page and delete the bucket.

4. To verify that you've deleted the bucket, open the **Buckets** list and enter the name of the bucket that you deleted. If the bucket can't be found, your deletion was successful.

Using the AWS SDK for Java

The following example shows you how to delete a bucket using the AWS SDK for Java. First, the code deletes objects in the bucket and then it deletes the bucket. For information about other AWS SDKs, see [Tools for Amazon Web Services](#).

Java

The following Java example deletes a bucket that contains objects. The example deletes all objects, and then it deletes the bucket. The example works for buckets with or without versioning enabled.

Note

For buckets without versioning enabled, you can delete all objects directly and then delete the bucket. For buckets with versioning enabled, you must delete all object versions before deleting the bucket.

For instructions on creating and testing a working sample, see [Testing the Amazon S3 Java Code Examples \(p. 1192\)](#).

```
import com.amazonaws.AmazonServiceException;
import com.amazonaws.SdkClientException;
import com.amazonaws.auth.profile.ProfileCredentialsProvider;
import com.amazonaws.regions.Regions;
import com.amazonaws.services.s3.AmazonS3;
import com.amazonaws.services.s3.AmazonS3ClientBuilder;
import com.amazonaws.services.s3.model.*;

import java.util.Iterator;

public class DeleteBucket2 {

    public static void main(String[] args) {
        Regions clientRegion = Regions.DEFAULT_REGION;
        String bucketName = "*** Bucket name ***";

        try {
            AmazonS3 s3Client = AmazonS3ClientBuilder.standard()
                .withCredentials(new ProfileCredentialsProvider())
                .withRegion(clientRegion)
                .build();

            // Delete all objects from the bucket. This is sufficient
            // for unversioned buckets. For versioned buckets, when you attempt to
            delete objects, Amazon S3 inserts
            // delete markers for all objects, but doesn't delete the object versions.
            // To delete objects from versioned buckets, delete all of the object
            versions before deleting
            // the bucket (see below for an example).
            ObjectListing objectListing = s3Client.listObjects(bucketName);
            while (true) {
                Iterator<S3ObjectSummary> objIter =
objectListing.getObjectSummaries().iterator();
                while (objIter.hasNext()) {
                    s3Client.deleteObject(bucketName, objIter.next().getKey());
                }
            }
        } catch (AmazonServiceException e) {
            System.out.println("Caught an AmazonServiceException, which " +
"means your request made it to Amazon S3, but was rejected with an error response for some reason.");
            System.out.println("Error Message: " + e.getMessage());
        } catch (SdkClientException e) {
            System.out.println("Caught an SdkClientException, which means " +
"something bad happened on the wire while attempting to " +
"communicate with Amazon S3, such as a timeout or connection " +
"failure. Error Message: " + e.getMessage());
        }
    }
}
```

```
// If the bucket contains many objects, the listObjects() call
// might not return all of the objects in the first listing. Check to
// see whether the listing was truncated. If so, retrieve the next page
of objects
    // and delete them.
    if (objectListing.isTruncated()) {
        objectListing = s3Client.listNextBatchOfObjects(objectListing);
    } else {
        break;
    }
}

// Delete all object versions (required for versioned buckets).
VersionListing versionList = s3Client.listVersions(new
ListVersionsRequest().withBucketName(bucketName));
while (true) {
    Iterator<S3VersionSummary> versionIter =
versionList.getVersionSummaries().iterator();
    while (versionIter.hasNext()) {
        S3VersionSummary vs = versionIter.next();
        s3Client.deleteVersion(bucketName, vs.getKey(), vs.getVersionId());
    }

    if (versionList.isTruncated()) {
        versionList = s3Client.listNextBatchOfVersions(versionList);
    } else {
        break;
    }
}

// After all objects and object versions are deleted, delete the bucket.
s3Client.deleteBucket(bucketName);
} catch (AmazonServiceException e) {
    // The call was transmitted successfully, but Amazon S3 couldn't process
    // it, so it returned an error response.
    e.printStackTrace();
} catch (SdkClientException e) {
    // Amazon S3 couldn't be contacted for a response, or the client couldn't
    // parse the response from Amazon S3.
    e.printStackTrace();
}
}
```

Using the AWS CLI

You can delete a bucket that contains objects with the AWS CLI if it doesn't have versioning enabled. When you delete a bucket that contains objects, all the objects in the bucket are permanently deleted, including objects that are transitioned to the `S3 Glacier` storage class.

If your bucket does not have versioning enabled, you can use the `rb` (remove bucket) AWS CLI command with the `--force` parameter to delete the bucket and all the objects in it. This command deletes all objects first and then deletes the bucket.

If versioning is enabled versioned objects will not be deleted in this process which would cause the bucket deletion to fail because the bucket would not be empty. For more information about deleting versioned objects, see [Deleting object versions](#).

```
$ aws s3 rb s3://bucket-name --force
```

For more information, see [Using High-Level S3 Commands with the AWS Command Line Interface](#) in the AWS Command Line Interface User Guide.

Setting default server-side encryption behavior for Amazon S3 buckets

With Amazon S3 default encryption, you can set the default encryption behavior for an S3 bucket so that all new objects are encrypted when they are stored in the bucket. The objects are encrypted using server-side encryption with either Amazon S3-managed keys (SSE-S3) or AWS KMS keys stored in AWS Key Management Service (AWS KMS) (SSE-KMS).

When you configure your bucket to use default encryption with SSE-KMS, you can also enable S3 Bucket Keys to decrease request traffic from Amazon S3 to AWS Key Management Service (AWS KMS) and reduce the cost of encryption. For more information, see [Reducing the cost of SSE-KMS with Amazon S3 Bucket Keys \(p. 347\)](#).

When you use server-side encryption, Amazon S3 encrypts an object before saving it to disk and decrypts it when you download the objects. For more information about protecting data using server-side encryption and encryption key management, see [Protecting data using server-side encryption \(p. 338\)](#).

For more information about permissions required for default encryption, see [PutBucketEncryption](#) in the [Amazon Simple Storage Service API Reference](#).

To set up default encryption on a bucket, you can use the Amazon S3 console, AWS CLI, AWS SDKs, or the REST API. For more information, see [the section called “Enabling default encryption” \(p. 133\)](#).

Encrypting existing objects

To encrypt your existing Amazon S3 objects, you can use Amazon S3 Batch Operations. You provide S3 Batch Operations with a list of objects to operate on, and Batch Operations calls the respective API to perform the specified operation. You can use the [Batch Operations Copy operation](#) to copy existing unencrypted objects and write them back to the same bucket as encrypted objects. A single Batch Operations job can perform the specified operation on billions of objects. For more information, see [Performing large-scale batch operations on Amazon S3 objects \(p. 881\)](#) and the AWS Storage Blog post [Encrypting objects with Amazon S3 Batch Operations](#).

You can also encrypt existing objects using the Copy Object API. For more information, see the AWS Storage Blog post [Encrypting existing Amazon S3 objects with the AWS CLI](#).

Note

Amazon S3 buckets with default bucket encryption using SSE-KMS cannot be used as destination buckets for [the section called “Logging server access” \(p. 978\)](#). Only SSE-S3 default encryption is supported for server access log destination buckets.

Using encryption for cross-account operations

Be aware of the following when using encryption for cross-account operations:

- The AWS managed key (aws/s3) is used when a AWS KMS key Amazon Resource Name (ARN) or alias is not provided at request time, nor via the bucket's default encryption configuration.
- If you're uploading or accessing S3 objects using AWS Identity and Access Management (IAM) principals that are in the same AWS account as your KMS key, you can use the AWS managed key (aws/s3).
- Use a customer managed key if you want to grant cross-account access to your S3 objects. You can configure the policy of a customer managed key to allow access from another account.

- If specifying your own KMS key, you should use a fully qualified KMS key ARN. When using a KMS key alias, be aware that AWS KMS will resolve the key within the requester's account. This can result in data encrypted with a KMS key that belongs to the requester, and not the bucket administrator.
- You must specify a key that you (the requester) have been granted `Encrypt` permission to. For more information, see [Allows key users to use a KMS key for cryptographic operations in the AWS Key Management Service Developer Guide](#).

For more information about when to use customer managed keys and the AWS managed KMS keys, see [Should I use an AWS managed key or a customer managed KMS key to encrypt my objects on Amazon S3?](#)

Using default encryption with replication

When you enable default encryption for a replication destination bucket, the following encryption behavior applies:

- If objects in the source bucket are not encrypted, the replica objects in the destination bucket are encrypted using the default encryption settings of the destination bucket. This results in the `ETag` of the source object being different from the `ETag` of the replica object. You must update applications that use the `ETag` to accommodate for this difference.
- If objects in the source bucket are encrypted using SSE-S3 or SSE-KMS, the replica objects in the destination bucket use the same encryption as the source object encryption. The default encryption settings of the destination bucket are not used.

For more information about using default encryption with SSE-KMS, see [Replicating encrypted objects \(p. 814\)](#).

Using Amazon S3 Bucket Keys with default encryption

When you configure your bucket to use default encryption for SSE-KMS on new objects, you can also configure S3 Bucket Keys. S3 Bucket Keys decrease the number of transactions from Amazon S3 to AWS KMS to reduce the cost of server-side encryption using AWS Key Management Service (SSE-KMS).

When you configure your bucket to use S3 Bucket Keys for SSE-KMS on new objects, AWS KMS generates a bucket-level key that is used to create a unique `data key` for objects in the bucket. This bucket key is used for a time-limited period within Amazon S3, reducing the need for Amazon S3 to make requests to AWS KMS to complete encryption operations.

For more information about using an S3 Bucket Key, see [Using Amazon S3 Bucket Keys \(p. 347\)](#).

Enabling Amazon S3 default bucket encryption

You can set the default encryption behavior on an Amazon S3 bucket so that all objects are encrypted when they are stored in the bucket. The objects are encrypted using server-side encryption with either Amazon S3-managed keys (SSE-S3) or AWS Key Management Service (AWS KMS) keys.

When you configure default encryption using AWS KMS, you can also configure S3 Bucket Key. For more information, see [Reducing the cost of SSE-KMS with Amazon S3 Bucket Keys \(p. 347\)](#).

Default encryption works with all existing and new Amazon S3 buckets. Without default encryption, to encrypt all objects stored in a bucket, you must include encryption information with every object storage request. You must also set up an Amazon S3 bucket policy to reject storage requests that don't include encryption information.

There are no additional charges for using default encryption for S3 buckets. Requests to configure the default encryption feature incur standard Amazon S3 request charges. For information about pricing, see [Amazon S3 pricing](#). For SSE-KMS KMS key storage, AWS KMS charges apply and are listed at [AWS KMS pricing](#).

Changes to note before enabling default encryption

After you enable default encryption for a bucket, the following encryption behavior applies:

- There is no change to the encryption of the objects that existed in the bucket before default encryption was enabled.
- When you upload objects after enabling default encryption:
 - If your `PUT` request headers don't include encryption information, Amazon S3 uses the bucket's default encryption settings to encrypt the objects.
 - If your `PUT` request headers include encryption information, Amazon S3 uses the encryption information from the `PUT` request to encrypt objects before storing them in Amazon S3.
- If you use the SSE-KMS option for your default encryption configuration, you are subject to the RPS (requests per second) limits of AWS KMS. For more information about AWS KMS limits and how to request a limit increase, see [AWS KMS limits](#).

Using the S3 console

To enable default encryption on an Amazon S3 bucket

1. Sign in to the AWS Management Console and open the Amazon S3 console at <https://console.aws.amazon.com/s3/>.
2. In the **Buckets** list, choose the name of the bucket that you want.
3. Choose **Properties**.
4. Under **Default encryption**, choose **Edit**.
5. To enable or disable server-side encryption, choose **Enable or Disable**.
6. To enable server-side encryption using an Amazon S3-managed key, under **Encryption key type**, choose **Amazon S3 key (SSE-S3)**.

For more information about using Amazon S3 server-side encryption to encrypt your data, see [Protecting data using server-side encryption with Amazon S3-managed encryption keys \(SSE-S3\) \(p. 355\)](#).

7. To enable server-side encryption using an AWS KMS key, follow these steps:

- a. Under **Encryption key type**, choose **AWS Key Management Service key (SSE-KMS)**.

Important

If you use the AWS KMS option for your default encryption configuration, you are subject to the RPS (requests per second) limits of AWS KMS. For more information about AWS KMS quotas and how to request a quota increase, see [Quotas](#).

- b. Under **AWS KMS key** choose one of the following:

- **AWS managed key (aws/s3)**
- **Choose from your KMS root keys**, and choose your **KMS root key**.
- **Enter KMS root key ARN**, and enter your AWS KMS key ARN.

Important

You can only use KMS keys that are enabled in the same AWS Region as the bucket. When you choose **Choose from your KMS keys**, the S3 console only lists 100 KMS keys per Region. If you have more than 100 KMS keys in the same Region, you can only

see the first 100 KMS keys in the S3 console. To use a KMS key that is not listed in the console, choose **Custom KMS ARN**, and enter the KMS key ARN.

When you use an AWS KMS key for server-side encryption in Amazon S3, you must choose a symmetric encryption KMS key. Amazon S3 only supports symmetric encryption KMS keys and not asymmetric KMS keys. For more information, see [Using symmetric and asymmetric keys in the AWS Key Management Service Developer Guide](#).

For more information about creating an AWS KMS key, see [Creating keys in the AWS Key Management Service Developer Guide](#). For more information about using AWS KMS with Amazon S3, see [Protecting data using server-side encryption with AWS Key Management Service \(SSE-KMS\) \(p. 338\)](#).

8. To use S3 Bucket Keys, under **Bucket Key**, choose **Enable**.

When you configure your bucket to use default encryption with SSE-KMS, you can also enable S3 Bucket Key. S3 Bucket Keys decrease request traffic from Amazon S3 to AWS KMS and lower the cost of encryption. For more information, see [Reducing the cost of SSE-KMS with Amazon S3 Bucket Keys \(p. 347\)](#).

9. Choose **Save changes**.

Using the AWS CLI

These examples show you how to configure default encryption using Amazon S3-managed encryption (SSE-S3) or AWS KMS encryption (SSE-KMS) with an S3 Bucket Key.

For more information about default encryption, see [Setting default server-side encryption behavior for Amazon S3 buckets \(p. 132\)](#). For more information about using the AWS CLI to configure default encryption, see [put-bucket-encryption](#).

Example – Default encryption with SSE-S3

This example configures default bucket encryption with Amazon S3-managed encryption.

```
aws s3api put-bucket-encryption --bucket bucket-name --server-side-encryption-configuration
'{
    "Rules": [
        {
            "ApplyServerSideEncryptionByDefault": {
                "SSEAlgorithm": "AES256"
            }
        }
    ]
}'
```

Example – Default encryption with SSE-KMS using an S3 Bucket Key

This example configures default bucket encryption with SSE-KMS using an S3 Bucket Key.

```
aws s3api put-bucket-encryption --bucket bucket-name --server-side-encryption-configuration
'{
    "Rules": [
        {
            "ApplyServerSideEncryptionByDefault": {
                "SSEAlgorithm": "aws:kms",
                "KMSMasterKeyID": "KMS-Key-ARN"
            },
            "BucketKeyEnabled": true
        }
    ]
}'
```

Using the REST API

Use the REST API PUT Bucket encryption operation to enable default encryption and to set the type of server-side encryption to use—SSE-S3 or SSE-KMS.

For more information, see [PutBucketEncryption](#) in the *Amazon Simple Storage Service API Reference*.

Monitoring default encryption with CloudTrail and CloudWatch

You can track default encryption configuration requests for Amazon S3 buckets using AWS CloudTrail events. The following API event names are used in CloudTrail logs:

- `PutBucketEncryption`
- `GetBucketEncryption`
- `DeleteBucketEncryption`

You can also create Amazon CloudWatch Events with S3 bucket-level operations as the event type. For more information about CloudTrail events, see [Enable logging for objects in a bucket using the console \(p. 971\)](#).

You can use CloudTrail logs for object-level Amazon S3 actions to track `PUT` and `POST` requests to Amazon S3. You can use these actions to verify whether default encryption is being used to encrypt objects when incoming `PUT` requests don't have encryption headers.

When Amazon S3 encrypts an object using the default encryption settings, the log includes the following field as the name/value pair: `"SSEApplied": "Default_SSE_S3"` or `"SSEApplied": "Default_SSE_KMS"`.

When Amazon S3 encrypts an object using the `PUT` encryption headers, the log includes one of the following fields as the name/value pair: `"SSEApplied": "SSE_S3"`, `"SSEApplied": "SSE_KMS"` or `"SSEApplied": "SSE_C"`.

For multipart uploads, this information is included in the `InitiateMultipartUpload` API requests. For more information about using CloudTrail and CloudWatch, see [Monitoring Amazon S3 \(p. 959\)](#).

Configuring fast, secure file transfers using Amazon S3 Transfer Acceleration

Amazon S3 Transfer Acceleration is a bucket-level feature that enables fast, easy, and secure transfers of files over long distances between your client and an S3 bucket. Transfer Acceleration is designed to optimize transfer speeds from across the world into S3 buckets. Transfer Acceleration takes advantage of the globally distributed edge locations in Amazon CloudFront. As the data arrives at an edge location, the data is routed to Amazon S3 over an optimized network path.

When you use Transfer Acceleration, additional data transfer charges might apply. For more information about pricing, see [Amazon S3 pricing](#).

Why use Transfer Acceleration?

You might want to use Transfer Acceleration on a bucket for various reasons:

- Your customers upload to a centralized bucket from all over the world.

- You transfer gigabytes to terabytes of data on a regular basis across continents.
- You can't use all of your available bandwidth over the internet when uploading to Amazon S3.

For more information about when to use Transfer Acceleration, see [Amazon S3 FAQs](#).

Requirements for using Transfer Acceleration

The following are required when you are using Transfer Acceleration on an S3 bucket:

- Transfer Acceleration is only supported on virtual-hosted style requests. For more information about virtual-hosted style requests, see [Making requests using the REST API \(p. 1174\)](#).
- The name of the bucket used for Transfer Acceleration must be DNS-compliant and must not contain periods (".").
- Transfer Acceleration must be enabled on the bucket. For more information, see [Enabling and using S3 Transfer Acceleration \(p. 139\)](#).

After you enable Transfer Acceleration on a bucket, it might take up to 20 minutes before the data transfer speed to the bucket increases.

Note

Transfer Acceleration is currently not supported for buckets located in the following Regions:

- Africa (Cape Town) (af-south-1)
- Asia Pacific (Hong Kong) (ap-east-1)
- Asia Pacific (Osaka) (ap-northeast-3)
- Europe (Stockholm) (eu-north-1)
- Europe (Milan) (eu-south-1)
- Middle East (Bahrain) (me-south-1)
- To access the bucket that is enabled for Transfer Acceleration, you must use the endpoint `bucketname.s3-accelerate.amazonaws.com`. Or, use the dual-stack endpoint `bucketname.s3-accelerate.dualstack.amazonaws.com` to connect to the enabled bucket over IPv6.
- You must be the bucket owner to set the transfer acceleration state. The bucket owner can assign permissions to other users to allow them to set the acceleration state on a bucket. The `s3:PutAccelerateConfiguration` permission permits users to enable or disable Transfer Acceleration on a bucket. The `s3:GetAccelerateConfiguration` permission permits users to return the Transfer Acceleration state of a bucket, which is either `Enabled` or `Suspended`. For more information about these permissions, see [Example — Bucket subresource operations \(p. 416\)](#) and [Identity and access management in Amazon S3 \(p. 394\)](#).

The following sections describe how to get started and use Amazon S3 Transfer Acceleration for transferring data.

Topics

- [Getting started with Amazon S3 Transfer Acceleration \(p. 137\)](#)
- [Enabling and using S3 Transfer Acceleration \(p. 139\)](#)
- [Using the Amazon S3 Transfer Acceleration Speed Comparison tool \(p. 143\)](#)

Getting started with Amazon S3 Transfer Acceleration

You can use Amazon S3 Transfer Acceleration for fast, easy, and secure transfers of files over long distances between your client and an S3 bucket. Transfer Acceleration uses the globally distributed edge

locations in Amazon CloudFront. As the data arrives at an edge location, data is routed to Amazon S3 over an optimized network path.

To get started using Amazon S3 Transfer Acceleration, perform the following steps:

1. Enable Transfer Acceleration on a bucket

You can enable Transfer Acceleration on a bucket any of the following ways:

- Use the Amazon S3 console.
- Use the REST API [PUT Bucket accelerate](#) operation.
- Use the AWS CLI and AWS SDKs. For more information, see [Developing with Amazon S3 using the AWS SDKs, and explorers \(p. 1184\)](#).

For more information, see [Enabling and using S3 Transfer Acceleration \(p. 139\)](#).

Note

For your bucket to work with transfer acceleration, the bucket name must conform to DNS naming requirements and must not contain periods (".").

2. Transfer data to and from the acceleration-enabled bucket

Use one of the following s3-accelerate endpoint domain names:

- To access an acceleration-enabled bucket, use `bucketname.s3-accelerate.amazonaws.com`.
- To access an acceleration-enabled bucket over IPv6, use `bucketname.s3-accelerate.dualstack.amazonaws.com`.

Amazon S3 dual-stack endpoints support requests to S3 buckets over IPv6 and IPv4. The Transfer Acceleration dual-stack endpoint only uses the virtual hosted-style type of endpoint name. For more information, see [Getting started making requests over IPv6 \(p. 1140\)](#) and [Using Amazon S3 dual-stack endpoints \(p. 1142\)](#).

Note

You can continue to use the regular endpoint in addition to the accelerate endpoints.

You can point your Amazon S3 PUT object and GET object requests to the s3-accelerate endpoint domain name after you enable Transfer Acceleration. For example, suppose that you currently have a REST API application using [PUT Object](#) that uses the hostname `mybucket.s3.us-east-1.amazonaws.com` in the `PUT` request. To accelerate the `PUT`, you change the hostname in your request to `mybucket.s3-accelerate.amazonaws.com`. To go back to using the standard upload speed, change the name back to `mybucket.s3.us-east-1.amazonaws.com`.

After Transfer Acceleration is enabled, it can take up to 20 minutes for you to realize the performance benefit. However, the accelerate endpoint is available as soon as you enable Transfer Acceleration.

You can use the accelerate endpoint in the AWS CLI, AWS SDKs, and other tools that transfer data to and from Amazon S3. If you are using the AWS SDKs, some of the supported languages use an accelerate endpoint client configuration flag so you don't need to explicitly set the endpoint for Transfer Acceleration to `bucketname.s3-accelerate.amazonaws.com`. For examples of how to use an accelerate endpoint client configuration flag, see [Enabling and using S3 Transfer Acceleration \(p. 139\)](#).

You can use all Amazon S3 operations through the transfer acceleration endpoints *except* for the following:

- [GET Service \(list buckets\)](#)
- [PUT Bucket \(create bucket\)](#)
- [DELETE Bucket](#)

Also, Amazon S3 Transfer Acceleration does not support cross-Region copies using [PUT Object - Copy](#).

Enabling and using S3 Transfer Acceleration

You can use Amazon S3 Transfer Acceleration transfer files quickly and securely over long distances between your client and an S3 bucket. You can enable Transfer Acceleration using the S3 console, the AWS Command Line Interface (AWS CLI), or the AWS SDKs.

This section provides examples of how to enable Amazon S3 Transfer Acceleration on a bucket and use the acceleration endpoint for the enabled bucket.

For more information about Transfer Acceleration requirements, see [Configuring fast, secure file transfers using Amazon S3 Transfer Acceleration \(p. 136\)](#).

Using the S3 console

Note

If you want to compare accelerated and non-accelerated upload speeds, open the [Amazon S3 Transfer Acceleration Speed Comparison tool](#).

The Speed Comparison tool uses multipart upload to transfer a file from your browser to various AWS Regions with and without Amazon S3 transfer acceleration. You can compare the upload speed for direct uploads and transfer accelerated uploads by Region.

To enable transfer acceleration for an S3 bucket

1. Sign in to the AWS Management Console and open the Amazon S3 console at <https://console.aws.amazon.com/s3/>.
2. In the **Buckets** list, choose the name of the bucket that you want to enable transfer acceleration for.
3. Choose **Properties**.
4. Under **Transfer acceleration**, choose **Edit**.
5. Choose **Enable**, and choose **Save changes**.

To access accelerated data transfers

1. After Amazon S3 enables transfer acceleration for your bucket, view the **Properties** tab for the bucket.
2. Under **Transfer acceleration**, **Accelerated endpoint** displays the transfer acceleration endpoint for your bucket. Use this endpoint to access accelerated data transfers to and from your bucket.

If you suspend transfer acceleration, the accelerate endpoint no longer works.

Using the AWS CLI

The following are examples of AWS CLI commands used for Transfer Acceleration. For instructions on setting up the AWS CLI, see [Developing with Amazon S3 using the AWS CLI \(p. 1183\)](#).

Enabling Transfer Acceleration on a bucket

Use the AWS CLI [put-bucket-accelerate-configuration](#) command to enable or suspend Transfer Acceleration on a bucket.

The following example sets `Status=Enabled` to enable Transfer Acceleration on a bucket. You use `Status=Suspended` to suspend Transfer Acceleration.

Example

```
$ aws s3api put-bucket-accelerate-configuration --bucket bucketname --accelerate-configuration Status=Enabled
```

Using Transfer Acceleration

You can direct all Amazon S3 requests made by s3 and s3api AWS CLI commands to the accelerate endpoint: s3-accelerate.amazonaws.com. To do this, set the configuration value use_accelerate_endpoint to true in a profile in your AWS Config file. Transfer Acceleration must be enabled on your bucket to use the accelerate endpoint.

All requests are sent using the virtual style of bucket addressing: my-bucket.s3-accelerate.amazonaws.com. Any ListBuckets, CreateBucket, and DeleteBucket requests are not sent to the accelerate endpoint because the endpoint doesn't support those operations.

For more information about use_accelerate_endpoint, see [AWS CLI S3 Configuration](#) in the *AWS CLI Command Reference*.

The following example sets use_accelerate_endpoint to true in the default profile.

Example

```
$ aws configure set default.s3.use_accelerate_endpoint true
```

If you want to use the accelerate endpoint for some AWS CLI commands but not others, you can use either one of the following two methods:

- Use the accelerate endpoint for any s3 or s3api command by setting the --endpoint-url parameter to https://s3-accelerate.amazonaws.com.
- Set up separate profiles in your AWS Config file. For example, create one profile that sets use_accelerate_endpoint to true and a profile that does not set use_accelerate_endpoint. When you run a command, specify which profile you want to use, depending upon whether you want to use the accelerate endpoint.

Uploading an object to a bucket enabled for Transfer Acceleration

The following example uploads a file to a bucket enabled for Transfer Acceleration by using the default profile that has been configured to use the accelerate endpoint.

Example

```
$ aws s3 cp file.txt s3://bucketname/keyword --region region
```

The following example uploads a file to a bucket enabled for Transfer Acceleration by using the --endpoint-url parameter to specify the accelerate endpoint.

Example

```
$ aws configure set s3.addressing_style virtual
$ aws s3 cp file.txt s3://bucketname/keyword --region region --endpoint-url https://s3-accelerate.amazonaws.com
```

Using the AWS SDKs

The following are examples of using Transfer Acceleration to upload objects to Amazon S3 using the AWS SDK. Some of the AWS SDK supported languages (for example, Java and .NET) use an

accelerate endpoint client configuration flag so you don't need to explicitly set the endpoint for Transfer Acceleration to `bucketname.s3-accelerate.amazonaws.com`.

Java

Example

The following example shows how to use an accelerate endpoint to upload an object to Amazon S3. The example does the following:

- Creates an `AmazonS3Client` that is configured to use accelerate endpoints. All buckets that the client accesses must have Transfer Acceleration enabled.
- Enables Transfer Acceleration on a specified bucket. This step is necessary only if the bucket you specify doesn't already have Transfer Acceleration enabled.
- Verifies that transfer acceleration is enabled for the specified bucket.
- Uploads a new object to the specified bucket using the bucket's accelerate endpoint.

For more information about using Transfer Acceleration, see [Getting started with Amazon S3 Transfer Acceleration \(p. 137\)](#). For instructions on creating and testing a working sample, see [Testing the Amazon S3 Java Code Examples \(p. 1192\)](#).

```
import com.amazonaws.AmazonServiceException;
import com.amazonaws.SdkClientException;
import com.amazonaws.auth.profile.ProfileCredentialsProvider;
import com.amazonaws.regions.Regions;
import com.amazonaws.services.s3.AmazonS3;
import com.amazonaws.services.s3.AmazonS3ClientBuilder;
import com.amazonaws.services.s3.model.BucketAccelerateConfiguration;
import com.amazonaws.services.s3.model.BucketAccelerateStatus;
import com.amazonaws.services.s3.model.GetBucketAccelerateConfigurationRequest;
import com.amazonaws.services.s3.model.SetBucketAccelerateConfigurationRequest;

public class TransferAcceleration {
    public static void main(String[] args) {
        Regions clientRegion = Regions.DEFAULT_REGION;
        String bucketName = "*** Bucket name ***";
        String keyName = "*** Key name ***";

        try {
            // Create an Amazon S3 client that is configured to use the accelerate endpoint.
            AmazonS3 s3Client = AmazonS3ClientBuilder.standard()
                .withRegion(clientRegion)
                .withCredentials(new ProfileCredentialsProvider())
                .enableAccelerateMode()
                .build();

            // Enable Transfer Acceleration for the specified bucket.
            s3Client.setBucketAccelerateConfiguration(
                new SetBucketAccelerateConfigurationRequest(bucketName,
                    new BucketAccelerateConfiguration(
                        BucketAccelerateStatus.Enabled)));
        }

        // Verify that transfer acceleration is enabled for the bucket.
        String accelerateStatus = s3Client.getBucketAccelerateConfiguration(
            new GetBucketAccelerateConfigurationRequest(bucketName))
            .getStatus();
        System.out.println("Bucket accelerate status: " + accelerateStatus);

        // Upload a new object using the accelerate endpoint.
    }
}
```

```
        s3Client.putObject(bucketName, keyName, "Test object for transfer
acceleration");
        System.out.println("Object \\" + keyName + "\\" uploaded with transfer
acceleration.");
    } catch (AmazonServiceException e) {
        // The call was transmitted successfully, but Amazon S3 couldn't process
        // it, so it returned an error response.
        e.printStackTrace();
    } catch (SdkClientException e) {
        // Amazon S3 couldn't be contacted for a response, or the client
        // couldn't parse the response from Amazon S3.
        e.printStackTrace();
    }
}
```

.NET

The following example shows how to use the AWS SDK for .NET to enable Transfer Acceleration on a bucket. For instructions on how to create and test a working sample, see [Running the Amazon S3 .NET Code Examples \(p. 1193\)](#).

Example

```
using Amazon;
using Amazon.S3;
using Amazon.S3.Model;
using System;
using System.Threading.Tasks;

namespace Amazon.DocSamples.S3
{
    class TransferAccelerationTest
    {
        private const string bucketName = "*** bucket name ***";
        // Specify your bucket region (an example region is shown).
        private static readonly RegionEndpoint bucketRegion = RegionEndpoint.USWest2;
        private static IAmazonS3 s3Client;
        public static void Main()
        {
            s3Client = new AmazonS3Client(bucketRegion);
            EnableAccelerationAsync().Wait();
        }

        static async Task EnableAccelerationAsync()
        {
            try
            {
                var putRequest = new PutBucketAccelerateConfigurationRequest
                {
                    BucketName = bucketName,
                    AccelerateConfiguration = new AccelerateConfiguration
                    {
                        Status = BucketAccelerateStatus.Enabled
                    }
                };
                await s3Client.PutBucketAccelerateConfigurationAsync(putRequest);

                var getRequest = new GetBucketAccelerateConfigurationRequest
                {
                    BucketName = bucketName
                };
            }
        }
    }
}
```

```
        var response = await
s3Client.GetBucketAccelerateConfigurationAsync(getRequest);

        Console.WriteLine("Acceleration state = '{0}' ", response.Status);
    }
    catch (AmazonS3Exception amazonS3Exception)
    {
        Console.WriteLine(
            "Error occurred. Message:{0} when setting transfer
acceleration",
            amazonS3Exception.Message);
    }
}
}
```

When uploading an object to a bucket that has Transfer Acceleration enabled, you specify using the acceleration endpoint at the time of creating a client.

```
var client = new AmazonS3Client(new AmazonS3Config
{
    RegionEndpoint = TestRegionEndpoint,
    UseAccelerateEndpoint = true
})
```

Javascript

For an example of enabling Transfer Acceleration by using the AWS SDK for JavaScript, see [Calling the putBucketAccelerateConfiguration operation in the AWS SDK for JavaScript API Reference](#).

Python (Boto)

For an example of enabling Transfer Acceleration by using the SDK for Python, see [put_bucket_accelerate_configuration in the AWS SDK for Python \(Boto3\) API Reference](#).

Other

For information about using other AWS SDKs, see [Sample Code and Libraries](#).

Using the Amazon S3 Transfer Acceleration Speed Comparison tool

You can use the [Amazon S3 Transfer Acceleration Speed Comparison tool](#) to compare accelerated and non-accelerated upload speeds across Amazon S3 Regions. The Speed Comparison tool uses multipart uploads to transfer a file from your browser to various Amazon S3 Regions with and without using Transfer Acceleration.

You can access the Speed Comparison tool using either of the following methods:

- Copy the following URL into your browser window, replacing `region` with the AWS Region that you are using (for example, `us-west-2`) and `yourBucketName` with the name of the bucket that you want to evaluate:

`https://s3-accelerate-speedtest.s3-accelerate.amazonaws.com/en/accelerate-speed-comparsion.html?region=region&origBucketName=yourBucketName`

For a list of the Regions supported by Amazon S3, see [Amazon S3 endpoints and quotas](#) in the [AWS General Reference](#).

- Use the Amazon S3 console.

Using Requester Pays buckets for storage transfers and usage

In general, bucket owners pay for all Amazon S3 storage and data transfer costs that are associated with their bucket. However, you can configure a bucket to be a *Requester Pays* bucket. With Requester Pays buckets, the requester instead of the bucket owner pays the cost of the request and the data download from the bucket. The bucket owner always pays the cost of storing data.

Typically, you configure buckets to be Requester Pays buckets when you want to share data but not incur charges associated with others accessing the data. For example, you might use Requester Pays buckets when making available large datasets, such as zip code directories, reference data, geospatial information, or web crawling data.

Important

If you enable Requester Pays on a bucket, anonymous access to that bucket is not allowed.

You must authenticate all requests involving Requester Pays buckets. The request authentication enables Amazon S3 to identify and charge the requester for their use of the Requester Pays bucket.

When the requester assumes an AWS Identity and Access Management (IAM) role before making their request, the account to which the role belongs is charged for the request. For more information about IAM roles, see [IAM roles](#) in the *IAM User Guide*.

After you configure a bucket to be a Requester Pays bucket, requesters must include `x-amz-request-payer` in their API request header, for DELETE, GET, HEAD, POST, and PUT requests, or as a parameter in a REST request to show that they understand that they will be charged for the request and the data download.

Requester Pays buckets do not support the following:

- Anonymous requests
- SOAP requests
- Using a Requester Pays bucket as the target bucket for end-user logging, or vice versa. However, you can turn on end-user logging on a Requester Pays bucket where the target bucket is not a Requester Pays bucket.

How Requester Pays charges work

The charge for successful Requester Pays requests is straightforward: The requester pays for the data transfer and the request, and the bucket owner pays for the data storage. However, the bucket owner is charged for the request under the following conditions:

- The requester doesn't include the parameter `x-amz-request-payer` in the header (DELETE, GET, HEAD, POST, and PUT) or as a parameter (REST) in the request (HTTP code 403).
- Request authentication fails (HTTP code 403).
- The request is anonymous (HTTP code 403).
- The request is a SOAP request.

For more information about Requester Pays, see the following topics.

Topics

- [Configuring Requester Pays on a bucket \(p. 145\)](#)
- [Retrieving the requestPayment configuration using the REST API \(p. 146\)](#)
- [Downloading objects in Requester Pays buckets \(p. 146\)](#)

Configuring Requester Pays on a bucket

You can configure an Amazon S3 bucket to be a *Requester Pays* bucket so that the requester pays the cost of the request and data download instead of the bucket owner.

This section provides examples of how to configure Requester Pays on an Amazon S3 bucket using the console and the REST API.

Using the S3 console

To enable Requester Pays for an S3 bucket

1. Sign in to the AWS Management Console and open the Amazon S3 console at <https://console.aws.amazon.com/s3/>.
2. In the **Buckets** list, choose the name of the bucket that you want to enable Requester Pays for.
3. Choose **Properties**.
4. Under **Requester pays**, choose **Edit**.
5. Choose **Enable**, and choose **Save changes**.

Amazon S3 enables Requester Pays for your bucket and displays your **Bucket overview**. Under **Requester pays**, you see **Enabled**.

Using the REST API

Only the bucket owner can set the `RequestPaymentConfiguration.payer` configuration value of a bucket to `BucketOwner` (the default) or `Requester`. Setting the `requestPayment` resource is optional. By default, the bucket is not a Requester Pays bucket.

To revert a Requester Pays bucket to a regular bucket, you use the value `BucketOwner`. Typically, you would use `BucketOwner` when uploading data to the Amazon S3 bucket, and then you would set the value to `Requester` before publishing the objects in the bucket.

To set `requestPayment`

- Use a `PUT` request to set the `Payer` value to `Requester` on a specified bucket.

```
PUT ?requestPayment HTTP/1.1
Host: [BucketName].s3.amazonaws.com
Content-Length: 173
Date: Wed, 01 Mar 2009 12:00:00 GMT
Authorization: AWS [Signature]

<RequestPaymentConfiguration xmlns="http://s3.amazonaws.com/doc/2006-03-01/">
<Payer>Requester</Payer>
</RequestPaymentConfiguration>
```

If the request succeeds, Amazon S3 returns a response similar to the following.

```
HTTP/1.1 200 OK
x-amz-id-2: [id]
```

```
x-amz-request-id: [request_id]
Date: Wed, 01 Mar 2009 12:00:00 GMT
Content-Length: 0
Connection: close
Server: AmazonS3
x-amz-request-charged:requester
```

You can set Requester Pays only at the bucket level. You can't set Requester Pays for specific objects within the bucket.

You can configure a bucket to be BucketOwner or Requester at any time. However, there might be a few minutes before the new configuration value takes effect.

Note

Bucket owners who give out presigned URLs should consider carefully before configuring a bucket to be Requester Pays, especially if the URL has a long lifetime. The bucket owner is charged each time the requester uses a presigned URL that uses the bucket owner's credentials.

Retrieving the requestPayment configuration using the REST API

You can determine the Payer value that is set on a bucket by requesting the resource `requestPayment`.

To return the `requestPayment` resource

- Use a GET request to obtain the `requestPayment` resource, as shown in the following request.

```
GET ?requestPayment HTTP/1.1
Host: [BucketName].s3.amazonaws.com
Date: Wed, 01 Mar 2009 12:00:00 GMT
Authorization: AWS [Signature]
```

If the request succeeds, Amazon S3 returns a response similar to the following.

```
HTTP/1.1 200 OK
x-amz-id-2: [id]
x-amz-request-id: [request_id]
Date: Wed, 01 Mar 2009 12:00:00 GMT
Content-Type: [type]
Content-Length: [length]
Connection: close
Server: AmazonS3

<?xml version="1.0" encoding="UTF-8"?>
<RequestPaymentConfiguration xmlns="http://s3.amazonaws.com/doc/2006-03-01/">
<Payer>Requester</Payer>
</RequestPaymentConfiguration>
```

This response shows that the `payer` value is set to Requester.

Downloading objects in Requester Pays buckets

Because requesters are charged for downloading data from Requester Pays buckets, the requests must contain a special parameter, `x-amz-request-payer`, which confirms that the requester knows that they will be charged for the download. To access objects in Requester Pays buckets, requests must include one of the following.

- For DELETE, GET, HEAD, POST, and PUT requests, include `x-amz-request-payer : requester` in the header
- For signed URLs, include `x-amz-request-payer=requester` in the request

If the request succeeds and the requester is charged, the response includes the header `x-amz-request-charged: requester`. If `x-amz-request-payer` is not in the request, Amazon S3 returns a 403 error and charges the bucket owner for the request.

Note

Bucket owners do not need to add `x-amz-request-payer` to their requests. Ensure that you have included `x-amz-request-payer` and its value in your signature calculation. For more information, see [Constructing the CanonicalizedAmzHeaders Element \(p. 1213\)](#).

Using the REST API

To download objects from a Requester Pays bucket

- Use a GET request to download an object from a Requester Pays bucket, as shown in the following request.

```
GET / [destinationObject] HTTP/1.1
Host: [BucketName].s3.amazonaws.com
x-amz-request-payer : requester
Date: Wed, 01 Mar 2009 12:00:00 GMT
Authorization: AWS [Signature]
```

If the GET request succeeds and the requester is charged, the response includes `x-amz-request-charged: requester`.

Amazon S3 can return an Access Denied error for requests that try to get objects from a Requester Pays bucket. For more information, see [Error Responses](#) in the *Amazon Simple Storage Service API Reference*.

Using the AWS CLI

To download objects from a Requester Pays bucket using the AWS CLI, you specify `--request-payer requester` as part of your `get-object` request. For more information, see [get-object](#) in the *AWS CLI Reference*.

Bucket restrictions and limitations

An Amazon S3 bucket is owned by the AWS account that created it. Bucket ownership is not transferable to another account.

When you create a bucket, you choose its name and the AWS Region to create it in. After you create a bucket, you can't change its name or Region.

When naming a bucket, choose a name that is relevant to you or your business. Avoid using names associated with others. For example, you should avoid using AWS or Amazon in your bucket name.

By default, you can create up to 100 buckets in each of your AWS accounts. If you need additional buckets, you can increase your account bucket limit to a maximum of 1,000 buckets by submitting a service limit increase. There is no difference in performance whether you use many buckets or just a few.

For information about how to increase your bucket limit, see [AWS service quotas](#) in the *AWS General Reference*.

Reusing bucket names

If a bucket is empty, you can delete it. After a bucket is deleted, the name becomes available for reuse. However, after you delete the bucket, you might not be able to reuse the name for various reasons.

For example, when you delete the bucket and the name becomes available for reuse, another AWS account might create a bucket with that name. In addition, some time might pass before you can reuse the name of a deleted bucket. If you want to use the same bucket name, we recommend that you don't delete the bucket.

For more information about bucket names, see [Bucket naming rules \(p. 118\)](#).

Objects and bucket limitations

There is no max bucket size or limit to the number of objects that you can store in a bucket. You can store all of your objects in a single bucket, or you can organize them across several buckets. However, you can't create a bucket from within another bucket.

Bucket operations

The high availability engineering of Amazon S3 is focused on *get*, *put*, *list*, and *delete* operations. Because bucket operations work against a centralized, global resource space, it is not recommended to create, delete, or configure buckets on the high availability code path of your application. It's better to create, delete, or configure buckets in a separate initialization or setup routine that you run less often.

Bucket naming and automatically created buckets

If your application automatically creates buckets, choose a bucket naming scheme that is unlikely to cause naming conflicts. Ensure that your application logic will choose a different bucket name if a bucket name is already taken.

For more information about bucket naming, see [Bucket naming rules \(p. 118\)](#).

Uploading, downloading, and working with objects in Amazon S3

To store your data in Amazon S3, you work with resources known as buckets and objects. A *bucket* is a container for objects. An *object* is a file and any metadata that describes that file.

To store an object in Amazon S3, you create a bucket and then upload the object to a bucket. When the object is in the bucket, you can open it, download it, and copy it. When you no longer need an object or a bucket, you can clean up these resources.

With Amazon S3, you pay only for what you use. For more information about Amazon S3 features and pricing, see [Amazon S3](#). If you are a new Amazon S3 customer, you can get started with Amazon S3 for free. For more information, see [AWS Free Tier](#).

Topics

- [Amazon S3 objects overview \(p. 149\)](#)
- [Creating object key names \(p. 150\)](#)
- [Working with object metadata \(p. 153\)](#)
- [Uploading objects \(p. 158\)](#)
- [Uploading and copying objects using multipart upload \(p. 167\)](#)
- [Copying objects \(p. 201\)](#)
- [Downloading an object \(p. 209\)](#)
- [Checking object integrity \(p. 215\)](#)
- [Deleting Amazon S3 objects \(p. 223\)](#)
- [Organizing, listing, and working with your objects \(p. 243\)](#)
- [Using presigned URLs \(p. 257\)](#)
- [Transforming objects with S3 Object Lambda \(p. 271\)](#)

Amazon S3 objects overview

Amazon S3 is an object store that uses unique key-values to store as many objects as you want. You store these objects in one or more buckets, and each object can be up to 5 TB in size. An object consists of the following:

Key

The name that you assign to an object. You use the object key to retrieve the object. For more information, see [Working with object metadata \(p. 153\)](#).

Version ID

Within a bucket, a key and version ID uniquely identify an object. The version ID is a string that Amazon S3 generates when you add an object to a bucket. For more information, see [Using versioning in S3 buckets \(p. 638\)](#).

Value

The content that you are storing.

An object value can be any sequence of bytes. Objects can range in size from zero to 5 TB. For more information, see [Uploading objects \(p. 158\)](#).

Metadata

A set of name-value pairs with which you can store information regarding the object. You can assign metadata, referred to as user-defined metadata, to your objects in Amazon S3. Amazon S3 also assigns system-metadata to these objects, which it uses for managing objects. For more information, see [Working with object metadata \(p. 153\)](#).

Subresources

Amazon S3 uses the subresource mechanism to store object-specific additional information. Because subresources are subordinates to objects, they are always associated with some other entity such as an object or a bucket. For more information, see [Object subresources \(p. 150\)](#).

Access control information

You can control access to the objects you store in Amazon S3. Amazon S3 supports both the resource-based access control, such as an access control list (ACL) and bucket policies, and user-based access control. For more information about access control, see the following:

- [Access control best practices \(p. 22\)](#)
- [Access policy guidelines \(p. 400\)](#)
- [Identity and access management in Amazon S3 \(p. 394\)](#)
- [Configuring ACLs \(p. 562\)](#)

Your Amazon S3 resources (for example, buckets and objects) are private by default. You must explicitly grant permission for others to access these resources. For more information about sharing objects, see [Sharing objects using presigned URLs \(p. 258\)](#).

Object subresources

Amazon S3 defines a set of subresources associated with buckets and objects. Subresources are subordinates to objects. This means that subresources don't exist on their own. They are always associated with some other entity, such as an object or a bucket.

The following table lists the subresources associated with Amazon S3 objects.

Subresource	Description
acl	Contains a list of grants identifying the grantees and the permissions granted. When you create an object, the acl identifies the object owner as having full control over the object. You can retrieve an object ACL or replace it with an updated list of grants. Any update to an ACL requires you to replace the existing ACL. For more information about ACLs, see Access control list (ACL) overview (p. 554) .

Creating object key names

The *object key* (or key name) uniquely identifies the object in an Amazon S3 bucket. *Object metadata* is a set of name-value pairs. For more information about object metadata, see [Working with object metadata \(p. 153\)](#).

When you create an object, you specify the key name, which uniquely identifies the object in the bucket. For example, on the [Amazon S3 console](#), when you highlight a bucket, a list of objects in your bucket

appears. These names are the *object keys*. The object key name is a sequence of Unicode characters with UTF-8 encoding of up to 1,024 bytes long.

Note

Object key names with the value "soap" aren't supported for [virtual-hosted-style requests](#). For object key name values where "soap" is used, a [path-style URL](#) must be used instead.

The Amazon S3 data model is a flat structure: You create a bucket, and the bucket stores objects. There is no hierarchy of subbuckets or subfolders. However, you can infer logical hierarchy using key name prefixes and delimiters as the Amazon S3 console does. The Amazon S3 console supports a concept of folders. For more information about how to edit metadata from the Amazon S3 console, see [Editing object metadata in the Amazon S3 console \(p. 157\)](#).

Suppose that your bucket (admin-created) has four objects with the following object keys:

Development/Projects.xls
Finance/statement1.pdf
Private/taxdocument.pdf
s3-dg.pdf

The console uses the key name prefixes (Development/, Finance/, and Private/) and delimiter ('/') to present a folder structure. The s3-dg.pdf key does not have a prefix, so its object appears directly at the root level of the bucket. If you open the Development/ folder, you see the Projects.xlsx object in it.

- Amazon S3 supports buckets and objects, and there is no hierarchy. However, by using prefixes and delimiters in an object key name, the Amazon S3 console and the AWS SDKs can infer hierarchy and introduce the concept of folders.
- The Amazon S3 console implements folder object creation by creating a zero-byte object with the folder *prefix and delimiter* value as the key. These folder objects don't appear in the console. Otherwise they behave like any other objects and can be viewed and manipulated through the REST API, AWS CLI, and AWS SDKs.

Object key naming guidelines

You can use any UTF-8 character in an object key name. However, using certain characters in key names can cause problems with some applications and protocols. The following guidelines help you maximize compliance with DNS, web-safe characters, XML parsers, and other APIs.

Safe characters

The following character sets are generally safe for use in key names.

Alphanumeric characters

- 0-9
- a-z
- A-Z

Special characters

- Exclamation point (!)
- Hyphen (-)
- Underscore (_)
- Period (.)
- Asterisk (*)

- Single quote (')
- Open parenthesis (()
- Close parenthesis ())

The following are examples of valid object key names:

- 4my-organization
- my.great_photos-2014/jan/myvacation.jpg
- videos/2014/birthday/video1.wmv

Note

Objects with key names ending with period(s) "." downloaded using the Amazon S3 console will have the period(s) "." removed from the key name of the downloaded object. To download an object with the key name ending in period(s) "." retained in the downloaded object, you must use the AWS Command Line Interface (AWS CLI), AWS SDKs, or REST API.

In addition, be aware of the following prefix limitations:

- Objects with a prefix of "./" must uploaded or downloaded with the AWS Command Line Interface (AWS CLI), AWS SDKs, or REST API. You cannot use the Amazon S3 console.
- Objects with a prefix of "../" cannot be uploaded using the AWS Command Line Interface (AWS CLI) or Amazon S3 console.

Characters that might require special handling

The following characters in a key name might require additional code handling and likely need to be URL encoded or referenced as HEX. Some of these are non-printable characters that your browser might not handle, which also requires special handling:

- Ampersand ("&")
- Dollar ("\$")
- ASCII character ranges 00–1F hex (0–31 decimal) and 7F (127 decimal)
- 'At' symbol ("@")
- Equals ("=")
- Semicolon (";")
- Forward slash ("/")
- Colon (":")
- Plus ("+")
- Space – Significant sequences of spaces might be lost in some uses (especially multiple spaces)
- Comma (",")
- Question mark ("?")

Characters to avoid

Avoid the following characters in a key name because of significant special handling for consistency across all applications.

- Backslash ("\")
- Left curly brace ("{")
- Non-printable ASCII characters (128–255 decimal characters)

- Caret ("^")
- Right curly brace ("}")
- Percent character ("%")
- Grave accent / back tick ("`")
- Right square bracket ("]")
- Quotation marks
- 'Greater Than' symbol (">")
- Left square bracket ("[")
- Tilde ("~")
- 'Less Than' symbol ("<")
- 'Pound' character ("#")
- Vertical bar / pipe ("|")

XML related object key constraints

As specified by the [XML standard on end-of-line handling](#), all XML text is normalized such that single carriage returns (ASCII code 13) and carriage returns immediately followed by a line feed (ASCII code 10) are replaced by a single line feed character. To ensure the correct parsing of object keys in XML requests, carriage returns and [other special characters must be replaced with their equivalent XML entity code](#) when they are inserted within XML tags. The following is a list of such special characters and their equivalent entity codes:

- ' as '
- " as "
- & as &
- < as <
- > as >
- \r as  or

- \n as
 or

Example

The following example illustrates the use of an XML entity code as a substitution for a carriage return. This DeleteObjects request deletes an object with the key parameter: /some/prefix/objectwith\rcarriagereturn (where the \r is the carriage return).

```
<Delete xmlns="http://s3.amazonaws.com/doc/2006-03-01/">
  <Object>
    <Key>/some/prefix/objectwith<br/>carriagereturn</Key>
  </Object>
</Delete>
```

Working with object metadata

You can set object metadata in Amazon S3 at the time you upload the object. Object metadata is a set of name-value pairs. After you upload the object, you cannot modify object metadata. The only way to modify object metadata is to make a copy of the object and set the metadata.

When you create an object, you also specify the key name, which uniquely identifies the object in the bucket. The object key (or key name) uniquely identifies the object in an Amazon S3 bucket. For more information, see [Creating object key names \(p. 150\)](#).

There are two kinds of metadata in Amazon S3: *system-defined metadata* and *user-defined metadata*. The sections below provide more information about system-defined and user-defined metadata. For more information about editing metadata using the Amazon S3 console, see [Editing object metadata in the Amazon S3 console \(p. 157\)](#).

System-defined object metadata

For each object stored in a bucket, Amazon S3 maintains a set of system metadata. Amazon S3 processes this system metadata as needed. For example, Amazon S3 maintains object creation date and size metadata and uses this information as part of object management.

There are two categories of system metadata:

1. Metadata such as object creation date is system controlled, where only Amazon S3 can modify the value.
2. Other system metadata, such as the storage class configured for the object and whether the object has server-side encryption enabled, are examples of system metadata whose values you control. If your bucket is configured as a website, sometimes you might want to redirect a page request to another page or an external URL. In this case, a webpage is an object in your bucket. Amazon S3 stores the page redirect value as system metadata whose value you control.

When you create objects, you can configure values of these system metadata items or update the values when you need to. For more information about storage classes, see [Using Amazon S3 storage classes \(p. 688\)](#).

Amazon S3 uses AWS KMS keys to encrypt your Amazon S3 objects. AWS KMS encrypts only the object data. The checksum, along with the specified algorithm, are stored as part of the object's metadata. If server-side encryption is requested for the object, then the checksum is stored in encrypted form. For more information about server-side encryption, see [Protecting data using encryption \(p. 337\)](#).

Note

The PUT request header is limited to 8 KB in size. Within the PUT request header, the system-defined metadata is limited to 2 KB in size. The size of system-defined metadata is measured by taking the sum of the number of bytes in the US-ASCII encoding of each key and value.

The following table provides a list of system-defined metadata and whether you can update it.

Name	Description	Can user modify the value?
Date	Current date and time.	No
Cache-Control	A general header field used to specify caching policies.	Yes
Content-Disposition	Object presentational information.	Yes
Content-Length	Object size in bytes.	No
Content-Type	Object type.	Yes
Last-Modified	Object creation date or the last modified date, whichever is the latest.	No

Name	Description	Can user modify the value?
ETag	Represents a specific version of an object. For objects that are not uploaded as a multipart upload and are either unencrypted or encrypted by server-side encryption with Amazon S3 managed keys (SSE-S3), the ETag is an MD5 digest of the data.	No
x-amz-server-side-encryption	Indicates whether server-side encryption is enabled for the object, and whether that encryption is from the AWS Key Management Service (AWS KMS) or from Amazon S3 managed encryption (SSE-S3). For more information, see Protecting data using server-side encryption (p. 338) .	Yes
x-amz-checksum-crc32, x-amz-checksum-crc32c, x-amz-checksum-sha1, x-amz-checksum-sha256	Contains the checksum or digest of the object. At most, one of these headers will be set at a time, depending on the checksum algorithm that you instruct Amazon S3 to use. For more information about choosing the checksum algorithm, see Checking object integrity (p. 215) .	No
x-amz-version-id	Object version. When you enable versioning on a bucket, Amazon S3 assigns a version number to objects added to the bucket. For more information, see Using versioning in S3 buckets (p. 638) .	No
x-amz-delete-marker	In a bucket that has versioning enabled, this Boolean marker indicates whether the object is a delete marker.	No
x-amz-storage-class	Storage class used for storing the object. For more information, see Using Amazon S3 storage classes (p. 688) .	Yes
x-amz-website-redirect-location	Redirects requests for the associated object to another object in the same bucket or an external URL. For more information, see (Optional) Configuring a webpage redirect (p. 1130) . This value is unique to each individual object and is not copied by default.	Yes
x-amz-server-side-encryption-aws-kms-key-id	If x-amz-server-side-encryption is present and has the value of aws:kms, this indicates the ID of the AWS KMS symmetric encryption KMS key that was used for the object.	Yes
x-amz-server-side-encryption-customer-algorithm	Indicates whether server-side encryption with customer-provided encryption keys (SSE-C) is enabled. For more information, see Protecting data using server-side encryption with customer-provided encryption keys (SSE-C) (p. 366) .	Yes
x-amz-tagging	The tag-set for the object. The tag-set must be encoded as URL Query parameters.	Yes

User-defined object metadata

When uploading an object, you can also assign metadata to the object. You provide this optional information as a name-value (key-value) pair when you send a PUT or POST request to create the object.

When you upload objects using the REST API, the optional user-defined metadata names must begin with "x-amz-meta-" to distinguish them from other HTTP headers. When you retrieve the object using the REST API, this prefix is returned. When you upload objects using the SOAP API, the prefix is not required. When you retrieve the object using the SOAP API, the prefix is removed, regardless of which API you used to upload the object.

Note

SOAP support over HTTP is deprecated, but SOAP is still available over HTTPS. New Amazon S3 features are not supported for SOAP. Instead of using SOAP, we recommend that you use either the REST API or the AWS SDKs.

When metadata is retrieved through the REST API, Amazon S3 combines headers that have the same name (ignoring case) into a comma-delimited list. If some metadata contains unprintable characters, it is not returned. Instead, the x-amz-missing-meta header is returned with a value of the number of unprintable metadata entries. The HeadObject action retrieves metadata from an object without returning the object itself. This operation is useful if you're only interested in an object's metadata. To use HEAD, you must have READ access to the object. For more information, see [HeadObject](#) in the *Amazon Simple Storage Service API Reference*.

User-defined metadata is a set of key-value pairs. Amazon S3 stores user-defined metadata keys in lowercase.

Amazon S3 allows arbitrary Unicode characters in your metadata values.

To avoid issues around the presentation of these metadata values, you should conform to using US-ASCII characters when using REST and UTF-8 when using SOAP or browser-based uploads via POST.

When using non US-ASCII characters in your metadata values, the provided unicode string is examined for non US-ASCII characters. Values of such headers are character decoded as per [RFC 2047](#) before storing and encoded as per [RFC 2047](#) to make them mail-safe before returning. If the string contains only US-ASCII characters, it is presented as is.

The following is an example.

```
PUT /Key HTTP/1.1
Host: awsexamplebucket1.s3.amazonaws.com
x-amz-meta-nonascii: ÄMÄZÖÑ S3

HEAD /Key HTTP/1.1
Host: awsexamplebucket1.s3.amazonaws.com
x-amz-meta-nonascii: =?UTF-8?B?w4PChE3Dg8KEWsODwpXDg8KRIFMz?=

PUT /Key HTTP/1.1
Host: awsexamplebucket1.s3.amazonaws.com
x-amz-meta-ascii: AMAZONS3

HEAD /Key HTTP/1.1
Host: awsexamplebucket1.s3.amazonaws.com
x-amz-meta-ascii: AMAZONS3
```

Note

The PUT request header is limited to 8 KB in size. Within the PUT request header, the user-defined metadata is limited to 2 KB in size. The size of user-defined metadata is measured by taking the sum of the number of bytes in the UTF-8 encoding of each key and value.

For information about changing the metadata of your object after it's been uploaded by creating a copy of the object, modifying it, and replacing the old object, or creating a new version, see [Editing object metadata in the Amazon S3 console \(p. 157\)](#).

Editing object metadata in the Amazon S3 console

You can use the Amazon S3 console to edit metadata of existing S3 objects. Some metadata is set by Amazon S3 when you upload the object. For example, `Content-Length` is the *key* (name) and the *value* is the size of the object in bytes.

You can also set some metadata when you upload the object and later edit it as your needs change. For example, you might have a set of objects that you initially store in the `STANDARD` storage class. Over time, you might no longer need this data to be highly available. So you change the storage class to `GLACIER` by editing the value of the `x-amz-storage-class` key from `STANDARD` to `GLACIER`.

Note

Consider the following issues when you are editing object metadata in Amazon S3:

- This action creates a *copy* of the object with updated settings and the last-modified date. If S3 Versioning is enabled, a new version of the object is created, and the existing object becomes an older version. If S3 Versioning is not enabled, a new copy of the object replaces the original object. The AWS account associated with the IAM role that changes the property also becomes the owner of the new object or (object version).
- Editing metadata updates values for existing key names.
- Objects that are encrypted with customer-provided encryption keys (SSE-C) cannot be copied using the console. You must use the AWS CLI, AWS SDK, or the Amazon S3 REST API.

Warning

When editing metadata of folders, wait for the `Edit metadata` operation to finish before adding new objects to the folder. Otherwise, new objects might also be edited.

The following topics describe how to edit metadata of an object using the Amazon S3 console.

Editing system-defined metadata

You can configure some, but not all, system metadata for an S3 object. For a list of system-defined metadata and whether you can modify their values, see [System-defined object metadata \(p. 154\)](#).

To edit system-defined metadata 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. Navigate to your Amazon S3 bucket or folder, and select the check box to the left of the names of the objects with metadata you want to edit.
3. On the **Actions** menu, choose **Edit actions**, and choose **Edit metadata**.
4. Review the objects listed, and choose **Add metadata**.
5. For metadata **Type**, select **System-defined**.
6. Specify a unique **Key** and the metadata **Value**.
7. To edit additional metadata, choose **Add metadata**. You can also choose **Remove** to remove a set of type-key-values.
8. When you are done, choose **Edit metadata** and Amazon S3 edits the metadata of the specified objects.

Editing user-defined metadata

You can edit user-defined metadata of an object by combining the metadata prefix, `x-amz-meta-`, and a name you choose to create a custom key. For example, if you add the custom name `alt-name`, the metadata key would be `x-amz-meta-alt-name`.

User-defined metadata can be as large as 2 KB total. To calculate the total size of user-defined metadata, sum the number of bytes in the UTF-8 encoding for each key and value. Both keys and their values must conform to US-ASCII standards. For more information, see [User-defined object metadata \(p. 155\)](#).

To edit user-defined metadata 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 **Buckets** list, choose the name of the bucket that contains the objects that you want to add metadata to.

You can also optionally navigate to a folder.
3. In the **Objects** list, select the check box next to the names of the objects that you want to add metadata to.
4. On the **Actions** menu, choose **Edit metadata**.
5. Review the objects listed, and choose **Add metadata**.
6. For metadata **Type**, choose **User-defined**.
7. Enter a unique custom **Key** following x-amz-meta-. Also enter a metadata **Value**.
8. To add additional metadata, choose **Add metadata**. You can also choose **Remove** to remove a set of type-key-values.
9. Choose **Edit metadata**.

Amazon S3 edits the metadata of the specified objects.

Uploading objects

When you upload a file to Amazon S3, it is stored as an S3 object. Objects consist of the file data and metadata that describes the object. You can have an unlimited number of objects in a bucket. Before you can upload files to an Amazon S3 bucket, you need write permissions for the bucket. For more information about access permissions, see [Identity and access management in Amazon S3 \(p. 394\)](#).

You can upload any file type—images, backups, data, movies, etc.—into an S3 bucket. The maximum size of a file that you can upload by using the Amazon S3 console is 160 GB. To upload a file larger than 160 GB, use the AWS CLI, AWS SDK, or Amazon S3 REST API.

If you upload an object with a key name that already exists in a versioning-enabled bucket, Amazon S3 creates another version of the object instead of replacing the existing object. For more information about versioning, see [Using the S3 console \(p. 644\)](#).

Depending on the size of the data you are uploading, Amazon S3 offers the following options:

- **Upload an object in a single operation using the AWS SDKs, REST API, or AWS CLI**—With a single PUT operation, you can upload a single object up to 5 GB in size.
- **Upload a single object using the Amazon S3 Console**—With the Amazon S3 Console, you can upload a single object up to 160 GB in size.
- **Upload an object in parts using the AWS SDKs, REST API, or AWS CLI**—Using the multipart upload API, you can upload a single large object, up to 5 TB in size.

The multipart upload API is designed to improve the upload experience for larger objects. You can upload an object in parts. These object parts can be uploaded independently, in any order, and in parallel. You can use a multipart upload for objects from 5 MB to 5 TB in size. For more information, see [Uploading and copying objects using multipart upload \(p. 167\)](#).

When uploading an object, you can optionally request that Amazon S3 encrypt it before saving it to disk, and decrypt it when you download it. For more information, see [Protecting data using encryption \(p. 337\)](#).

Using the S3 console

This procedure explains how to upload objects and folders to an S3 bucket using the console.

When you upload an object, the object key name is the file name and any optional prefixes. In the Amazon S3 console, you can create folders to organize your objects. In Amazon S3, folders are represented as prefixes that appear in the object key name. If you upload an individual object to a folder in the Amazon S3 console, the folder name is included in the object key name.

For example, if you upload an object named `sample1.jpg` to a folder named `backup`, the key name is `backup/sample1.jpg`. However, the object is displayed in the console as `sample1.jpg` in the `backup` folder. For more information about key names, see [Working with object metadata \(p. 153\)](#).

Note

If you rename an object or change any of the properties in the Amazon S3 console, for example **Storage Class**, **Encryption**, **Metadata**, a new object is created to replace the old one. If S3 Versioning is enabled, a new version of the object is created, and the existing object becomes an older version. The role that changes the property also becomes the owner of the new object or (object version).

When you upload a folder, Amazon S3 uploads all of the files and subfolders from the specified folder to your bucket. It then assigns an object key name that is a combination of the uploaded file name and the folder name. For example, if you upload a folder named `/images` that contains two files, `sample1.jpg` and `sample2.jpg`, Amazon S3 uploads the files and then assigns the corresponding key names, `images/sample1.jpg` and `images/sample2.jpg`. The key names include the folder name as a prefix. The Amazon S3 console displays only the part of the key name that follows the last `/`. For example, within an `images` folder the `images/sample1.jpg` and `images/sample2.jpg` objects are displayed as `sample1.jpg` and a `sample2.jpg`.

To upload folders and files to an S3 bucket

1. Sign in to the AWS Management Console and open the Amazon S3 console at <https://console.aws.amazon.com/s3/>.
2. In the **Buckets** list, choose the name of the bucket that you want to upload your folders or files to.
3. Choose **Upload**.
4. In the **Upload** window, do one of the following:
 - Drag and drop files and folders to the **Upload** window.
 - Choose **Add file** or **Add folder**, choose files or folders to upload, and choose **Open**.
5. To enable versioning, under **Destination**, choose **Enable Bucket Versioning**.
6. To upload the listed files and folders without configuring additional upload options, at the bottom of the page, choose **Upload**.

Amazon S3 uploads your objects and folders. When the upload completes, you can see a success message on the **Upload: status** page.

To configure additional object properties

1. To change access control list permissions, choose **Permissions**.
2. Under **Access control list (ACL)**, edit the permissions.

For information about object access permissions, see [Using the S3 console to set ACL permissions for an object \(p. 565\)](#). You can grant read access to your objects to the public (everyone in the world)

for all of the files that you're uploading. However, we recommend not changing the default setting for public read access. Granting public read access is applicable to a small subset of use cases, such as when buckets are used for websites. You can always change the object permissions after you upload the object.

3. To configure other additional properties, choose **Properties**.
4. Under **Storage class**, choose the storage class for the files you're uploading.

For more information about storage classes, see [Using Amazon S3 storage classes \(p. 688\)](#).

5. To update the encryption settings for your objects, under **Server-side encryption settings**, do the following.

- a. Choose **Specify an encryption key**.
- b. To encrypt the uploaded files using keys that are managed by Amazon S3, choose **Amazon S3 key (SSE-S3)**.

For more information, see [Protecting data using server-side encryption with Amazon S3-managed encryption keys \(SSE-S3\) \(p. 355\)](#).
- c. To encrypt the uploaded files using the AWS Key Management Service (AWS KMS), choose **AWS Key Management Service key (SSE-KMS)**. Then choose an option for **AWS KMS key**.
 - **AWS managed key** – Choose an [AWS managed key](#).
 - **Choose from your KMS root keys** – Choose a [customer managed key](#) from a list of KMS keys in the same Region as your bucket.

For more information about creating a customer managed key, see [Creating Keys](#) in the *AWS Key Management Service Developer Guide*. For more information about protecting data with AWS KMS, see [Protecting data using server-side encryption with AWS Key Management Service \(SSE-KMS\) \(p. 338\)](#).

- **Enter KMS root key ARN** – Specify the AWS KMS key ARN for a customer managed key, and enter the Amazon Resource Name (ARN).

You can use the KMS root key ARN to give an external account the ability to use an object that is protected by an AWS KMS key. To do this, choose **Enter KMS root key ARN**, and enter the Amazon Resource Name (ARN) for the external account. Administrators of an external account that have usage permissions to an object protected by your KMS key can further restrict access by creating a resource-level IAM policy.

Note

To encrypt objects in a bucket, you can use only AWS KMS keys that are available in the same AWS Region as the bucket.

6. To use additional checksums, choose **On**. Then for **Checksum function**, choose the function that you would like to use. Amazon S3 calculates and stores the checksum value after it receives the entire object. You can use the **Precalculated value** box to supply a precalculated value. If you do, Amazon S3 compares the value that you provided to the value that it calculates. If the two values do not match, Amazon S3 generates an error.

Additional checksums enable you to specify the checksum algorithm that you would like to use to verify your data. For more information about additional checksums, see [Checking object integrity \(p. 215\)](#).

7. To add tags to all of the objects that you are uploading, choose **Add tag**. Type a tag name in the **Key** field. Type a value for the tag.

Object tagging gives you a way to categorize storage. Each tag is a key-value pair. Key and tag values are case sensitive. You can have up to 10 tags per object. A tag key can be up to 128 Unicode

characters in length and tag values can be up to 255 Unicode characters in length. For more information about object tags, see [Categorizing your storage using tags \(p. 825\)](#).

8. To add metadata, choose **Add metadata**.

a. Under **Type**, choose **System defined or User defined**.

For system-defined metadata, you can select common HTTP headers, such as **Content-Type** and **Content-Disposition**. For a list of system-defined metadata and information about whether you can add the value, see [System-defined object metadata \(p. 154\)](#). Any metadata starting with prefix `x-amz-meta-` is treated as user-defined metadata. User-defined metadata is stored with the object and is returned when you download the object. Both the keys and their values must conform to US-ASCII standards. User-defined metadata can be as large as 2 KB. For more information about system-defined and user-defined metadata, see [Working with object metadata \(p. 153\)](#).

b. For **Key**, choose a key.

c. Type a value for the key.

9. To upload your objects, choose **Upload**.

Amazon S3 uploads your object. When the upload completes, you can see a success message on the [Upload: status](#) page.

10. Choose **Exit**.

Using the AWS SDKs

You can use the AWS SDK to upload objects in Amazon S3. The SDK provides wrapper libraries for you to upload data easily. For information, see the [List of supported SDKs](#).

Here are a few examples with a few select SDKs:

.NET

The following C# code example creates two objects with two `PutObjectRequest` requests:

- The first `PutObjectRequest` request saves a text string as sample object data. It also specifies the bucket and object key names.
- The second `PutObjectRequest` request uploads a file by specifying the file name. This request also specifies the `ContentType` header and optional object metadata (a title).

For instructions on how to create and test a working sample, see [Running the Amazon S3 .NET Code Examples \(p. 1193\)](#).

```
using Amazon;
using Amazon.S3;
using Amazon.S3.Model;
using System;
using System.Threading.Tasks;

namespace Amazon.DocSamples.S3
{
    class UploadObjectTest
    {
        private const string bucketName = "*** bucket name ***";
        // For simplicity the example creates two objects from the same file.
        // You specify key names for these objects.
        private const string keyName1 = "*** key name for first object created ***";
        private const string keyName2 = "*** key name for second object created ***";
```

```

private const string filePath = @""" file path ***";
private static readonly RegionEndpoint bucketRegion = RegionEndpoint.EUWest1;

private static IAmazonS3 client;

public static void Main()
{
    client = new AmazonS3Client(bucketRegion);
    WritingAnObjectAsync().Wait();
}

static async Task WritingAnObjectAsync()
{
    try
    {
        // 1. Put object-specify only key name for the new object.
        var putRequest1 = new PutObjectRequest
        {
            BucketName = bucketName,
            Key = keyName1,
            ContentBody = "sample text"
        };

        PutObjectResponse response1 = await client.PutObjectAsync(putRequest1);

        // 2. Put the object-set ContentType and add metadata.
        var putRequest2 = new PutObjectRequest
        {
            BucketName = bucketName,
            Key = keyName2,
            FilePath = filePath,
            ContentType = "text/plain"
        };

        putRequest2.Metadata.Add("x-amz-meta-title", "someTitle");
        PutObjectResponse response2 = await client.PutObjectAsync(putRequest2);
    }
    catch (AmazonS3Exception e)
    {
        Console.WriteLine(
            "Error encountered ***. Message:'{0}' when writing an object"
            , e.Message);
    }
    catch (Exception e)
    {
        Console.WriteLine(
            "Unknown encountered on server. Message:'{0}' when writing an
object"
            , e.Message);
    }
}
}

```

Java

The following example creates two objects. The first object has a text string as data, and the second object is a file. The example creates the first object by specifying the bucket name, object key, and text data directly in a call to `AmazonS3Client.putObject()`. The example creates the second object by using a `PutObjectRequest` that specifies the bucket name, object key, and file path. The `PutObjectRequest` also specifies the `ContentType` header and title metadata.

For instructions on creating and testing a working sample, see [Testing the Amazon S3 Java Code Examples \(p. 1192\)](#).

```
import com.amazonaws.AmazonServiceException;
import com.amazonaws.SdkClientException;
import com.amazonaws.regions.Regions;
import com.amazonaws.services.s3.AmazonS3;
import com.amazonaws.services.s3.AmazonS3ClientBuilder;
import com.amazonaws.services.s3.model.ObjectMetadata;
import com.amazonaws.services.s3.model.PutObjectRequest;

import java.io.File;
import java.io.IOException;

public class UploadObject {

    public static void main(String[] args) throws IOException {
        Regions clientRegion = Regions.DEFAULT_REGION;
        String bucketName = "*** Bucket name ***";
        String stringObjKeyName = "*** String object key name ***";
        String fileObjKeyName = "*** File object key name ***";
        String fileName = "*** Path to file to upload ***";

        try {
            //This code expects that you have AWS credentials set up per:
            // https://docs.aws.amazon.com/sdk-for-java/v1/developer-guide/setup-credentials.html
            AmazonS3 s3Client = AmazonS3ClientBuilder.standard()
                .withRegion(clientRegion)
                .build();

            // Upload a text string as a new object.
            s3Client.putObject(bucketName, stringObjKeyName, "Uploaded String Object");

            // Upload a file as a new object with ContentType and title specified.
            PutObjectRequest request = new PutObjectRequest(bucketName, fileObjKeyName,
new File(fileName));
            ObjectMetadata metadata = new ObjectMetadata();
            metadata.setContentType("plain/text");
            metadata.addUserMetadata("title", "someTitle");
            request.setMetadata(metadata);
            s3Client.putObject(request);
        } catch (AmazonServiceException e) {
            // The call was transmitted successfully, but Amazon S3 couldn't process
            // it, so it returned an error response.
            e.printStackTrace();
        } catch (SdkClientException e) {
            // Amazon S3 couldn't be contacted for a response, or the client
            // couldn't parse the response from Amazon S3.
            e.printStackTrace();
        }
    }
}
```

JavaScript

The following example upload an existing file to an Amazon S3 bucket in a specific Region.

```
// Import required AWS SDK clients and commands for Node.js.
import { PutObjectCommand } from "@aws-sdk/client-s3";
import { s3Client } from "./libs/s3Client.js"; // Helper function that creates an
Amazon S3 service client module.
import {path} from "path";
import {fs} from "fs";
```

```
const file = "OBJECT_PATH_AND_NAME"; // Path to and name of object. For example '../myFiles/index.js'.
const fileStream = fs.createReadStream(file);

// Set the parameters
export const uploadParams = {
  Bucket: "BUCKET_NAME",
  // Add the required 'Key' parameter using the 'path' module.
  Key: path.basename(file),
  // Add the required 'Body' parameter
  Body: fileStream,
};

// Upload file to specified bucket.
export const run = async () => {
  try {
    const data = await s3Client.send(new PutObjectCommand(uploadParams));
    console.log("Success", data);
    return data; // For unit tests.
  } catch (err) {
    console.log("Error", err);
  }
};
run();
```

PHP

This topic guides you through using classes from the AWS SDK for PHP to upload an object of up to 5 GB in size. For larger files, you must use multipart upload API. For more information, see [Uploading and copying objects using multipart upload \(p. 167\)](#).

This topic assumes that you are already following the instructions for [Using the AWS SDK for PHP](#) and [Running PHP Examples \(p. 1193\)](#) and have the AWS SDK for PHP properly installed.

Example — Creating an object in an Amazon S3 bucket by uploading data

The following PHP example creates an object in a specified bucket by uploading data using the `putObject()` method. For information about running the PHP examples in this guide, see [Running PHP Examples \(p. 1194\)](#).

```
require 'vendor/autoload.php';

use Aws\S3\S3Client;
use Aws\S3\Exception\S3Exception;

$bucket = '*** Your Bucket Name ***';
$keyname = '*** Your Object Key ***';

$s3 = new S3Client([
    'version' => 'latest',
    'region'  => 'us-east-1'
]);

try {
    // Upload data.
    $result = $s3->putObject([
        'Bucket'  => $bucket,
        'Key'     => $keyname,
        'Body'    => 'Hello, world!',
        'ACL'     => 'public-read'
    ]);
}
```

```
// Print the URL to the object.  
echo $result['ObjectURL'] . PHP_EOL;  
} catch (S3Exception $e) {  
    echo $e->getMessage() . PHP_EOL;  
}
```

Ruby

The AWS SDK for Ruby - Version 3 has two ways of uploading an object to Amazon S3. The first uses a managed file uploader, which makes it easy to upload files of any size from disk. To use the managed file uploader method:

1. Create an instance of the `Aws::S3::Resource` class.
2. Reference the target object by bucket name and key. Objects live in a bucket and have unique keys that identify each object.
3. Call `#upload_file` on the object.

Example

```
require "aws-sdk-s3"  
  
# Wraps Amazon S3 object actions.  
class ObjectUploadFileWrapper  
    attr_reader :object  
  
    # @param object [Aws::S3::Object] An existing Amazon S3 object.  
    def initialize(object)  
        @object = object  
    end  
  
    # Uploads a file to an Amazon S3 object by using a managed uploader.  
    #  
    # @param file_path [String] The path to the file to upload.  
    # @return [Boolean] True when the file is uploaded; otherwise false.  
    def upload_file(file_path)  
        @object.upload_file(file_path)  
        true  
    rescue Aws::Errors::ServiceError => e  
        puts "Couldn't upload file #{file_path} to #{@object.key}. Here's why:  
#{e.message}"  
        false  
    end  
end  
  
def run_demo  
    bucket_name = "doc-example-bucket"  
    object_key = "my-uploaded-file"  
    file_path = "object_upload_file.rb"  
  
    wrapper = ObjectUploadFileWrapper.new(Aws::S3::Object.new(bucket_name, object_key))  
    return unless wrapper.upload_file(file_path)  
  
    puts "File #{file_path} successfully uploaded to #{bucket_name}:#{object_key}."  
end  
  
run_demo if $PROGRAM_NAME == __FILE__
```

The second way that AWS SDK for Ruby - Version 3 can upload an object uses the `#put` method of `Aws::S3::Object`. This is useful if the object is a string or an I/O object that is not a file on disk. To use this method:

1. Create an instance of the `Aws::S3::Resource` class.
2. Reference the target object by bucket name and key.
3. Call `#put`, passing in the string or I/O object.

Example

```
require "aws-sdk-s3"

# Wraps Amazon S3 object actions.
class ObjectPutWrapper
  attr_reader :object

  # @param object [Aws::S3::Object] An existing Amazon S3 object.
  def initialize(object)
    @object = object
  end

  def put_object(source_file_path)
    File.open(source_file_path, "rb") do |file|
      @object.put(body: file)
    end
    true
  rescue Aws::Errors::ServiceError => e
    puts "Couldn't put #{source_file_path} to #{object.key}. Here's why: #{e.message}"
    false
  end
end

def run_demo
  bucket_name = "doc-example-bucket"
  object_key = "my-object-key"
  file_path = "my-local-file.txt"

  wrapper = ObjectPutWrapper.new(Aws::S3::Object.new(bucket_name, object_key))
  success = wrapper.put_object(file_path)
  return unless success

  puts "Put file #{file_path} into #{object_key} in #{bucket_name}."
end

run_demo if $PROGRAM_NAME == __FILE__
```

Using the REST API

You can send REST requests to upload an object. You can send a PUT request to upload data in a single operation. For more information, see [PUT Object](#).

Using the AWS CLI

You can send a `PUT` request to upload an object of up to 5 GB in a single operation. For more information, see the [PutObject](#) example in the *AWS CLI Command Reference*.

Uploading and copying objects using multipart upload

Multipart upload allows you to upload a single object as a set of parts. Each part is a contiguous portion of the object's data. You can upload these object parts independently and in any order. If transmission of any part fails, you can retransmit that part without affecting other parts. After all parts of your object are uploaded, Amazon S3 assembles these parts and creates the object. In general, when your object size reaches 100 MB, you should consider using multipart uploads instead of uploading the object in a single operation.

Using multipart upload provides the following advantages:

- **Improved throughput** – You can upload parts in parallel to improve throughput.
- **Quick recovery from any network issues** – Smaller part size minimizes the impact of restarting a failed upload due to a network error.
- **Pause and resume object uploads** – You can upload object parts over time. After you initiate a multipart upload, there is no expiry; you must explicitly complete or stop the multipart upload.
- **Begin an upload before you know the final object size** – You can upload an object as you are creating it.

We recommend that you use multipart upload in the following ways:

- If you're uploading large objects over a stable high-bandwidth network, use multipart upload to maximize the use of your available bandwidth by uploading object parts in parallel for multi-threaded performance.
- If you're uploading over a spotty network, use multipart upload to increase resiliency to network errors by avoiding upload restarts. When using multipart upload, you need to retry uploading only the parts that are interrupted during the upload. You don't need to restart uploading your object from the beginning.

Multipart upload process

Multipart upload is a three-step process: You initiate the upload, you upload the object parts, and after you have uploaded all the parts, you complete the multipart upload. Upon receiving the complete multipart upload request, Amazon S3 constructs the object from the uploaded parts, and you can then access the object just as you would any other object in your bucket.

You can list all of your in-progress multipart uploads or get a list of the parts that you have uploaded for a specific multipart upload. Each of these operations is explained in this section.

Multipart upload initiation

When you send a request to initiate a multipart upload, Amazon S3 returns a response with an upload ID, which is a unique identifier for your multipart upload. You must include this upload ID whenever you upload parts, list the parts, complete an upload, or stop an upload. If you want to provide any metadata describing the object being uploaded, you must provide it in the request to initiate multipart upload.

Parts upload

When uploading a part, in addition to the upload ID, you must specify a part number. You can choose any part number between 1 and 10,000. A part number uniquely identifies a part and its position in the object you are uploading. The part number that you choose doesn't need to be in a consecutive sequence.

(for example, it can be 1, 5, and 14). If you upload a new part using the same part number as a previously uploaded part, the previously uploaded part is overwritten.

Whenever you upload a part, Amazon S3 returns an *entity tag (ETag)* header in its response. For each part upload, you must record the part number and the ETag value. You must include these values in the subsequent request to complete the multipart upload.

Note

After you initiate a multipart upload and upload one or more parts, you must either complete or stop the multipart upload to stop getting charged for storage of the uploaded parts. Only *after* you either complete or stop a multipart upload will Amazon S3 free up the parts storage and stop charging you for the parts storage.

After stopping a multipart upload, you cannot upload any part using that upload ID again. If any part uploads were in-progress, they can still succeed or fail even after you stop the upload. To make sure you free all storage consumed by all parts, you must stop a multipart upload only after all part uploads have been completed.

Multipart upload completion

When you complete a multipart upload, Amazon S3 creates an object by concatenating the parts in ascending order based on the part number. If any object metadata was provided in the *initiate multipart upload* request, Amazon S3 associates that metadata with the object. After a successful *complete* request, the parts no longer exist.

Your *complete multipart upload* request must include the upload ID and a list of both part numbers and corresponding ETag values. The Amazon S3 response includes an ETag that uniquely identifies the combined object data. This ETag is not necessarily an MD5 hash of the object data.

Sample multipart upload calls

For this example, assume that you are generating a multipart upload for a 100 GB file. In this case, you would have the following API calls for the entire process. There would be a total of 1002 API calls.

- A [CreateMultipartUpload](#) call to start the process.
- 1000 individual [UploadPart](#) calls, each uploading a part of 100 MB, for a total size of 100 GB.
- A [CompleteMultipartUpload](#) call to finish the process.

Multipart upload listings

You can list the parts of a specific multipart upload or all in-progress multipart uploads. The list parts operation returns the parts information that you have uploaded for a specific multipart upload. For each list parts request, Amazon S3 returns the parts information for the specified multipart upload, up to a maximum of 1,000 parts. If there are more than 1,000 parts in the multipart upload, you must send a series of list part requests to retrieve all the parts. Note that the returned list of parts doesn't include parts that haven't finished uploading. Using the *list multipart uploads* operation, you can obtain a list of multipart uploads that are in progress.

An in-progress multipart upload is an upload that you have initiated, but have not yet completed or stopped. Each request returns at most 1,000 multipart uploads. If there are more than 1,000 multipart uploads in progress, you must send additional requests to retrieve the remaining multipart uploads. Use the returned listing only for verification. Do not use the result of this listing when sending a *complete multipart upload* request. Instead, maintain your own list of the part numbers that you specified when uploading parts and the corresponding ETag values that Amazon S3 returns.

Checksums with multipart upload operations

When you upload an object to Amazon S3, you can specify a checksum algorithm for Amazon S3 to use. Amazon S3 uses MD5 by default to verify data integrity; however, you can specify an additional

checksum algorithm to use. When using MD5, Amazon S3 calculates the checksum of the entire multipart object after the upload is complete. This checksum is not a checksum of the entire object, but rather a checksum of the checksums for each individual part.

When you instruct Amazon S3 to use additional checksums, Amazon S3 calculates the checksum value for each part and stores the values. You can use the API or SDK to retrieve the checksum value for individual parts by using `GetObject` or `HeadObject`.

Important

If you are using a multipart upload with additional checksums, the multipart part numbers must use consecutive part numbers. When using additional checksums, if you try to complete a multipart upload request with nonconsecutive part numbers, Amazon S3 generates HTTP 500 Internal Server Error error.

For more information about how checksums work with multipart objects, see [Checking object integrity \(p. 215\)](#).

Concurrent multipart upload operations

In a distributed development environment, it is possible for your application to initiate several updates on the same object at the same time. Your application might initiate several multipart uploads using the same object key. For each of these uploads, your application can then upload parts and send a complete upload request to Amazon S3 to create the object. When the buckets have S3 Versioning enabled, completing a multipart upload always creates a new version. For buckets that don't have versioning enabled, it is possible that some other request received between the time when a multipart upload is initiated and when it is completed might take precedence.

Note

It is possible for some other request received between the time you initiated a multipart upload and completed it to take precedence. For example, if another operation deletes a key after you initiate a multipart upload with that key, but before you complete it, the complete multipart upload response might indicate a successful object creation without you ever seeing the object.

Multipart upload and pricing

After you initiate a multipart upload, Amazon S3 retains all the parts until you either complete or stop the upload. Throughout its lifetime, you are billed for all storage, bandwidth, and requests for this multipart upload and its associated parts. If you stop the multipart upload, Amazon S3 deletes upload artifacts and any parts that you have uploaded, and you are no longer billed for them. For more information about pricing, see [Amazon S3 pricing](#).

API support for multipart upload

These libraries provide a high-level abstraction that makes uploading multipart objects easy. However, if your application requires, you can use the REST API directly. The following sections in the *Amazon Simple Storage Service API Reference* describe the REST API for multipart upload.

- [Create Multipart Upload](#)
- [Upload Part](#)
- [Upload Part \(Copy\)](#)
- [Complete Multipart Upload](#)
- [Abort Multipart Upload](#)
- [List Parts](#)
- [List Multipart Uploads](#)

AWS Command Line Interface support for multipart upload

The following topics in the AWS Command Line Interface describe the operations for multipart upload.

- [Initiate Multipart Upload](#)
- [Upload Part](#)
- [Upload Part \(Copy\)](#)
- [Complete Multipart Upload](#)
- [Abort Multipart Upload](#)
- [List Parts](#)
- [List Multipart Uploads](#)

AWS SDK support for multipart upload

You can use an AWS SDKs to upload an object in parts. For a list of AWS SDKs supported by API action see:

- [Create Multipart Upload](#)
- [Upload Part](#)
- [Upload Part \(Copy\)](#)
- [Complete Multipart Upload](#)
- [Abort Multipart Upload](#)
- [List Parts](#)
- [List Multipart Uploads](#)

Multipart upload API and permissions

You must have the necessary permissions to use the multipart upload operations. You can use access control lists (ACLs), the bucket policy, or the user policy to grant individuals permissions to perform these operations. The following table lists the required permissions for various multipart upload operations when using ACLs, a bucket policy, or a user policy.

Action	Required permissions
Create Multipart Upload	You must be allowed to perform the <code>s3:PutObject</code> action on an object to create multipart upload. The bucket owner can allow other principals to perform the <code>s3:PutObject</code> action.
Initiate Multipart Upload	You must be allowed to perform the <code>s3:PutObject</code> action on an object to initiate multipart upload. The bucket owner can allow other principals to perform the <code>s3:PutObject</code> action.
Initiator	Container element that identifies who initiated the multipart upload. If the initiator is an AWS account, this element provides the same information as the Owner element. If the initiator is an IAM User, this element provides the user ARN and display name.

Action	Required permissions
Upload Part	<p>You must be allowed to perform the <code>s3:PutObject</code> action on an object to upload a part.</p> <p>The bucket owner must allow the initiator to perform the <code>s3:PutObject</code> action on an object in order for the initiator to upload a part for that object.</p>
Upload Part (Copy)	<p>You must be allowed to perform the <code>s3:PutObject</code> action on an object to upload a part. Because you are uploading a part from an existing object, you must be allowed <code>s3:GetObject</code> on the source object.</p> <p>For the initiator to upload a part for an object, the owner of the bucket must allow the initiator to perform the <code>s3:PutObject</code> action on the object.</p>
Complete Multipart Upload	<p>You must be allowed to perform the <code>s3:PutObject</code> action on an object to complete a multipart upload.</p> <p>The bucket owner must allow the initiator to perform the <code>s3:PutObject</code> action on an object in order for the initiator to complete a multipart upload for that object.</p>
Stop Multipart Upload	<p>You must be allowed to perform the <code>s3:AbortMultipartUpload</code> action to stop a multipart upload.</p> <p>By default, the bucket owner and the initiator of the multipart upload are allowed to perform this action as a part of IAM and bucket policies. If the initiator is an IAM user, that user's AWS account is also allowed to stop that multipart upload. With VPC endpoint policies, the initiator of the multipart upload does not automatically gain the permission to perform the <code>s3:AbortMultipartUpload</code> action.</p> <p>In addition to these defaults, the bucket owner can allow other principals to perform the <code>s3:AbortMultipartUpload</code> action on an object. The bucket owner can deny any principal the ability to perform the <code>s3:AbortMultipartUpload</code> action.</p>
List Parts	<p>You must be allowed to perform the <code>s3>ListMultipartUploadParts</code> action to list parts in a multipart upload.</p> <p>By default, the bucket owner has permission to list parts for any multipart upload to the bucket. The initiator of the multipart upload has the permission to list parts of the specific multipart upload. If the multipart upload initiator is an IAM user, the AWS account controlling that IAM user also has permission to list parts of that upload.</p> <p>In addition to these defaults, the bucket owner can allow other principals to perform the <code>s3>ListMultipartUploadParts</code> action on an object. The bucket owner can also deny any principal the ability to perform the <code>s3>ListMultipartUploadParts</code> action.</p>
List Multipart Uploads	<p>You must be allowed to perform the <code>s3>ListBucketMultipartUploads</code> action on a bucket to list multipart uploads in progress to that bucket.</p> <p>In addition to the default, the bucket owner can allow other principals to perform the <code>s3>ListBucketMultipartUploads</code> action on the bucket.</p>

Action	Required permissions
AWS KMS Encrypt and Decrypt related permissions	<p>To perform a multipart upload with encryption using an AWS Key Management Service (AWS KMS) KMS key, the requester must have permission to the <code>kms:Decrypt</code> and <code>kms:GenerateDataKey</code> actions on the key. These permissions are required because Amazon S3 must decrypt and read data from the encrypted file parts before it completes the multipart upload.</p> <p>For more information, see Uploading a large file to Amazon S3 with encryption using an AWS KMS key in the <i>AWS Knowledge Center</i>.</p> <p>If your IAM user or role is in the same AWS account as the KMS key, then you must have these permissions on the key policy. If your IAM user or role belongs to a different account than the KMS key, then you must have the permissions on both the key policy and your IAM user or role.</p>

For information on the relationship between ACL permissions and permissions in access policies, see [Mapping of ACL permissions and access policy permissions \(p. 558\)](#). For information on IAM users, go to [Working with Users and Groups](#).

Topics

- [Configuring a bucket lifecycle policy to abort incomplete multipart uploads \(p. 172\)](#)
- [Uploading an object using multipart upload \(p. 174\)](#)
- [Uploading a directory using the high-level .NET TransferUtility class \(p. 187\)](#)
- [Listing multipart uploads \(p. 188\)](#)
- [Tracking a multipart upload \(p. 190\)](#)
- [Aborting a multipart upload \(p. 193\)](#)
- [Copying an object using multipart upload \(p. 196\)](#)
- [Amazon S3 multipart upload limits \(p. 201\)](#)

Configuring a bucket lifecycle policy to abort incomplete multipart uploads

As a best practice, we recommend you configure a lifecycle rule using the `AbortIncompleteMultipartUpload` action to minimize your storage costs. For more information about aborting a multipart upload, see [Aborting a multipart upload \(p. 193\)](#).

Amazon S3 supports a bucket lifecycle rule that you can use to direct Amazon S3 to stop multipart uploads that don't complete within a specified number of days after being initiated. When a multipart upload is not completed within the time frame, it becomes eligible for an abort operation and Amazon S3 stops the multipart upload (and deletes the parts associated with the multipart upload).

The following is an example lifecycle configuration that specifies a rule with the `AbortIncompleteMultipartUpload` action.

```
<LifecycleConfiguration>
  <Rule>
    <ID>sample-rule</ID>
    <Prefix></Prefix>
    <Status>Enabled</Status>
    <AbortIncompleteMultipartUpload>
      <DaysAfterInitiation>7</DaysAfterInitiation>
    </AbortIncompleteMultipartUpload>
```

```
</Rule>  
</LifecycleConfiguration>
```

In the example, the rule does not specify a value for the `Prefix` element ([object key name prefix](#)). Therefore, it applies to all objects in the bucket for which you initiated multipart uploads. Any multipart uploads that were initiated and did not complete within seven days become eligible for an abort operation. The abort action has no effect on completed multipart uploads.

For more information about the bucket lifecycle configuration, see [Managing your storage lifecycle \(p. 701\)](#).

Note

If the multipart upload is completed within the number of days specified in the rule, the `AbortIncompleteMultipartUpload` lifecycle action does not apply (that is, Amazon S3 does not take any action). Also, this action does not apply to objects. No objects are deleted by this lifecycle action.

The following `put-bucket-lifecycle-configuration` CLI command adds the lifecycle configuration for the specified bucket.

```
$ aws s3api put-bucket-lifecycle-configuration \
  --bucket bucketname \
  --lifecycle-configuration filename-containing-lifecycle-configuration
```

To test the CLI command, do the following:

1. Set up the AWS CLI. For instructions, see [Developing with Amazon S3 using the AWS CLI \(p. 1183\)](#).
2. Save the following example lifecycle configuration in a file (`lifecycle.json`). The example configuration specifies empty prefix and therefore it applies to all objects in the bucket. You can specify a prefix to restrict the policy to a subset of objects.

```
{
  "Rules": [
    {
      "ID": "Test Rule",
      "Status": "Enabled",
      "Filter": {
        "Prefix": ""
      },
      "AbortIncompleteMultipartUpload": {
        "DaysAfterInitiation": 7
      }
    }
  ]
}
```

3. Run the following CLI command to set lifecycle configuration on your bucket.

```
aws s3api put-bucket-lifecycle-configuration \
--bucket bucketname \
--lifecycle-configuration file://lifecycle.json
```

4. To verify, retrieve the lifecycle configuration using the `get-bucket-lifecycle` CLI command.

```
aws s3api get-bucket-lifecycle \
--bucket bucketname
```

5. To delete the lifecycle configuration, use the `delete-bucket-lifecycle` CLI command.

```
aws s3api delete-bucket-lifecycle \  
--bucket bucketname
```

Uploading an object using multipart upload

You can use the multipart upload to programmatically upload a single object to Amazon S3.

For more information, see the following sections.

Using the AWS SDKs (high-level API)

The AWS SDK exposes a high-level API, called `TransferManager`, that simplifies multipart uploads. For more information, see [Uploading and copying objects using multipart upload \(p. 167\)](#).

You can upload data from a file or a stream. You can also set advanced options, such as the part size you want to use for the multipart upload, or the number of concurrent threads you want to use when uploading the parts. You can also set optional object properties, the storage class, or the access control list (ACL). You use the `PutObjectRequest` and the `TransferManagerConfiguration` classes to set these advanced options.

When possible, `TransferManager` tries to use multiple threads to upload multiple parts of a single upload at once. When dealing with large content sizes and high bandwidth, this can increase throughput significantly.

In addition to file-upload functionality, the `TransferManager` class enables you to stop an in-progress multipart upload. An upload is considered to be in progress after you initiate it and until you complete or stop it. The `TransferManager` stops all in-progress multipart uploads on a specified bucket that were initiated before a specified date and time.

If you need to pause and resume multipart uploads, vary part sizes during the upload, or do not know the size of the data in advance, use the low-level PHP API. For more information about multipart uploads, including additional functionality offered by the low-level API methods, see [Using the AWS SDKs \(low-level-level API\) \(p. 180\)](#).

Java

The following example loads an object using the high-level multipart upload Java API (the `TransferManager` class). For instructions on creating and testing a working sample, see [Testing the Amazon S3 Java Code Examples \(p. 1192\)](#).

```
import com.amazonaws.AmazonServiceException;  
import com.amazonaws.SdkClientException;  
import com.amazonaws.auth.profile.ProfileCredentialsProvider;  
import com.amazonaws.regions.Regions;  
import com.amazonaws.services.s3.AmazonS3;  
import com.amazonaws.services.s3.AmazonS3ClientBuilder;  
import com.amazonaws.services.s3.transfer.TransferManager;  
import com.amazonaws.services.s3.transfer.TransferManagerBuilder;  
import com.amazonaws.services.s3.transfer.Upload;  
  
import java.io.File;  
  
public class HighLevelMultipartUpload {  
  
    public static void main(String[] args) throws Exception {  
        Regions clientRegion = Regions.DEFAULT_REGION;  
        String bucketName = "*** Bucket name ***";
```

```
String keyName = "*** Object key ***";
String filePath = "*** Path for file to upload ***;

try {
    AmazonS3 s3Client = AmazonS3ClientBuilder.standard()
        .withRegion(clientRegion)
        .withCredentials(new ProfileCredentialsProvider())
        .build();
    TransferManager tm = TransferManagerBuilder.standard()
        .withS3Client(s3Client)
        .build();

    // TransferManager processes all transfers asynchronously,
    // so this call returns immediately.
    Upload upload = tm.upload(bucketName, keyName, new File(filePath));
    System.out.println("Object upload started");

    // Optionally, wait for the upload to finish before continuing.
    upload.waitForCompletion();
    System.out.println("Object upload complete");
} catch (AmazonServiceException e) {
    // The call was transmitted successfully, but Amazon S3 couldn't process
    // it, so it returned an error response.
    e.printStackTrace();
} catch (SdkClientException e) {
    // Amazon S3 couldn't be contacted for a response, or the client
    // couldn't parse the response from Amazon S3.
    e.printStackTrace();
}
}
```

.NET

To upload a file to an S3 bucket, use the `TransferUtility` class. When uploading data from a file, you must provide the object's key name. If you don't, the API uses the file name for the key name. When uploading data from a stream, you must provide the object's key name.

To set advanced upload options—such as the part size, the number of threads when uploading the parts concurrently, metadata, the storage class, or ACL—use the `TransferUtilityUploadRequest` class.

The following C# example uploads a file to an Amazon S3 bucket in multiple parts. It shows how to use various `TransferUtility.Upload` overloads to upload a file. Each successive call to upload replaces the previous upload. For information about the example's compatibility with a specific version of the AWS SDK for .NET and instructions for creating and testing a working sample, see [Running the Amazon S3 .NET Code Examples \(p. 1193\)](#).

```
using Amazon;
using Amazon.S3;
using Amazon.S3.Transfer;
using System;
using System.IO;
using System.Threading.Tasks;

namespace Amazon.DocSamples.S3
{
    class UploadFileMPUHighLevelAPITest
    {
        private const string bucketName = "*** provide bucket name ***";
        private const string keyName = "*** provide a name for the uploaded object
***".
```

```
    private const string filePath = "*** provide the full path name of the file to
upload ***";
    // Specify your bucket region (an example region is shown).
    private static readonly RegionEndpoint bucketRegion = RegionEndpoint.USWest2;
    private static IAmazonS3 s3Client;

    public static void Main()
    {
        s3Client = new AmazonS3Client(bucketRegion);
        UploadFileAsync().Wait();
    }

    private static async Task UploadFileAsync()
    {
        try
        {
            var fileTransferUtility =
                new TransferUtility(s3Client);

            // Option 1. Upload a file. The file name is used as the object key
name.
            await fileTransferUtility.UploadAsync(filePath, bucketName);
            Console.WriteLine("Upload 1 completed");

            // Option 2. Specify object key name explicitly.
            await fileTransferUtility.UploadAsync(filePath, bucketName, keyName);
            Console.WriteLine("Upload 2 completed");

            // Option 3. Upload data from a type of System.IO.Stream.
            using (var fileToUpload =
                new FileStream(filePath, FileMode.Open, FileAccess.Read))
            {
                await fileTransferUtility.UploadAsync(fileToUpload,
                                                    bucketName, keyName);
            }
            Console.WriteLine("Upload 3 completed");

            // Option 4. Specify advanced settings.
            var fileTransferUtilityRequest = new TransferUtilityUploadRequest
            {
                BucketName = bucketName,
                FilePath = filePath,
                StorageClass = S3StorageClass.StandardInfrequentAccess,
                PartSize = 6291456, // 6 MB.
                Key = keyName,
                CannedACL = S3CannedACL.PublicRead
            };
            fileTransferUtilityRequest.Metadata.Add("param1", "Value1");
            fileTransferUtilityRequest.Metadata.Add("param2", "Value2");

            await fileTransferUtility.UploadAsync(fileTransferUtilityRequest);
            Console.WriteLine("Upload 4 completed");
        }
        catch (AmazonS3Exception e)
        {
            Console.WriteLine("Error encountered on server. Message:'{0}' when
writing an object", e.Message);
        }
        catch (Exception e)
        {
            Console.WriteLine("Unknown encountered on server. Message:'{0}' when
writing an object", e.Message);
        }
    }
}
```

```
}
```

PHP

This topic explains how to use the high-level `Aws\S3\Model\MultipartUpload\UploadBuilder` class from the AWS SDK for PHP for multipart file uploads. It assumes that you are already following the instructions for [Using the AWS SDK for PHP and Running PHP Examples \(p. 1193\)](#) and have the AWS SDK for PHP properly installed.

The following PHP example uploads a file to an Amazon S3 bucket. The example demonstrates how to set parameters for the `MultipartUploader` object.

For information about running the PHP examples in this guide, see [Running PHP Examples \(p. 1194\)](#).

```
require 'vendor/autoload.php';

use Aws\Common\Exception\MultipartUploadException;
use Aws\S3\MultipartUploader;
use Aws\S3\S3Client;

$bucket = '*** Your Bucket Name ***';
$keyname = '*** Your Object Key ***';

$s3 = new S3Client([
    'version' => 'latest',
    'region'  => 'us-east-1'
]);

// Prepare the upload parameters.
uploader = new MultipartUploader($s3, '/path/to/large/file.zip', [
    'bucket' => $bucket,
    'key'     => $keyname
]);

// Perform the upload.
try {
    $result = $uploader->upload();
    echo "Upload complete: {$result['ObjectURL']}" . PHP_EOL;
} catch (MultipartUploadException $e) {
    echo $e->getMessage() . PHP_EOL;
}
```

Python

The following example loads an object using the high-level multipart upload Python API (the `TransferManager` class).

```
import sys
import threading

import boto3
from boto3.s3.transfer import TransferConfig

MB = 1024 * 1024
s3 = boto3.resource('s3')

class TransferCallback:
    """
    Handle callbacks from the transfer manager.

```

```

The transfer manager periodically calls the __call__ method throughout
the upload and download process so that it can take action, such as
displaying progress to the user and collecting data about the transfer.
"""

def __init__(self, target_size):
    self._target_size = target_size
    self._total_transferred = 0
    self._lock = threading.Lock()
    self.thread_info = {}

def __call__(self, bytes_transferred):
    """
    The callback method that is called by the transfer manager.

    Display progress during file transfer and collect per-thread transfer
    data. This method can be called by multiple threads, so shared instance
    data is protected by a thread lock.
    """
    thread = threading.current_thread()
    with self._lock:
        self._total_transferred += bytes_transferred
        if thread.ident not in self.thread_info.keys():
            self.thread_info[thread.ident] = bytes_transferred
        else:
            self.thread_info[thread.ident] += bytes_transferred

        target = self._target_size * MB
        sys.stdout.write(
            f"\r{self._total_transferred} of {target} transferred "
            f"({(self._total_transferred / target) * 100:.2f}%)")
        sys.stdout.flush()

def upload_with_default_configuration(local_file_path, bucket_name,
                                      object_key, file_size_mb):
    """
    Upload a file from a local folder to an Amazon S3 bucket, using the default
    configuration.
    """
    transfer_callback = TransferCallback(file_size_mb)
    s3.Bucket(bucket_name).upload_file(
        local_file_path,
        object_key,
        Callback=transfer_callback)
    return transfer_callback.thread_info

def upload_with_chunksize_and_meta(local_file_path, bucket_name, object_key,
                                   file_size_mb, metadata=None):
    """
    Upload a file from a local folder to an Amazon S3 bucket, setting a
    multipart chunk size and adding metadata to the Amazon S3 object.

    The multipart chunk size controls the size of the chunks of data that are
    sent in the request. A smaller chunk size typically results in the transfer
    manager using more threads for the upload.

    The metadata is a set of key-value pairs that are stored with the object
    in Amazon S3.
    """
    transfer_callback = TransferCallback(file_size_mb)

    config = TransferConfig(multipart_chunksize=1 * MB)
    extra_args = {'Metadata': metadata} if metadata else None
    s3.Bucket(bucket_name).upload_file(

```

```
local_file_path,
object_key,
Config=config,
ExtraArgs=extra_args,
Callback=transfer_callback)
return transfer_callback.thread_info

def upload_with_high_threshold(local_file_path, bucket_name, object_key,
                               file_size_mb):
    """
    Upload a file from a local folder to an Amazon S3 bucket, setting a
    multipart threshold larger than the size of the file.

    Setting a multipart threshold larger than the size of the file results
    in the transfer manager sending the file as a standard upload instead of
    a multipart upload.
    """
    transfer_callback = TransferCallback(file_size_mb)
    config = TransferConfig(multipart_threshold=file_size_mb * 2 * MB)
    s3.Bucket(bucket_name).upload_file(
        local_file_path,
        object_key,
        Config=config,
        Callback=transfer_callback)
    return transfer_callback.thread_info

def upload_with_sse(local_file_path, bucket_name, object_key,
                    file_size_mb, sse_key=None):
    """
    Upload a file from a local folder to an Amazon S3 bucket, adding server-side
    encryption with customer-provided encryption keys to the object.

    When this kind of encryption is specified, Amazon S3 encrypts the object
    at rest and allows downloads only when the expected encryption key is
    provided in the download request.
    """
    transfer_callback = TransferCallback(file_size_mb)
    if sse_key:
        extra_args = {
            'SSECustomerAlgorithm': 'AES256',
            'SSECustomerKey': sse_key}
    else:
        extra_args = None
    s3.Bucket(bucket_name).upload_file(
        local_file_path,
        object_key,
        ExtraArgs=extra_args,
        Callback=transfer_callback)
    return transfer_callback.thread_info

def download_with_default_configuration(bucket_name, object_key,
                                         download_file_path, file_size_mb):
    """
    Download a file from an Amazon S3 bucket to a local folder, using the
    default configuration.
    """
    transfer_callback = TransferCallback(file_size_mb)
    s3.Bucket(bucket_name).Object(object_key).download_file(
        download_file_path,
        Callback=transfer_callback)
    return transfer_callback.thread_info
```

```
def download_with_single_thread(bucket_name, object_key,
                                download_file_path, file_size_mb):
    """
    Download a file from an Amazon S3 bucket to a local folder, using a
    single thread.
    """
    transfer_callback = TransferCallback(file_size_mb)
    config = TransferConfig(use_threads=False)
    s3.Bucket(bucket_name).Object(object_key).download_file(
        download_file_path,
        Config=config,
        Callback=transfer_callback)
    return transfer_callback.thread_info

def download_with_high_threshold(bucket_name, object_key,
                                  download_file_path, file_size_mb):
    """
    Download a file from an Amazon S3 bucket to a local folder, setting a
    multipart threshold larger than the size of the file.

    Setting a multipart threshold larger than the size of the file results
    in the transfer manager sending the file as a standard download instead
    of a multipart download.
    """
    transfer_callback = TransferCallback(file_size_mb)
    config = TransferConfig(multipart_threshold=file_size_mb * 2 * MB)
    s3.Bucket(bucket_name).Object(object_key).download_file(
        download_file_path,
        Config=config,
        Callback=transfer_callback)
    return transfer_callback.thread_info

def download_with_sse(bucket_name, object_key, download_file_path,
                      file_size_mb, sse_key):
    """
    Download a file from an Amazon S3 bucket to a local folder, adding a
    customer-provided encryption key to the request.

    When this kind of encryption is specified, Amazon S3 encrypts the object
    at rest and allows downloads only when the expected encryption key is
    provided in the download request.
    """
    transfer_callback = TransferCallback(file_size_mb)

    if sse_key:
        extra_args = {
            'SSECustomerAlgorithm': 'AES256',
            'SSECustomerKey': sse_key}
    else:
        extra_args = None
    s3.Bucket(bucket_name).Object(object_key).download_file(
        download_file_path,
        ExtraArgs=extra_args,
        Callback=transfer_callback)
    return transfer_callback.thread_info
```

Using the AWS SDKs (low-level API)

The AWS SDK exposes a low-level API that closely resembles the Amazon S3 REST API for multipart uploads (see [Uploading and copying objects using multipart upload \(p. 167\)](#)). Use the low-level API when you need to pause and resume multipart uploads, vary part sizes during the upload, or do not

know the size of the upload data in advance. When you don't have these requirements, use the high-level API (see [Using the AWS SDKs \(high-level API\) \(p. 174\)](#)).

Java

The following example shows how to use the low-level Java classes to upload a file. It performs the following steps:

- Initiates a multipart upload using the `AmazonS3Client.initiateMultipartUpload()` method, and passes in an `InitiateMultipartUploadRequest` object.
- Saves the upload ID that the `AmazonS3Client.initiateMultipartUpload()` method returns. You provide this upload ID for each subsequent multipart upload operation.
- Uploads the parts of the object. For each part, you call the `AmazonS3Client.uploadPart()` method. You provide part upload information using an `UploadPartRequest` object.
- For each part, saves the ETag from the response of the `AmazonS3Client.uploadPart()` method in a list. You use the ETag values to complete the multipart upload.
- Calls the `AmazonS3Client.completeMultipartUpload()` method to complete the multipart upload.

Example

For instructions on creating and testing a working sample, see [Testing the Amazon S3 Java Code Examples \(p. 1192\)](#).

```
import com.amazonaws.AmazonServiceException;
import com.amazonaws.SdkClientException;
import com.amazonaws.auth.profile.ProfileCredentialsProvider;
import com.amazonaws.regions.Regions;
import com.amazonaws.services.s3.AmazonS3;
import com.amazonaws.services.s3.AmazonS3ClientBuilder;
import com.amazonaws.services.s3.model.*;

import java.io.File;
import java.io.IOException;
import java.util.ArrayList;
import java.util.List;

public class LowLevelMultipartUpload {

    public static void main(String[] args) throws IOException {
        Regions clientRegion = Regions.DEFAULT_REGION;
        String bucketName = "*** Bucket name ***";
        String keyName = "*** Key name ***";
        String filePath = "*** Path to file to upload ***";

        File file = new File(filePath);
        long contentLength = file.length();
        long partSize = 5 * 1024 * 1024; // Set part size to 5 MB.

        try {
            AmazonS3 s3Client = AmazonS3ClientBuilder.standard()
                .withRegion(clientRegion)
                .withCredentials(new ProfileCredentialsProvider())
                .build();

            // Create a list of ETag objects. You retrieve ETags for each object part
            uploaded, // then, after each individual part has been uploaded, pass the list of
            ETags to
        }
    }
}
```

```

// the request to complete the upload.
List<PartETag> partETags = new ArrayList<PartETag>();

// Initiate the multipart upload.
InitiateMultipartUploadRequest initRequest = new
InitiateMultipartUploadRequest(bucketName, keyName);
InitiateMultipartUploadResult initResponse =
s3Client.initiateMultipartUpload(initRequest);

// Upload the file parts.
long filePosition = 0;
for (int i = 1; filePosition < contentLength; i++) {
    // Because the last part could be less than 5 MB, adjust the part size
as needed.
    partSize = Math.min(partSize, (contentLength - filePosition));

    // Create the request to upload a part.
    UploadPartRequest uploadRequest = new UploadPartRequest()
        .withBucketName(bucketName)
        .withKey(keyName)
        .withUploadId(initResponse.getUploadId())
        .withPartNumber(i)
        .withFileOffset(filePosition)
        .withFile(file)
        .withPartSize(partSize);

    // Upload the part and add the response's ETag to our list.
    UploadPartResult uploadResult = s3Client.uploadPart(uploadRequest);
    partETags.add(uploadResult.getPartETag());

    filePosition += partSize;
}

// Complete the multipart upload.
CompleteMultipartUploadRequest compRequest = new
CompleteMultipartUploadRequest(bucketName, keyName,
                               initResponse.getUploadId(), partETags);
s3Client.completeMultipartUpload(compRequest);
} catch (AmazonServiceException e) {
    // The call was transmitted successfully, but Amazon S3 couldn't process
    // it, so it returned an error response.
    e.printStackTrace();
} catch (SdkClientException e) {
    // Amazon S3 couldn't be contacted for a response, or the client
    // couldn't parse the response from Amazon S3.
    e.printStackTrace();
}
}
}

```

.NET

The following C# example shows how to use the low-level AWS SDK for .NET multipart upload API to upload a file to an S3 bucket. For information about Amazon S3 multipart uploads, see [Uploading and copying objects using multipart upload \(p. 167\)](#).

Note

Note When you use the AWS SDK for .NET API to upload large objects, a timeout might occur while data is being written to the request stream. You can set an explicit timeout using the `UploadPartRequest`.

The following C# example uploads a file to an S3 bucket using the low-level multipart upload API. For information about the example's compatibility with a specific version of the AWS SDK for .NET

and instructions for creating and testing a working sample, see [Running the Amazon S3 .NET Code Examples \(p. 1193\)](#).

```
using Amazon;
using Amazon.Runtime;
using Amazon.S3;
using Amazon.S3.Model;
using System;
using System.Collections.Generic;
using System.IO;
using System.Threading.Tasks;

namespace Amazon.DocSamples.S3
{
    class UploadFileMPULowLevelAPITest
    {
        private const string bucketName = "*** provide bucket name ***";
        private const string keyName = "*** provide a name for the uploaded object ***";
        private const string filePath = "*** provide the full path name of the file to upload ***";
        // Specify your bucket region (an example region is shown).
        private static readonly RegionEndpoint bucketRegion = RegionEndpoint.USWest2;
        private static IAmazonS3 s3Client;

        public static void Main()
        {
            s3Client = new AmazonS3Client(bucketRegion);
            Console.WriteLine("Uploading an object");
            UploadObjectAsync().Wait();
        }

        private static async Task UploadObjectAsync()
        {
            // Create list to store upload part responses.
            List<UploadPartResponse> uploadResponses = new List<UploadPartResponse>();

            // Setup information required to initiate the multipart upload.
            InitiateMultipartUploadRequest initiateRequest = new InitiateMultipartUploadRequest
            {
                BucketName = bucketName,
                Key = keyName
            };

            // Initiate the upload.
            InitiateMultipartUploadResponse initResponse =
                await s3Client.InitiateMultipartUploadAsync(initiateRequest);

            // Upload parts.
            long contentLength = new FileInfo(filePath).Length;
            long partSize = 5 * (long)Math.Pow(2, 20); // 5 MB

            try
            {
                Console.WriteLine("Uploading parts");

                long filePosition = 0;
                for (int i = 1; filePosition < contentLength; i++)
                {
                    UploadPartRequest uploadRequest = new UploadPartRequest
                    {
                        BucketName = bucketName,
                        Key = keyName,
                        UploadId = initResponse.UploadId,
                        PartNumber = i,
                        InputStream = File.OpenRead(filePath),
                        ContentLength = partSize
                    };
                    uploadResponses.Add(await s3Client.UploadPartAsync(uploadRequest));
                    filePosition += partSize;
                }
            }
            catch (AmazonS3Exception e)
            {
                Console.WriteLine("Error uploading part {0}: {1}", i, e.Message);
            }
        }
    }
}
```

```
        PartNumber = i,
        PartSize = partSize,
        FilePosition = filePosition,
        FilePath = filePath
    };

    // Track upload progress.
    uploadRequest.StreamTransferProgress +=
        new
EventHandler<StreamTransferProgressArgs>(UploadPartProgressEventCallback);

    // Upload a part and add the response to our list.
    uploadResponses.Add(await s3Client.UploadPartAsync(uploadRequest));

    filePosition += partSize;
}

// Setup to complete the upload.
CompleteMultipartUploadRequest completeRequest = new
CompleteMultipartUploadRequest
{
    BucketName = bucketName,
    Key = keyName,
    UploadId = initResponse.UploadId
};
completeRequest.AddPartETags(uploadResponses);

// Complete the upload.
CompleteMultipartUploadResponse completeUploadResponse =
    await s3Client.CompleteMultipartUploadAsync(completeRequest);
}
catch (Exception exception)
{
    Console.WriteLine("An AmazonS3Exception was thrown: { 0 }",
exception.Message);

    // Abort the upload.
    AbortMultipartUploadRequest abortMPURequest = new
AbortMultipartUploadRequest
{
    BucketName = bucketName,
    Key = keyName,
    UploadId = initResponse.UploadId
};
await s3Client.AbortMultipartUploadAsync(abortMPURequest);
}
}
public static void UploadPartProgressEventCallback(object sender,
StreamTransferProgressArgs e)
{
    // Process event.
    Console.WriteLine("{0}/{1}", e.TransferredBytes, e.TotalBytes);
}
}
```

PHP

This topic shows how to use the low-level `uploadPart` method from version 3 of the AWS SDK for PHP to upload a file in multiple parts. It assumes that you are already following the instructions for [Using the AWS SDK for PHP and Running PHP Examples \(p. 1193\)](#) and have the AWS SDK for PHP properly installed.

The following PHP example uploads a file to an Amazon S3 bucket using the low-level PHP API multipart upload. For information about running the PHP examples in this guide, see [Running PHP Examples \(p. 1194\)](#).

```
require 'vendor/autoload.php';

use Aws\S3\S3Client;

$bucket = '*** Your Bucket Name ***';
$keyname = '*** Your Object Key ***';
$filename = '*** Path to and Name of the File to Upload ***';

$s3 = new S3Client([
    'version' => 'latest',
    'region'  => 'us-east-1'
]);

$result = $s3->createMultipartUpload([
    'Bucket'      => $bucket,
    'Key'         => $keyname,
    'StorageClass' => 'REDUCED_REDUNDANCY',
    'Metadata'   => [
        'param1' => 'value 1',
        'param2' => 'value 2',
        'param3' => 'value 3'
    ]
]);
$uploadId = $result['UploadId'];

// Upload the file in parts.
try {
    $file = fopen($filename, 'r');
    $partNumber = 1;
    while (!feof($file)) {
        $result = $s3->uploadPart([
            'Bucket'      => $bucket,
            'Key'         => $keyname,
            'UploadId'   => $uploadId,
            'PartNumber' => $partNumber,
            'Body'        => fread($file, 5 * 1024 * 1024),
        ]);
        $parts['Parts'][$partNumber] = [
            'PartNumber' => $partNumber,
            'ETag'        => $result['ETag'],
        ];
        $partNumber++;
        echo "Uploading part {$partNumber} of {$filename}." . PHP_EOL;
    }
    fclose($file);
} catch (S3Exception $e) {
    $result = $s3->abortMultipartUpload([
        'Bucket'      => $bucket,
        'Key'         => $keyname,
        'UploadId'   => $uploadId
    ]);

    echo "Upload of {$filename} failed." . PHP_EOL;
}

// Complete the multipart upload.
$result = $s3->completeMultipartUpload([
    'Bucket'      => $bucket,
    'Key'         => $keyname,
    'UploadId'   => $uploadId,
```

```
        'MultipartUpload'    => $parts,
]);
$url = $result['Location'];

echo "Uploaded {$filename} to {$url}." . PHP_EOL;
```

Using the AWS SDK for Ruby

The AWS SDK for Ruby version 3 supports Amazon S3 multipart uploads in two ways. For the first option, you can use managed file uploads. For more information, see [Uploading Files to Amazon S3](#) in the *AWS Developer Blog*. Managed file uploads are the recommended method for uploading files to a bucket. They provide the following benefits:

- Manage multipart uploads for objects larger than 15MB.
- Correctly open files in binary mode to avoid encoding issues.
- Use multiple threads for uploading parts of large objects in parallel.

Alternatively, you can use the following multipart upload client operations directly:

- [create_multipart_upload](#) – Initiates a multipart upload and returns an upload ID.
- [upload_part](#) – Uploads a part in a multipart upload.
- [upload_part_copy](#) – Uploads a part by copying data from an existing object as data source.
- [complete_multipart_upload](#) – Completes a multipart upload by assembling previously uploaded parts.
- [abort_multipart_upload](#) – Stops a multipart upload.

For more information, see [Using the AWS SDK for Ruby - Version 3 \(p. 1194\)](#).

Using the REST API

The following sections in the *Amazon Simple Storage Service API Reference* describe the REST API for multipart upload.

- [Initiate Multipart Upload](#)
- [Upload Part](#)
- [Complete Multipart Upload](#)
- [Stop Multipart Upload](#)
- [List Parts](#)
- [List Multipart Uploads](#)

Using the AWS CLI

The following sections in the AWS Command Line Interface (AWS CLI) describe the operations for multipart upload.

- [Initiate Multipart Upload](#)
- [Upload Part](#)
- [Upload Part \(Copy\)](#)
- [Complete Multipart Upload](#)
- [Abort Multipart Upload](#)
- [List Parts](#)

- [List Multipart Uploads](#)

You can also use the REST API to make your own REST requests, or you can use one of the AWS SDKs. For more information about the REST API, see [Using the REST API \(p. 186\)](#). For more information about the SDKs, see [Uploading an object using multipart upload \(p. 174\)](#).

Uploading a directory using the high-level .NET TransferUtility class

You can use the `TransferUtility` class to upload an entire directory. By default, the API uploads only the files at the root of the specified directory. You can, however, specify recursively uploading files in all of the subdirectories.

To select files in the specified directory based on filtering criteria, specify filtering expressions. For example, to upload only the `.pdf` files from a directory, specify the `"*.pdf"` filter expression.

When uploading files from a directory, you don't specify the key names for the resulting objects. Amazon S3 constructs the key names using the original file path. For example, assume that you have a directory called `c:\myfolder` with the following structure:

Example

```
c:\myfolder
  \a.txt
  \b.pdf
  \media\
    An.mp3
```

When you upload this directory, Amazon S3 uses the following key names:

Example

```
a.txt
b.pdf
media/An.mp3
```

Example

The following C# example uploads a directory to an Amazon S3 bucket. It shows how to use various `TransferUtility.UploadDirectory` overloads to upload the directory. Each successive call to `upload` replaces the previous upload. For instructions on how to create and test a working sample, see [Running the Amazon S3 .NET Code Examples \(p. 1193\)](#).

```
using Amazon;
using Amazon.S3;
using Amazon.S3.Transfer;
using System;
using System.IO;
using System.Threading.Tasks;

namespace Amazon.DocSamples.S3
{
    class UploadDirMPUHighLevelAPITest
    {
        private const string existingBucketName = "*** bucket name ***";
        private const string directoryPath = @ "*** directory path ***";
        // The example uploads only .txt files.
```

```
private const string wildCard = "*.txt";
// Specify your bucket region (an example region is shown).
private static readonly RegionEndpoint bucketRegion = RegionEndpoint.USWest2;
private static IAmazonS3 s3Client;
static void Main()
{
    s3Client = new AmazonS3Client(bucketRegion);
    UploadDirAsync().Wait();
}

private static async Task UploadDirAsync()
{
    try
    {
        var directoryTransferUtility =
            new TransferUtility(s3Client);

        // 1. Upload a directory.
        await directoryTransferUtility.UploadDirectoryAsync(directoryPath,
            existingBucketName);
        Console.WriteLine("Upload statement 1 completed");

        // 2. Upload only the .txt files from a directory
        //     and search recursively.
        await directoryTransferUtility.UploadDirectoryAsync(
            directoryPath,
            existingBucketName,
            wildCard,
            SearchOption.AllDirectories);
        Console.WriteLine("Upload statement 2 completed");

        // 3. The same as Step 2 and some optional configuration.
        //     Search recursively for .txt files to upload.
        var request = new TransferUtilityUploadDirectoryRequest
        {
            BucketName = existingBucketName,
            Directory = directoryPath,
            SearchOption = SearchOption.AllDirectories,
            SearchPattern = wildCard
        };

        await directoryTransferUtility.UploadDirectoryAsync(request);
        Console.WriteLine("Upload statement 3 completed");
    }
    catch (AmazonS3Exception e)
    {
        Console.WriteLine(
            "Error encountered ***. Message:'{0}' when writing an object",
            e.Message);
    }
    catch (Exception e)
    {
        Console.WriteLine(
            "Unknown encountered on server. Message:'{0}' when writing an object",
            e.Message);
    }
}
}
```

Listing multipart uploads

You can use the AWS SDKs (low-level API) to retrieve a list of in-progress multipart uploads in Amazon S3.

Listing multipart uploads using the AWS SDK (low-level API)

Java

The following tasks guide you through using the low-level Java classes to list all in-progress multipart uploads on a bucket.

Low-level API multipart uploads listing process

1	Create an instance of the <code>ListMultipartUploadsRequest</code> class and provide the bucket name.
2	Run the <code>AmazonS3Client.listMultipartUploads</code> method. The method returns an instance of the <code>MultipartUploadListing</code> class that gives you information about the multipart uploads in progress.

The following Java code example demonstrates the preceding tasks.

Example

```
ListMultipartUploadsRequest allMultipartUploadsRequest =
    new ListMultipartUploadsRequest(existingBucketName);
MultipartUploadListing multipartUploadListing =
    s3Client.listMultipartUploads(allMultipartUploadsRequest);
```

.NET

To list all of the in-progress multipart uploads on a specific bucket, use the AWS SDK for .NET low-level multipart upload API's `ListMultipartUploadsRequest` class. The `AmazonS3Client.ListMultipartUploads` method returns an instance of the `ListMultipartUploadsResponse` class that provides information about the in-progress multipart uploads.

An in-progress multipart upload is a multipart upload that has been initiated using the initiate multipart upload request, but has not yet been completed or stopped. For more information about Amazon S3 multipart uploads, see [Uploading and copying objects using multipart upload \(p. 167\)](#).

The following C# example shows how to use the AWS SDK for .NET to list all in-progress multipart uploads on a bucket. For information about the example's compatibility with a specific version of the AWS SDK for .NET and instructions on how to create and test a working sample, see [Running the Amazon S3 .NET Code Examples \(p. 1193\)](#).

```
ListMultipartUploadsRequest request = new ListMultipartUploadsRequest
{
    BucketName = bucketName // Bucket receiving the uploads.
};

ListMultipartUploadsResponse response = await
    AmazonS3Client.ListMultipartUploadsAsync(request);
```

PHP

This topic shows how to use the low-level API classes from version 3 of the AWS SDK for PHP to list all in-progress multipart uploads on a bucket. It assumes that you are already following the instructions for [Using the AWS SDK for PHP and Running PHP Examples \(p. 1193\)](#) and have the AWS SDK for PHP properly installed.

The following PHP example demonstrates listing all in-progress multipart uploads on a bucket.

```
require 'vendor/autoload.php';

use Aws\S3\S3Client;

$bucket = '*** Your Bucket Name ***';

$s3 = new S3Client([
    'version' => 'latest',
    'region'  => 'us-east-1'
]);

// Retrieve a list of the current multipart uploads.
$result = $s3->listMultipartUploads([
    'Bucket' => $bucket
]);

// Write the list of uploads to the page.
print_r($result->toArray());
```

Listing multipart uploads using the REST API

The following sections in the *Amazon Simple Storage Service API Reference* describe the REST API for listing multipart uploads:

- [ListParts](#)-list the uploaded parts for a specific multipart upload.
- [ListMultipartUploads](#)-list in-progress multipart uploads.

Listing multipart uploads using the AWS CLI

The following sections in the AWS Command Line Interface describe the operations for listing multipart uploads.

- [list-parts](#)-list the uploaded parts for a specific multipart upload.
- [list-multipart-uploads](#)-list in-progress multipart uploads.

Tracking a multipart upload

The high-level multipart upload API provides a listen interface, `ProgressListener`, to track the upload progress when uploading an object to Amazon S3. Progress events occur periodically and notify the listener that bytes have been transferred.

Java

Example

```
TransferManager tm = new TransferManager(new ProfileCredentialsProvider());

PutObjectRequest request = new PutObjectRequest(
    existingBucketName, keyName, new File(filePath));

// Subscribe to the event and provide event handler.
request.setProgressListener(new ProgressListener() {
    public void progressChanged(ProgressEvent event) {
        System.out.println("Transferred bytes: " +
            event.getBytesTransferred());
    }
});
```

```
});
```

Example

The following Java code uploads a file and uses the `ProgressListener` to track the upload progress. For instructions on how to create and test a working sample, see [Testing the Amazon S3 Java Code Examples \(p. 1192\)](#).

```
import java.io.File;

import com.amazonaws.AmazonClientException;
import com.amazonaws.auth.profile.ProfileCredentialsProvider;
import com.amazonaws.event.ProgressEvent;
import com.amazonaws.event.ProgressListener;
import com.amazonaws.services.s3.model.PutObjectRequest;
import com.amazonaws.services.s3.transfer.TransferManager;
import com.amazonaws.services.s3.transfer.Upload;

public class TrackMPUProgressUsingHighLevelAPI {

    public static void main(String[] args) throws Exception {
        String existingBucketName = "*** Provide bucket name ***";
        String keyName           = "*** Provide object key ***";
        String filePath          = "*** file to upload ***";

        TransferManager tm = new TransferManager(new ProfileCredentialsProvider());

        // For more advanced uploads, you can create a request object
        // and supply additional request parameters (ex: progress listeners,
        // canned ACLs, etc.)
        PutObjectRequest request = new PutObjectRequest(
            existingBucketName, keyName, new File(filePath));

        // You can ask the upload for its progress, or you can
        // add a ProgressListener to your request to receive notifications
        // when bytes are transferred.
        request.setGeneralProgressListener(new ProgressListener() {
            @Override
            public void progressChanged(ProgressEvent progressEvent) {
                System.out.println("Transferred bytes: " +
                    progressEvent.getBytesTransferred());
            }
        });

        // TransferManager processes all transfers asynchronously,
        // so this call will return immediately.
        Upload upload = tm.upload(request);

        try {
            // You can block and wait for the upload to finish
            upload.waitForCompletion();
        } catch (AmazonClientException amazonClientException) {
            System.out.println("Unable to upload file, upload aborted.");
            amazonClientException.printStackTrace();
        }
    }
}
```

.NET

The following C# example uploads a file to an S3 bucket using the `TransferUtility` class, and tracks the progress of the upload. For information about the example's compatibility with a specific

version of the AWS SDK for .NET and instructions for creating and testing a working sample, see [Running the Amazon S3 .NET Code Examples \(p. 1193\)](#).

```
using Amazon;
using Amazon.S3;
using Amazon.S3.Transfer;
using System;
using System.Threading.Tasks;

namespace Amazon.DocSamples.S3
{
    class TrackMPUUsingHighLevelAPITest
    {
        private const string bucketName = "*** provide the bucket name ***";
        private const string keyName = "*** provide the name for the uploaded object
***";
        private const string filePath = " *** provide the full path name of the file to
upload ***";
        // Specify your bucket region (an example region is shown).
        private static readonly RegionEndpoint bucketRegion = RegionEndpoint.USWest2;
        private static IAmazonS3 s3Client;

        public static void Main()
        {
            s3Client = new AmazonS3Client(bucketRegion);
            TrackMPUAsync().Wait();
        }

        private static async Task TrackMPUAsync()
        {
            try
            {
                var fileTransferUtility = new TransferUtility(s3Client);

                // Use TransferUtilityUploadRequest to configure options.
                // In this example we subscribe to an event.
                var uploadRequest =
                    new TransferUtilityUploadRequest
                {
                    BucketName = bucketName,
                    FilePath = filePath,
                    Key = keyName
                };

                uploadRequest.UploadProgressEvent +=
                    new EventHandler<UploadProgressArgs>
                    (uploadRequest_UploadPartProgressEvent);

                await fileTransferUtility.UploadAsync(uploadRequest);
                Console.WriteLine("Upload completed");
            }
            catch (AmazonS3Exception e)
            {
                Console.WriteLine("Error encountered on server. Message:'{0}' when
writing an object", e.Message);
            }
            catch (Exception e)
            {
                Console.WriteLine("Unknown encountered on server. Message:'{0}' when
writing an object", e.Message);
            }
        }
    }
}
```

```
    static void uploadRequest_UploadPartProgressEvent(object sender,
UploadProgressArgs e)
{
    // Process event.
    Console.WriteLine("{0}/{1}", e.TransferredBytes, e.TotalBytes);
}
}
```

Aborting a multipart upload

After you initiate a multipart upload, you begin uploading parts. Amazon S3 stores these parts, but it creates the object from the parts only after you upload all of them and send a successful request to complete the multipart upload (you should verify that your request to complete multipart upload is successful). Upon receiving the complete multipart upload request, Amazon S3 assembles the parts and creates an object. If you don't send the complete multipart upload request successfully, Amazon S3 does not assemble the parts and does not create any object.

You are billed for all storage associated with uploaded parts. For more information, see [Multipart upload and pricing \(p. 169\)](#). So it's important that you either complete the multipart upload to have the object created or stop the multipart upload to remove any uploaded parts.

You can stop an in-progress multipart upload in Amazon S3 using the AWS Command Line Interface (AWS CLI), REST API, or AWS SDKs. You can also stop an incomplete multipart upload using a bucket lifecycle policy.

Using the AWS SDKs (high-level API)

Java

The `TransferManager` class provides the `abortMultipartUploads` method to stop multipart uploads in progress. An upload is considered to be in progress after you initiate it and until you complete it or stop it. You provide a `Date` value, and this API stops all the multipart uploads on that bucket that were initiated before the specified `Date` and are still in progress.

The following tasks guide you through using the high-level Java classes to stop multipart uploads.

High-level API multipart uploads stopping process

1	Create an instance of the <code>TransferManager</code> class.
2	Run the <code>TransferManager.abortMultipartUploads</code> method by passing the bucket name and a <code>Date</code> value.

The following Java code stops all multipart uploads in progress that were initiated on a specific bucket over a week ago. For instructions on how to create and test a working sample, see [Testing the Amazon S3 Java Code Examples \(p. 1192\)](#).

```
import java.util.Date;

import com.amazonaws.AmazonClientException;
import com.amazonaws.auth.profile.ProfileCredentialsProvider;
import com.amazonaws.services.s3.transfer.TransferManager;

public class AbortMPUUsingHighLevelAPI {
```

```
public static void main(String[] args) throws Exception {
    String existingBucketName = "**** Provide existing bucket name ****";

    TransferManager tm = new TransferManager(new ProfileCredentialsProvider());

    int sevenDays = 1000 * 60 * 60 * 24 * 7;
    Date oneWeekAgo = new Date(System.currentTimeMillis() - sevenDays);

    try {
        tm.abortMultipartUploads(existingBucketName, oneWeekAgo);
    } catch (AmazonClientException amazonClientException) {
        System.out.println("Unable to upload file, upload was aborted.");
        amazonClientException.printStackTrace();
    }
}
```

Note

You can also stop a specific multipart upload. For more information, see [Using the AWS SDKs \(low-level API\) \(p. 195\)](#).

.NET

The following C# example stops all in-progress multipart uploads that were initiated on a specific bucket over a week ago. For information about the example's compatibility with a specific version of the AWS SDK for .NET and instructions on creating and testing a working sample, see [Running the Amazon S3 .NET Code Examples \(p. 1193\)](#).

```
using Amazon;
using Amazon.S3;
using Amazon.S3.Transfer;
using System;
using System.Threading.Tasks;

namespace Amazon.DocSamples.S3
{
    class AbortMPUUsingHighLevelAPITest
    {
        private const string bucketName = "**** provide bucket name ****";
        // Specify your bucket region (an example region is shown).
        private static readonly RegionEndpoint bucketRegion = RegionEndpoint.USWest2;
        private static IAmazonS3 s3Client;

        public static void Main()
        {
            s3Client = new AmazonS3Client(bucketRegion);
            AbortMPUAsync().Wait();
        }

        private static async Task AbortMPUAsync()
        {
            try
            {
                var transferUtility = new TransferUtility(s3Client);

                // Abort all in-progress uploads initiated before the specified date.
                await transferUtility.AbortMultipartUploadsAsync(
                    bucketName, DateTime.Now.AddDays(-7));
            }
            catch (AmazonS3Exception e)
            {
                Console.WriteLine("Error encountered on server. Message:'{0}' when
writing an object", e.Message);
            }
        }
    }
}
```

```
        }
        catch (Exception e)
        {
            Console.WriteLine("Unknown encountered on server. Message:{0}' when
writing an object", e.Message);
        }
    }
}
```

Note

You can also stop a specific multipart upload. For more information, see [Using the AWS SDKs \(low-level API\) \(p. 195\)](#).

Using the AWS SDKs (low-level API)

You can stop an in-progress multipart upload by calling the `AmazonS3.abortMultipartUpload` method. This method deletes any parts that were uploaded to Amazon S3 and frees up the resources. You must provide the upload ID, bucket name, and key name. The following Java code example demonstrates how to stop an in-progress multipart upload.

To stop a multipart upload, you provide the upload ID, and the bucket and key names that are used in the upload. After you have stopped a multipart upload, you can't use the upload ID to upload additional parts. For more information about Amazon S3 multipart uploads, see [Uploading and copying objects using multipart upload \(p. 167\)](#).

Java

The following Java code example stops an in-progress multipart upload.

Example

```
InitiateMultipartUploadRequest initRequest =
    new InitiateMultipartUploadRequest(existingBucketName, keyName);
InitiateMultipartUploadResult initResponse =
    s3Client.initiateMultipartUpload(initRequest);

AmazonS3 s3Client = new AmazonS3Client(new ProfileCredentialsProvider());
s3Client.abortMultipartUpload(new AbortMultipartUploadRequest(
    existingBucketName, keyName, initResponse.getUploadId()));
```

Note

Instead of a specific multipart upload, you can stop all your multipart uploads initiated before a specific time that are still in progress. This clean-up operation is useful to stop old multipart uploads that you initiated but did not complete or stop. For more information, see [Using the AWS SDKs \(high-level API\) \(p. 193\)](#).

.NET

The following C# example shows how to stop a multipart upload. For a complete C# sample that includes the following code, see [Using the AWS SDKs \(low-level API\) \(p. 180\)](#).

```
AbortMultipartUploadRequest abortMPURequest = new AbortMultipartUploadRequest
{
    BucketName = existingBucketName,
    Key = keyName,
    UploadId = initResponse.UploadId
};
await AmazonS3Client.AbortMultipartUploadAsync(abortMPURequest);
```

You can also abort all in-progress multipart uploads that were initiated prior to a specific time. This clean-up operation is useful for aborting multipart uploads that didn't complete or were aborted. For more information, see [Using the AWS SDKs \(high-level API\) \(p. 193\)](#).

PHP

This example shows how to use a class from version 3 of the AWS SDK for PHP to abort a multipart upload that is in progress. It assumes that you are already following the instructions for [Using the AWS SDK for PHP and Running PHP Examples \(p. 1193\)](#) and have the AWS SDK for PHP properly installed. The example the `abortMultipartUpload()` method.

For information about running the PHP examples in this guide, see [Running PHP Examples \(p. 1194\)](#).

```
require 'vendor/autoload.php';

use Aws\S3\S3Client;

$bucket = '*** Your Bucket Name ***';
$keyname = '*** Your Object Key ***';
$uploadId = '*** Upload ID of upload to Abort ***';

$s3 = new S3Client([
    'version' => 'latest',
    'region'  => 'us-east-1'
]);

// Abort the multipart upload.
$s3->abortMultipartUpload([
    'Bucket'    => $bucket,
    'Key'        => $keyname,
    'UploadId'  => $uploadId,
]);
```

Using the REST API

For more information about using the REST API to stop a multipart upload, see [AbortMultipartUpload](#) in the [Amazon Simple Storage Service API Reference](#).

Using the AWS CLI

For more information about using the AWS CLI to stop a multipart upload, see [abort-multipart-upload](#) in the [AWS CLI Command Reference](#).

Copying an object using multipart upload

The examples in this section show you how to copy objects greater than 5 GB using the multipart upload API. You can copy objects less than 5 GB in a single operation. For more information, see [Copying objects \(p. 201\)](#).

Using the AWS SDKs

To copy an object using the low-level API, do the following:

- Initiate a multipart upload by calling the `AmazonS3Client.initiateMultipartUpload()` method.
- Save the upload ID from the response object that the `AmazonS3Client.initiateMultipartUpload()` method returns. You provide this upload ID for each part-upload operation.

- Copy all of the parts. For each part that you need to copy, create a new instance of the `CopyPartRequest` class. Provide the part information, including the source and destination bucket names, source and destination object keys, upload ID, locations of the first and last bytes of the part, and part number.
- Save the responses of the `AmazonS3Client.copyPart()` method calls. Each response includes the ETag value and part number for the uploaded part. You need this information to complete the multipart upload.
- Call the `AmazonS3Client.completeMultipartUpload()` method to complete the copy operation.

Java

Example

The following example shows how to use the Amazon S3 low-level Java API to perform a multipart copy. For instructions on creating and testing a working sample, see [Testing the Amazon S3 Java Code Examples \(p. 1192\)](#).

```
import com.amazonaws.AmazonServiceException;
import com.amazonaws.SdkClientException;
import com.amazonaws.auth.profile.ProfileCredentialsProvider;
import com.amazonaws.regions.Regions;
import com.amazonaws.services.s3.AmazonS3;
import com.amazonaws.services.s3.AmazonS3ClientBuilder;
import com.amazonaws.services.s3.model.*;

import java.io.IOException;
import java.util.ArrayList;
import java.util.List;

public class LowLevelMultipartCopy {

    public static void main(String[] args) throws IOException {
        Regions clientRegion = Regions.DEFAULT_REGION;
        String sourceBucketName = "*** Source bucket name ***";
        String sourceObjectKey = "*** Source object key ***";
        String destBucketName = "*** Target bucket name ***";
        String destObjectKey = "*** Target object key ***";

        try {
            AmazonS3 s3Client = AmazonS3ClientBuilder.standard()
                .withCredentials(new ProfileCredentialsProvider())
                .withRegion(clientRegion)
                .build();

            // Initiate the multipart upload.
            InitiateMultipartUploadRequest initRequest = new
InitiateMultipartUploadRequest(destBucketName, destObjectKey);
            InitiateMultipartUploadResult initResult =
s3Client.initiateMultipartUpload(initRequest);

            // Get the object size to track the end of the copy operation.
            GetObjectMetadataRequest metadataRequest = new
GetObjectMetadataRequest(sourceBucketName, sourceObjectKey);
            ObjectMetadata metadataResult =
s3Client.getObjectMetadata(metadataRequest);
            long objectSize = metadataResult.getContentLength();

            // Copy the object using 5 MB parts.
            long partSize = 5 * 1024 * 1024;
            long bytePosition = 0;
```

```

int partNum = 1;
List<CopyPartResult> copyResponses = new ArrayList<CopyPartResult>();
while (bytePosition < objectSize) {
    // The last part might be smaller than partSize, so check to make sure
    // that lastByte isn't beyond the end of the object.
    long lastByte = Math.min(bytePosition + partSize - 1, objectSize - 1);

    // Copy this part.
    CopyPartRequest copyRequest = new CopyPartRequest()
        .withSourceBucketName(sourceBucketName)
        .withSourceKey(sourceObjectKey)
        .withDestinationBucketName(destBucketName)
        .withDestinationKey(destObjectKey)
        .withUploadId(initResult.getUploadId())
        .withFirstByte(bytePosition)
        .withLastByte(lastByte)
        .withPartNumber(partNum++);
    copyResponses.add(s3Client.copyPart(copyRequest));
    bytePosition += partSize;
}

// Complete the upload request to concatenate all uploaded parts and make
// the copied object available.
CompleteMultipartUploadRequest completeRequest = new
CompleteMultipartUploadRequest(
    destBucketName,
    destObjectKey,
    initResult.getUploadId(),
    getETags(copyResponses));
s3Client.completeMultipartUpload(completeRequest);
System.out.println("Multipart copy complete.");
} catch (AmazonServiceException e) {
    // The call was transmitted successfully, but Amazon S3 couldn't process
    // it, so it returned an error response.
    e.printStackTrace();
} catch (SdkClientException e) {
    // Amazon S3 couldn't be contacted for a response, or the client
    // couldn't parse the response from Amazon S3.
    e.printStackTrace();
}
}

// This is a helper function to construct a list of ETags.
private static List<PartETag> getETags(List<CopyPartResult> responses) {
    List<PartETag> etags = new ArrayList<PartETag>();
    for (CopyPartResult response : responses) {
        etags.add(new PartETag(response.getPartNumber(), response.getETag()));
    }
    return etags;
}
}

```

.NET

The following C# example shows how to use the AWS SDK for .NET to copy an Amazon S3 object that is larger than 5 GB from one source location to another, such as from one bucket to another. To copy objects that are smaller than 5 GB, use the single-operation copy procedure described in [Using the AWS SDKs \(p. 204\)](#). For more information about Amazon S3 multipart uploads, see [Uploading and copying objects using multipart upload \(p. 167\)](#).

This example shows how to copy an Amazon S3 object that is larger than 5 GB from one S3 bucket to another using the AWS SDK for .NET multipart upload API. For information about SDK

compatibility and instructions for creating and testing a working sample, see [Running the Amazon S3 .NET Code Examples \(p. 1193\)](#).

```
using Amazon;
using Amazon.S3;
using Amazon.S3.Model;
using System;
using System.Collections.Generic;
using System.Threading.Tasks;

namespace Amazon.DocSamples.S3
{
    class CopyObjectUsingMPUapiTest
    {
        private const string sourceBucket = "**** provide the name of the bucket with
source object ****";
        private const string targetBucket = "**** provide the name of the bucket to copy
the object to ****";
        private const string sourceObjectKey = "**** provide the name of object to copy
****";
        private const string targetObjectKey = "**** provide the name of the object copy
****";
        // Specify your bucket region (an example region is shown).
        private static readonly RegionEndpoint bucketRegion = RegionEndpoint.USWest2;
        private static IAmazonS3 s3Client;

        public static void Main()
        {
            s3Client = new AmazonS3Client(bucketRegion);
            Console.WriteLine("Copying an object");
            MPUCopyObjectAsync().Wait();
        }
        private static async Task MPUCopyObjectAsync()
        {
            // Create a list to store the upload part responses.
            List<UploadPartResponse> uploadResponses = new List<UploadPartResponse>();
            List<CopyPartResponse> copyResponses = new List<CopyPartResponse>();

            // Setup information required to initiate the multipart upload.
            InitiateMultipartUploadRequest initiateRequest =
                new InitiateMultipartUploadRequest
                {
                    BucketName = targetBucket,
                    Key = targetObjectKey
                };

            // Initiate the upload.
            InitiateMultipartUploadResponse initResponse =
                await s3Client.InitiateMultipartUploadAsync(initiateRequest);

            // Save the upload ID.
            String uploadId = initResponse.UploadId;

            try
            {
                // Get the size of the object.
                GetObjectMetadataRequest metadataRequest = new GetObjectMetadataRequest
                {
                    BucketName = sourceBucket,
                    Key = sourceObjectKey
                };

                GetObjectMetadataResponse metadataResponse =
                    await s3Client.GetObjectMetadataAsync(metadataRequest);
                long objectSize = metadataResponse.ContentLength; // Length in bytes.
            }
        }
    }
}
```

```
// Copy the parts.
long partSize = 5 * (long)Math.Pow(2, 20); // Part size is 5 MB.

long bytePosition = 0;
for (int i = 1; bytePosition < objectSize; i++)
{
    CopyPartRequest copyRequest = new CopyPartRequest
    {
        DestinationBucket = targetBucket,
        DestinationKey = targetObjectKey,
        SourceBucket = sourceBucket,
        SourceKey = sourceObjectKey,
        UploadId = uploadId,
        FirstByte = bytePosition,
        LastByte = bytePosition + partSize - 1 >= objectSize ?
objectSize - 1 : bytePosition + partSize - 1,
        PartNumber = i
    };

    copyResponses.Add(await s3Client.CopyPartAsync(copyRequest));

    bytePosition += partSize;
}

// Set up to complete the copy.
CompleteMultipartUploadRequest completeRequest =
new CompleteMultipartUploadRequest
{
    BucketName = targetBucket,
    Key = targetObjectKey,
    UploadId = initResponse.UploadId
};
completeRequest.AddPartETags(copyResponses);

// Complete the copy.
CompleteMultipartUploadResponse completeUploadResponse =
    await s3Client.CompleteMultipartUploadAsync(completeRequest);
}
catch (AmazonS3Exception e)
{
    Console.WriteLine("Error encountered on server. Message:'{0}' when
writing an object", e.Message);
}
catch (Exception e)
{
    Console.WriteLine("Unknown encountered on server. Message:'{0}' when
writing an object", e.Message);
}
}
```

Using the REST API

The following sections in the *Amazon Simple Storage Service API Reference* describe the REST API for multipart upload. For copying an existing object, use the Upload Part (Copy) API and specify the source object by adding the `x-amz-copy-source` request header in your request.

- [Initiate Multipart Upload](#)
- [Upload Part](#)
- [Upload Part \(Copy\)](#)

- [Complete Multipart Upload](#)
- [Abort Multipart Upload](#)
- [List Parts](#)
- [List Multipart Uploads](#)

You can use these APIs to make your own REST requests, or you can use one of the SDKs we provide. For more information about using Multipart Upload with the AWS CLI, see [Using the AWS CLI \(p. 186\)](#). For more information about the SDKs, see [AWS SDK support for multipart upload \(p. 170\)](#).

Amazon S3 multipart upload limits

The following table provides multipart upload core specifications. For more information, see [Uploading and copying objects using multipart upload \(p. 167\)](#).

Item	Specification
Maximum object size	5 TiB
Maximum number of parts per upload	10,000
Part numbers	1 to 10,000 (inclusive)
Part size	5 MiB to 5 GiB. There is no minimum size limit on the last part of your multipart upload.
Maximum number of parts returned for a list parts request	1000
Maximum number of multipart uploads returned in a list multipart uploads request	1000

Copying objects

The copy operation creates a copy of an object that is already stored in Amazon S3.

You can create a copy of your object up to 5 GB in a single atomic operation. However, to copy an object that is greater than 5 GB, you must use the multipart upload API.

Using the copy operation, you can:

- Create additional copies of objects
- Rename objects by copying them and deleting the original ones
- Move objects across Amazon S3 locations (for example, us-west-1 and Europe)
- Change object metadata

Each Amazon S3 object has metadata. It is a set of name-value pairs. You can set object metadata at the time you upload it. After you upload the object, you cannot modify object metadata. The only way to modify object metadata is to make a copy of the object and set the metadata. In the copy operation, set the same object as the source and target.

Each object has metadata. Some of it is system metadata and other is user-defined. You can control some of the system metadata, such as the storage class configuration to use for the object, and you can

configure server-side encryption. When you copy an object, user-controlled system metadata and user-defined metadata are also copied. Amazon S3 resets the system-controlled metadata. For example, when you copy an object, Amazon S3 resets the creation date of the copied object. You don't need to set any of these values in your copy request.

When copying an object, you might decide to update some of the metadata values. For example, if your source object is configured to use S3 Standard storage, you might choose to use S3 Intelligent-Tiering for the object copy. You might also decide to alter some of the user-defined metadata values present on the source object. If you choose to update any of the object's user-configurable metadata (system or user-defined) during the copy, then you must explicitly specify all of the user-configurable metadata present on the source object in your request, even if you are changing only one of the metadata values.

For more information about the object metadata, see [Working with object metadata \(p. 153\)](#).

Note

- Copying objects across locations incurs bandwidth charges.
- If the source object is archived in [S3 Glacier Flexible Retrieval](#) or [S3 Glacier Deep Archive](#), you must first restore a temporary copy before you can copy the object to another bucket. For information about archiving objects, see [Transitioning to the S3 Glacier Flexible Retrieval and S3 Glacier Deep Archive storage classes \(object archival\) \(p. 705\)](#).

When copying objects, you can request that Amazon S3 save the target object encrypted with an AWS KMS key, an Amazon S3 managed encryption key, or a customer-provided encryption key. Accordingly, you must specify encryption information in your request. If the copy source is an object that is stored in Amazon S3 using server-side encryption with a customer-provided key, you must provide encryption information in your request so that Amazon S3 can decrypt the object for copying. For more information, see [Protecting data using encryption \(p. 337\)](#).

When copying objects, you can choose to use a different checksum algorithm for the object. Whether you choose to use the same algorithm or a new one, Amazon S3 calculates a new checksum value after the object is copied. Amazon S3 does not directly copy the value of the checksum. The checksum value of objects that were loaded using multipart uploads might change. For more information about how the checksum is calculated, see [Using part-level checksums for multipart uploads \(p. 222\)](#).

To copy more than one Amazon S3 object with a single request, you can use Amazon S3 batch operations. You provide S3 Batch Operations with a list of objects to operate on. S3 Batch Operations calls the respective API to perform the specified operation. A single Batch Operations job can perform the specified operation on billions of objects containing exabytes of data.

The S3 Batch Operations feature tracks progress, sends notifications, and stores a detailed completion report of all actions, providing a fully managed, auditable, serverless experience. You can use S3 Batch Operations through the AWS Management Console, AWS CLI, AWS SDKs, or REST API. For more information, see [the section called "Batch Operations basics" \(p. 881\)](#).

To copy an object

To copy an object, use the following methods.

Using the S3 console

In the Amazon S3 console, you can copy or move an object. For more information, see the following procedures.

Note

Objects encrypted with customer-provided encryption keys (SSE-C) cannot be copied or moved using the S3 console. To copy or move objects encrypted with SSE-C, use the AWS CLI, AWS SDK, or the Amazon S3 REST API.

To copy an object

1. Sign in to the AWS Management Console and open the Amazon S3 console at <https://console.aws.amazon.com/s3/>.
2. Navigate to the Amazon S3 bucket or folder that contains the objects that you want to copy.
3. Select the check box to the left of the names of the objects that you want to copy.
4. Choose **Actions** and choose **Copy** from the list of options that appears.

Alternatively, choose **Copy** from the options in the upper-right corner.
5. Select the destination type and destination account. To specify the destination path, choose **Browse S3**, navigate to the destination, and select the check box to the left of the destination. Choose **Choose destination** in the lower-right corner.

Alternatively, enter the destination path.
6. If you do *not* have bucket versioning enabled, you might be asked to acknowledge that existing objects with the same name are overwritten. If this is OK, select the check box and proceed. If you want to keep all versions of objects in this bucket, select **Enable Bucket Versioning**. You can also update the default encryption and S3 Object Lock properties.
7. Under **Additional checksums**, choose whether you want to copy the objects using the existing checksum function or replace the existing checksum function with a new one. When you uploaded the objects, you had the option to specify the checksum algorithm that was used to verify data integrity. When copying the object, you have the option to choose a new function. If you did not originally specify an additional checksum, you can use this section of the copy options to add one.

Note

Even if you opt to use the same checksum function, your checksum value might change if you copy the object and it is over 16 MB in size. The checksum value might change because of how checksums are calculated for multipart uploads. For more information about how the checksum might change when copying the object, see [Using part-level checksums for multipart uploads \(p. 222\)](#).

To change the checksum function, choose **Replace with a new checksum function**. Choose the new checksum function from the box. When the object is copied over, the new checksum is calculated and stored using the specified algorithm.

8. Choose **Copy** in the bottom-right corner. Amazon S3 copies your objects to the destination.

To move objects

1. Sign in to the AWS Management Console and open the Amazon S3 console at <https://console.aws.amazon.com/s3/>.
2. Navigate to the Amazon S3 bucket or folder that contains the objects that you want to move.
3. Select the check box to the left of the names of the objects that you want to move.
4. Choose **Actions** and choose **Move** from the list of options that appears.

Alternatively, choose **Move** from the options in the upper-right corner.
5. To specify the destination path, choose **Browse S3**, navigate to the destination, and select the check box to the left of the destination. Choose **Choose destination** in the lower-right corner.

Alternatively, enter the destination path.
6. If you do *not* have bucket versioning enabled, you might be asked to acknowledge that existing objects with the same name are overwritten. If this is OK, select the check box and proceed. If you want to keep all versions of objects in this bucket, select **Enable Bucket Versioning**. You can also update the default encryption and Object Lock properties.
7. Choose **Move** in the bottom-right corner. Amazon S3 moves your objects to the destination.

Note

- This action creates a copy of all specified objects with updated settings, updates the last-modified date in the specified location, and adds a delete marker to the original object.
- When moving folders, wait for the move action to finish before making additional changes in the folders.
- This action updates metadata for bucket versioning, encryption, Object Lock features, and archived objects.

Using the AWS SDKs

The examples in this section show how to copy objects up to 5 GB in a single operation. For copying objects greater than 5 GB, you must use multipart upload API. For more information, see [Copying an object using multipart upload \(p. 196\)](#).

Java

Example

The following example copies an object in Amazon S3 using the AWS SDK for Java. For instructions on creating and testing a working sample, see [Testing the Amazon S3 Java Code Examples \(p. 1192\)](#).

```
import com.amazonaws.AmazonServiceException;
import com.amazonaws.SdkClientException;
import com.amazonaws.auth.profile.ProfileCredentialsProvider;
import com.amazonaws.regions.Regions;
import com.amazonaws.services.s3.AmazonS3;
import com.amazonaws.services.s3.AmazonS3ClientBuilder;
import com.amazonaws.services.s3.model.CopyObjectRequest;

import java.io.IOException;

public class CopyObjectSingleOperation {

    public static void main(String[] args) throws IOException {
        Regions clientRegion = Regions.DEFAULT_REGION;
        String bucketName = "*** Bucket name ***";
        String sourceKey = "*** Source object key *** ";
        String destinationKey = "*** Destination object key ***";

        try {
            AmazonS3 s3Client = AmazonS3ClientBuilder.standard()
                .withCredentials(new ProfileCredentialsProvider())
                .withRegion(clientRegion)
                .build();

            // Copy the object into a new object in the same bucket.
            CopyObjectRequest copyObjRequest = new CopyObjectRequest(bucketName,
                sourceKey, bucketName, destinationKey);
            s3Client.copyObject(copyObjRequest);
        } catch (AmazonServiceException e) {
            // The call was transmitted successfully, but Amazon S3 couldn't process
            // it, so it returned an error response.
            e.printStackTrace();
        } catch (SdkClientException e) {
            // Amazon S3 couldn't be contacted for a response, or the client
            // couldn't parse the response from Amazon S3.
            e.printStackTrace();
        }
    }
}
```

```
}
```

.NET

The following C# example uses the high-level AWS SDK for .NET to copy objects that are as large as 5 GB in a single operation. For objects that are larger than 5 GB, use the multipart upload copy example described in [Copying an object using multipart upload \(p. 196\)](#).

This example makes a copy of an object that is a maximum of 5 GB. For information about the example's compatibility with a specific version of the AWS SDK for .NET and instructions on how to create and test a working sample, see [Running the Amazon S3 .NET Code Examples \(p. 1193\)](#).

```
using Amazon;
using Amazon.S3;
using Amazon.S3.Model;
using System;
using System.Threading.Tasks;

namespace Amazon.DocSamples.S3
{
    class CopyObjectTest
    {
        private const string sourceBucket = "*** provide the name of the bucket with
source object ***";
        private const string destinationBucket = "*** provide the name of the bucket to
copy the object to ***";
        private const string objectKey = "*** provide the name of object to copy ***";
        private const string destObjectKey = "*** provide the destination object key
name ***";
        // Specify your bucket region (an example region is shown).
        private static readonly RegionEndpoint bucketRegion = RegionEndpoint.USWest2;
        private static IAmazonS3 s3Client;

        public static void Main()
        {
            s3Client = new AmazonS3Client(bucketRegion);
            Console.WriteLine("Copying an object");
            CopyingObjectAsync().Wait();
        }

        private static async Task CopyingObjectAsync()
        {
            try
            {
                CopyObjectRequest request = new CopyObjectRequest
                {
                    SourceBucket = sourceBucket,
                    SourceKey = objectKey,
                    DestinationBucket = destinationBucket,
                    DestinationKey = destObjectKey
                };
                CopyObjectResponse response = await s3Client.CopyObjectAsync(request);
            }
            catch (AmazonS3Exception e)
            {
                Console.WriteLine("Error encountered on server. Message:'{0}' when
writing an object", e.Message);
            }
            catch (Exception e)
            {
                Console.WriteLine("Unknown encountered on server. Message:'{0}' when
writing an object", e.Message);
            }
        }
    }
}
```

```
        }
    }
}
```

PHP

This topic guides you through using classes from version 3 of the AWS SDK for PHP to copy a single object and multiple objects within Amazon S3, from one bucket to another or within the same bucket.

This topic assumes that you are already following the instructions for [Using the AWS SDK for PHP and Running PHP Examples \(p. 1193\)](#) and have the AWS SDK for PHP properly installed.

The following PHP example illustrates the use of the `copyObject()` method to copy a single object within Amazon S3 and using a batch of calls to `CopyObject` using the `getCommand()` method to make multiple copies of an object.

Copying objects

1	Create an instance of an Amazon S3 client by using the <code>Aws\S3\S3Client</code> class constructor.
2	To make multiple copies of an object, you run a batch of calls to the Amazon S3 client <code>getCommand()</code> method, which is inherited from the <code>Aws\CommandInterface</code> class. You provide the <code>CopyObject</code> command as the first argument and an array containing the source bucket, source key name, target bucket, and target key name as the second argument.

```
require 'vendor/autoload.php';

use Aws\S3\S3Client;

$sourceBucket = '*** Your Source Bucket Name ***';
$sourceKeyname = '*** Your Source Object Key ***';
$targetBucket = '*** Your Target Bucket Name ***';

$s3 = new S3Client([
    'version' => 'latest',
    'region'  => 'us-east-1'
]);

// Copy an object.
$s3->copyObject([
    'Bucket'      => $targetBucket,
    'Key'         => "{$sourceKeyname}-copy",
    'CopySource'  => "{$sourceBucket}/{$sourceKeyname}",
]);

// Perform a batch of CopyObject operations.
$batch = array();
for ($i = 1; $i <= 3; $i++) {
    $batch[] = $s3->getCommand('CopyObject', [
        'Bucket'      => $targetBucket,
        'Key'         => "{$targetKeyname}-{$i}",
        'CopySource'  => "{$sourceBucket}/{$sourceKeyname}",
    ]);
}
try {
    $results = CommandPool::batch($s3, $batch);
    foreach($results as $result) {
```

```
        if ($result instanceof ResultInterface) {
            // Result handling here
        }
        if ($result instanceof AwsException) {
            // AwsException handling here
        }
    }
} catch (\Exception $e) {
    // General error handling here
}
```

Ruby

The following tasks guide you through using the Ruby classes to copy an object in Amazon S3 from one bucket to another or within the same bucket.

Copying objects

1	Use the Amazon S3 modularized gem for version 3 of the AWS SDK for Ruby, require 'aws-sdk-s3', and provide your AWS credentials. For more information about how to provide your credentials, see Making requests using AWS account or IAM user credentials (p. 1147) .
2	Provide the request information, such as source bucket name, source key name, destination bucket name, and destination key.

The following Ruby code example demonstrates the preceding tasks using the `#copy_object` method to copy an object from one bucket to another.

```
require "aws-sdk-s3"

# Wraps Amazon S3 object actions.
class ObjectCopyWrapper
    attr_reader :source_object

    # @param source_object [Aws::S3::Object] An existing Amazon S3 object. This is used
    # as the source object for
    #                                         copy actions.
    def initialize(source_object)
        @source_object = source_object
    end

    # Copy the source object to the specified target bucket and rename it with the target
    # key.
    #
    # @param target_bucket [Aws::S3::Bucket] An existing Amazon S3 bucket where the
    # object is copied.
    # @param target_object_key [String] The key to give the copy of the object.
    # @return [Aws::S3::Object, nil] The copied object when successful; otherwise, nil.
    def copy_object(target_bucket, target_object_key)
        @source_object.copy_to(bucket: target_bucket.name, key: target_object_key)
        target_bucket.object(target_object_key)
    rescue Aws::Errors::ServiceError => e
        puts "Couldn't copy #{@source_object.key} to #{target_object_key}. Here's why:
#{e.message}"
    end
end

# Replace the source and target bucket names with existing buckets you own and replace
# the source object key
# with an existing object in the source bucket.
def run_demo
```

```
source_bucket_name = "doc-example-bucket1"
source_key = "my-source-file.txt"
target_bucket_name = "doc-example-bucket2"
target_key = "my-target-file.txt"

source_bucket = Aws::S3::Bucket.new(source_bucket_name)
wrapper = ObjectCopyWrapper.new(source_bucket.object(source_key))
target_bucket = Aws::S3::Bucket.new(target_bucket_name)
target_object = wrapper.copy_object(target_bucket, target_key)
return unless target_object

puts "Copied #{source_key} from #{source_bucket_name} to
#{target_object.bucket_name}:#{target_object.key}."
end

run_demo if $PROGRAM_NAME == __FILE__
```

Copying an object using the REST API

This example describes how to copy an object using REST. For more information about the REST API, go to [PUT Object \(Copy\)](#).

This example copies the `flotsam` object from the `pacific` bucket to the `jetsam` object of the `atlantic` bucket, preserving its metadata.

```
PUT /jetsam HTTP/1.1
Host: atlantic.s3.amazonaws.com
x-amz-copy-source: /pacific/flotsam
Authorization: AWS AKIAIOSFODNN7EXAMPLE:ENoSbxYByFA0UGLZUqJN5EunLDg=
Date: Wed, 20 Feb 2008 22:12:21 +0000
```

The signature was generated from the following information.

```
PUT\r\n
\r\n
\r\n
Wed, 20 Feb 2008 22:12:21 +0000\r\n

x-amz-copy-source:/pacific/flotsam\r\n
/atlantic/jetsam
```

Amazon S3 returns the following response that specifies the ETag of the object and when it was last modified.

```
HTTP/1.1 200 OK
x-amz-id-2: Vyaxt7qEbzbv34BnSu5hctyyNSlHTYZFMWK4FtzO+ix8JQnyaLdTshL0KxatbaOzt
x-amz-request-id: 6B13C3C5B34AF333
Date: Wed, 20 Feb 2008 22:13:01 +0000

Content-Type: application/xml
Transfer-Encoding: chunked
Connection: close
Server: AmazonS3
<?xml version="1.0" encoding="UTF-8"?>

<CopyObjectResult>
  <LastModified>2008-02-20T22:13:01</LastModified>
  <ETag>"7e9c608af58950deeb370c98608ed097"</ETag>
</CopyObjectResult>
```

Downloading an object

This section explains how to download objects from an S3 bucket.

Data transfer fees apply when you download objects. For information about Amazon S3 features, and pricing, see [Amazon S3](#).

You can download a single object per request using the Amazon S3 console. To download multiple objects, use the AWS CLI, AWS SDKs, or REST API.

When you download an object programmatically, its metadata is returned in the response headers. There are times when you want to override certain response header values returned in a GET response. For example, you might override the Content-Disposition response header value in your GET request. The REST GET Object API (see [GET Object](#)) allows you to specify query string parameters in your GET request to override these values. The AWS SDKs for Java, .NET, and PHP also provide necessary objects you can use to specify values for these response headers in your GET request.

When retrieving objects that are stored encrypted using server-side encryption, you must provide appropriate request headers. For more information, see [Protecting data using encryption \(p. 337\)](#).

Using the S3 console

This section explains how to use the Amazon S3 console to download an object from an S3 bucket using a presigned URL.

Note

- You can only download one object at a time.
- Objects with key names ending with period(s) "." downloaded using the Amazon S3 console will have the period(s) "." removed from the key name of the downloaded object. To download an object with the key name ending in period(s) "." retained in the downloaded object, you must use the AWS Command Line Interface (AWS CLI), AWS SDKs, or REST API. For AWS CLI, REST API, and AWS SDK information and examples, see [Downloading an object](#).

To download an object from an S3 bucket

1. Sign in to the AWS Management Console and open the Amazon S3 console at <https://console.aws.amazon.com/s3/>.
2. In the **Buckets** list, choose the name of the bucket that you want to download an object from.
3. You can download an object from an S3 bucket in any of the following ways:
 - Select the object and choose **Download** or choose **Download as** from the **Actions** menu if you want to download the object to a specific folder.
 - If you want to download a specific version of the object, select the **Show versions** button. Select the version of the object that you want and choose **Download** or choose **Download as** from the **Actions** menu if you want to download the object to a specific folder.

Using the AWS SDKs

Java

When you download an object through the AWS SDK for Java, Amazon S3 returns all of the object's metadata and an input stream from which to read the object's contents.

To retrieve an object, you do the following:

- Execute the `AmazonS3Client.getObject()` method, providing the bucket name and object key in the request.
- Execute one of the `S3Object` instance methods to process the input stream.

Note

Your network connection remains open until you read all of the data or close the input stream. We recommend that you read the content of the stream as quickly as possible.

The following are some variations you might use:

- Instead of reading the entire object, you can read only a portion of the object data by specifying the byte range that you want in the request.
- You can optionally override the response header values by using a `ResponseHeaderOverrides` object and setting the corresponding request property. For example, you can use this feature to indicate that the object should be downloaded into a file with a different file name than the object key name.

The following example retrieves an object from an Amazon S3 bucket three ways: first, as a complete object, then as a range of bytes from the object, then as a complete object with overridden response header values. For more information about getting objects from Amazon S3, see [GET Object](#). For instructions on creating and testing a working sample, see [Testing the Amazon S3 Java Code Examples \(p. 1192\)](#).

```
import com.amazonaws.AmazonServiceException;
import com.amazonaws.SdkClientException;
import com.amazonaws.auth.profile.ProfileCredentialsProvider;
import com.amazonaws.regions.Regions;
import com.amazonaws.services.s3.AmazonS3;
import com.amazonaws.services.s3.AmazonS3ClientBuilder;
import com.amazonaws.services.s3.model.GetObjectRequest;
import com.amazonaws.services.s3.model.ResponseHeaderOverrides;
import com.amazonaws.services.s3.model.S3Object;

import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStream;
import java.io.InputStreamReader;

public class GetObject2 {

    public static void main(String[] args) throws IOException {
        Regions clientRegion = Regions.DEFAULT_REGION;
        String bucketName = "*** Bucket name ***";
        String key = "*** Object key ***";

        S3Object fullObject = null, objectPortion = null, headerOverrideObject = null;
        try {
            AmazonS3 s3Client = AmazonS3ClientBuilder.standard()
                .withRegion(clientRegion)
                .withCredentials(new ProfileCredentialsProvider())
                .build();

            // Get an object and print its contents.
            System.out.println("Downloading an object");
            fullObject = s3Client.getObject(new GetObjectRequest(bucketName, key));
            System.out.println("Content-Type: " +
                fullObject.getObjectMetadata().getContentType());
        }
    }
}
```

.NET

When you download an object, you get all of the object's metadata and a stream from which to read the contents. You should read the content of the stream as quickly as possible because the data is streamed directly from Amazon S3 and your network connection will remain open until you read all the data or close the input stream. You do the following to get an object:

- Execute the `getObject` method by providing bucket name and object key in the request.
 - Execute one of the `GetObjectResponse` methods to process the stream.

The following are some variations you might use:

- Instead of reading the entire object, you can read only the portion of the object data by specifying the byte range in the request, as shown in the following C# example:

Example

```
GetObjectRequest request = new GetObjectRequest
{
    BucketName = bucketName,
    Key = keyName,
    ByteRange = new ByteRange(0, 10)
};
```

- When retrieving an object, you can optionally override the response header values (see [Downloading an object \(p. 209\)](#)) by using the ResponseHeaderOverrides object and setting the corresponding request property. The following C# code example shows how to do this. For example, you can use this feature to indicate that the object should be downloaded into a file with a different file name than the object key name.

Example

```
GetObjectRequest request = new GetObjectRequest
{
    BucketName = bucketName,
    Key = keyName
};

ResponseHeaderOverrides responseHeaders = new ResponseHeaderOverrides();
responseHeaders.CacheControl = "No-cache";
responseHeaders.ContentDisposition = "attachment; filename=testing.txt";

request.ResponseHeaderOverrides = responseHeaders;
```

Example

The following C# code example retrieves an object from an Amazon S3 bucket. From the response, the example reads the object data using the `GetObjectResponse.ResponseStream` property. The example also shows how you can use the `GetObjectResponse.Metadata` collection to read object metadata. If the object you retrieve has the `x-amz-meta-title` metadata, the code prints the metadata value.

For instructions on how to create and test a working sample, see [Running the Amazon S3 .NET Code Examples \(p. 1193\)](#).

```
using Amazon;
using Amazon.S3;
using Amazon.S3.Model;
using System;
using System.IO;
using System.Threading.Tasks;

namespace Amazon.DocSamples.S3
{
    class GetObjectTest
    {
        private const string bucketName = "*** bucket name ***";
        private const string keyName = "*** object key ***";
        // Specify your bucket region (an example region is shown).
```

```
private static readonly RegionEndpoint bucketRegion = RegionEndpoint.USWest2;
private static IAmazonS3 client;

public static void Main()
{
    client = new AmazonS3Client(bucketRegion);
    ReadObjectDataAsync().Wait();
}

static async Task ReadObjectDataAsync()
{
    string responseBody = "";
    try
    {
        GetObjectRequest request = new GetObjectRequest
        {
            BucketName = bucketName,
            Key = keyName
        };
        using (GetObjectResponse response = await
client.GetObjectAsync(request))
            using (Stream responseStream = response.ResponseStream)
            using (StreamReader reader = new StreamReader(responseStream))
            {
                string title = response.Metadata["x-amz-meta-title"]; // Assume you
have "title" as medata added to the object.
                string contentType = response.Headers["Content-Type"];
                Console.WriteLine("Object metadata, Title: {0}", title);
                Console.WriteLine("Content type: {0}", contentType);

                responseBody = reader.ReadToEnd(); // Now you process the response
body.
            }
    }
    catch (AmazonS3Exception e)
    {
        // If bucket or object does not exist
        Console.WriteLine("Error encountered ***. Message:'{0}' when reading
object", e.Message);
    }
    catch (Exception e)
    {
        Console.WriteLine("Unknown encountered on server. Message:'{0}' when
reading object", e.Message);
    }
}
}
```

PHP

This topic explains how to use a class from the AWS SDK for PHP to retrieve an Amazon S3 object. You can retrieve an entire object or a byte range from the object. We assume that you are already following the instructions for [Using the AWS SDK for PHP and Running PHP Examples \(p. 1193\)](#) and have the AWS SDK for PHP properly installed.

When retrieving an object, you can optionally override the response header values by adding the response keys, `ResponseContentType`, `ResponseContentLanguage`, `ResponseContentDisposition`, `ResponseCacheControl`, and `ResponseExpires`, to the `getObject()` method, as shown in the following PHP code example:

Example

```
$result = $s3->getObject([
```

```
'Bucket'                => $bucket,
'Key'                   => $keyname,
'ResponseContentType'  => 'text/plain',
'ResponseContentLanguage' => 'en-US',
'ResponseContentDisposition' => 'attachment; filename=testing.txt',
'ResponseCacheControl'   => 'No-cache',
'ResponseExpires'        => gmdate(DATE_RFC2822, time() + 3600),
]);
];
```

For more information about retrieving objects, see [Downloading an object \(p. 209\)](#).

The following PHP example retrieves an object and displays the content of the object in the browser. The example shows how to use the `getObject()` method. For information about running the PHP examples in this guide, see [Running PHP Examples \(p. 1194\)](#).

```
require 'vendor/autoload.php';

use Aws\S3\S3Client;
use Aws\S3\Exception\S3Exception;

$bucket = '*** Your Bucket Name ***';
$keyname = '*** Your Object Key ***';

$s3 = new S3Client([
    'version' => 'latest',
    'region'  => 'us-east-1'
]);

try {
    // Get the object.
    $result = $s3->getObject([
        'Bucket' => $bucket,
        'Key'     => $keyname
    ]);

    // Display the object in the browser.
    header("Content-Type: {$result['ContentType']}");
    echo $result['Body'];
} catch (S3Exception $e) {
    echo $e->getMessage() . PHP_EOL;
}
```

Using the REST API

You can use the AWS SDK to retrieve object keys from a bucket. However, if your application requires it, you can send REST requests directly. You can send a GET request to retrieve object keys.

For more information about the request and response format, see [Get Object](#).

Using the AWS CLI

The example below shows you how you can use the AWS CLI to download an object from Amazon S3. For more information and examples, see [get-object](#) in the *AWS CLI Command Reference*.

```
aws s3api get-object --bucket DOC-EXAMPLE-BUCKET1 --key dir/my_images.tar.bz2
my_images.tar.bz2
```

Checking object integrity

Amazon S3 uses checksum values to verify the integrity of data that you upload to or download from Amazon S3. In addition, you can request that another checksum value be calculated for any object that you store in Amazon S3. You can select from one of several checksum algorithms to use when uploading or copying your data. Amazon S3 uses this algorithm to compute an additional checksum value and store it as part of the object metadata.

When you upload an object, you can optionally include a precalculated checksum as part of your request. Amazon S3 compares the provided checksum to the checksum that it calculates by using your specified algorithm. If the two values don't match, Amazon S3 reports an error.

Using supported checksum algorithms

Amazon S3 offers you the option to choose the checksum algorithm that is used to validate your data during upload or download. You can select one of the following Secure Hash Algorithms (SHA) or Cyclic Redundancy Check (CRC) data-integrity check algorithms:

- CRC32
- CRC32C
- SHA-1
- SHA-256

When you upload an object, you can specify the algorithm that you want to use:

- **When you're using the AWS Management Console**, you select the checksum algorithm that you want to use. When you do, you can optionally specify the checksum value of the object. When Amazon S3 receives the object, it calculates the checksum by using the algorithm that you specified. If the two checksum values don't match, Amazon S3 generates an error.
- **When you're using an SDK**, you can set the value of the `x-amz-sdk-checksum-algorithm` parameter to the algorithm that you want Amazon S3 to use when calculating the checksum. Amazon S3 automatically calculates the checksum value.
- **When you're using the REST API**, you don't use the `x-amz-sdk-checksum-algorithm` parameter. Instead, you use one of the algorithm-specific headers (for example, `x-amz-checksum-crc32`).

For more information about uploading objects, see [Uploading objects \(p. 158\)](#).

To apply any of these checksum values to objects that are already uploaded to Amazon S3, you can copy the object. When you copy an object, you can specify whether you want to use the existing checksum algorithm or use a new one. You can specify a checksum algorithm when using any supported mechanism for copying objects, including S3 Batch Operations. For more information about S3 Batch Operations, see [Performing large-scale batch operations on Amazon S3 objects \(p. 881\)](#).

Important

If you're using a multipart upload with additional checksums, the multipart part numbers must use consecutive part numbers. When using additional checksums, if you try to complete a multipart upload request with nonconsecutive part numbers, Amazon S3 generates an HTTP 500 Internal Server Error error.

After uploading objects, you can get the checksum value and compare it to a precalculated or previously stored checksum value calculated using the same algorithm.

Using the S3 console

To learn more about using the console and specifying checksum algorithms to use when uploading objects, see [Uploading objects \(p. 158\)](#)

Using the AWS SDKs

The following example shows how you can use the AWS SDKs to upload a large file with multipart upload, download a large file, and validate a multipart upload file, all with using SHA-256 for file validation.

Java

Example Example: Uploading, downloading, and verifying a large file with SHA-256

For instructions on creating and testing a working sample, see [Testing the Amazon S3 Java Code Examples \(p. 1192\)](#).

```
import software.amazon.awssdk.auth.credentials.AwsCredentials;
import software.amazon.awssdk.auth.credentials.AwsCredentialsProvider;
import software.amazon.awssdk.core.ResponseInputStream;
import software.amazon.awssdk.core.sync.RequestBody;
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.s3.S3Client;
import software.amazon.awssdk.services.s3.model.AbortMultipartUploadRequest;
import software.amazon.awssdk.services.s3.model.ChecksumAlgorithm;
import software.amazon.awssdk.services.s3.model.ChecksumMode;
import software.amazon.awssdk.services.s3.model.CompleteMultipartUploadRequest;
import software.amazon.awssdk.services.s3.model.CompleteMultipartUploadResponse;
import software.amazon.awssdk.services.s3.model.CompletedMultipartUpload;
import software.amazon.awssdk.services.s3.model.CompletedPart;
import software.amazon.awssdk.services.s3.model.CreateMultipartUploadRequest;
import software.amazon.awssdk.services.s3.model.CreateMultipartUploadResponse;
import software.amazon.awssdk.services.s3.model.GetObjectAttributesRequest;
import software.amazon.awssdk.services.s3.model.GetObjectAttributesResponse;
import software.amazon.awssdk.services.s3.model.GetObjectRequest;
import software.amazon.awssdk.services.s3.model.GetObjectResponse;
import software.amazon.awssdk.services.s3.model.GetObjectTaggingRequest;
import software.amazon.awssdk.services.s3.model.ObjectAttributes;
import software.amazon.awssdk.services.s3.model.PutObjectTaggingRequest;
import software.amazon.awssdk.services.s3.model.Tag;
import software.amazon.awssdk.services.s3.model.Tagging;
import software.amazon.awssdk.services.s3.model.UploadPartRequest;
import software.amazon.awssdk.services.s3.model.UploadPartResponse;

import java.io.File;
import java.io.FileInputStream;
import java.io.FileOutputStream;
import java.io.IOException;
import java.io.InputStream;
import java.io.OutputStream;
import java.nio.ByteBuffer;
import java.security.MessageDigest;
import java.security.NoSuchAlgorithmException;
import java.util.ArrayList;
import java.util.Base64;
import java.util.List;

public class LargeObjectValidation {
    private static String FILE_NAME = "sample.file";
    private static String BUCKET = "sample-bucket";
    //Optional, if you want a method of storing the full multipart object checksum
    in S3.
    private static String CHECKSUM_TAG_KEYNAME = "fullObjectChecksum";
```

```

//If you have existing full-object checksums that you need to validate against,
you can do the full object validation on a sequential upload.
private static String SHA256_FILE_BYTES = "htCM5g7ZNdoSw8bN/mkgiAhXt5MFOvowVg
+LE9aIQmI=";
//Example Chunk Size - this must be greater than or equal to 5MB.
private static int CHUNK_SIZE = 5 * 1024 * 1024;

public static void main(String[] args) {
    S3Client s3Client = S3Client.builder()
        .region(Region.US_EAST_1)
        .credentialsProvider(new AwsCredentialsProvider() {
            @Override
            public AwsCredentials resolveCredentials() {
                return new AwsCredentials();
            }
            @Override
            public String accessKeyId() {
                return Constants.ACCESS_KEY;
            }
            @Override
            public String secretAccessKey() {
                return Constants.SECRET;
            }
        });
    s3Client.build();
    uploadLargeFileBracketedByChecksum(s3Client);
    downloadLargeFileBracketedByChecksum(s3Client);
    validateExistingFileAgainstS3Checksum(s3Client);
}

public static void uploadLargeFileBracketedByChecksum(S3Client s3Client) {
    System.out.println("Starting uploading file validation");
    File file = new File(FILE_NAME);
    try (InputStream in = new FileInputStream(file)) {
        MessageDigest sha256 = MessageDigest.getInstance("SHA-256");
        CreateMultipartUploadRequest createMultipartUploadRequest =
CreateMultipartUploadRequest.builder()
        .bucket(BUCKET)
        .key(FILE_NAME)
        .checksumAlgorithm(ChecksumAlgorithm.SHA256)
        .build();
        CreateMultipartUploadResponse createdUpload =
s3Client.createMultipartUpload(createMultipartUploadRequest);
        List<CompletedPart> completedParts = new ArrayList<CompletedPart>();
        int partNumber = 1;
        byte[] buffer = new byte[CHUNK_SIZE];
        int read = in.read(buffer);
        while (read != -1) {
            UploadPartRequest uploadPartRequest = UploadPartRequest.builder()
                .partNumber(partNumber).uploadId(createdUpload.uploadId()).key(FILE_NAME).bucket(BUCKET).checksumAlgorithm(SHA256)
                UploadPartResponse uploadedPart =
s3Client.uploadPart(uploadPartRequest,
RequestBody.fromByteBuffer(ByteBuffer.wrap(buffer, 0, read)));
            CompletedPart part =
CompletedPart.builder().partNumber(partNumber).checksumSHA256(uploadedPart.checksumSHA256()).eTag(uploadedPart.eTag())
            completedParts.add(part);
            sha256.update(buffer, 0, read);
            read = in.read(buffer);
            partNumber++;
        }
        String fullObjectChecksum =
Base64.getEncoder().encodeToString(sha256.digest());
        if (!fullObjectChecksum.equals(SHA256_FILE_BYTES)) {
    }
}

```

```

//Because the SHA256 is uploaded after the part is uploaded; the
upload is bracketed and the full object can be fully validated.

s3Client.abortMultipartUpload(AbortMultipartUploadRequest.builder().bucket(BUCKET).key(FILE_NAME).
    throw new IOException("Byte mismatch between stored checksum and
upload, do not proceed with upload and cleanup");
}
CompletedMultipartUpload completedMultipartUpload =
CompletedMultipartUpload.builder().parts(completedParts).build();
CompleteMultipartUploadResponse completedUploadResponse =
s3Client.completeMultipartUpload()

CompleteMultipartUploadRequest.builder().bucket(BUCKET).key(FILE_NAME).uploadId(createdUpload.uploadId);
Tag checksumTag =
Tag.builder().key(CHECKSUM_TAG_KEYNAME).value(fullObjectChecksum).build();
//Optionally, if you need the full object checksum stored with the
file; you could add it as a tag after completion.

s3Client.putObjectTagging(PutObjectTaggingRequest.builder().bucket(BUCKET).key(FILE_NAME).tagging());
} catch (IOException | NoSuchAlgorithmException e) {
    e.printStackTrace();
}
GetObjectAttributesResponse
objectAttributes =
s3Client.getObjectAttributes(GetObjectAttributesRequest.builder().bucket(BUCKET).key(FILE_NAME)
    .objectAttributes(ObjectAttributes.OBJECT_PARTS,
ObjectAttributes.CHECKSUM).build());
System.out.println(objectAttributes.objectParts().parts());
System.out.println(objectAttributes.checksum().checksumSHA256());
}

public static void downloadLargeFileBracketedByChecksum(S3Client s3Client) {
    System.out.println("Starting downloading file validation");
    File file = new File("DOWNLOADED_" + FILE_NAME);
    try (OutputStream out = new FileOutputStream(file)) {
        GetObjectAttributesResponse
        objectAttributes =
s3Client.getObjectAttributes(GetObjectAttributesRequest.builder().bucket(BUCKET).key(FILE_NAME)
    .objectAttributes(ObjectAttributes.OBJECT_PARTS,
ObjectAttributes.CHECKSUM).build());
        //Optionally if you need the full object checksum, you can grab a tag
you added on the upload
        List<Tag> objectTags =
s3Client.getObjectTagging(GetObjectTaggingRequest.builder().bucket(BUCKET).key(FILE_NAME).build());
        String fullObjectChecksum = null;
        for (Tag objectTag : objectTags) {
            if (objectTag.key().equals(CHECKSUM_TAG_KEYNAME)) {
                fullObjectChecksum = objectTag.value();
                break;
            }
        }
        MessageDigest sha256FullObject = MessageDigest.getInstance("SHA-256");
        MessageDigest sha256ChecksumOfChecksums =
MessageDigest.getInstance("SHA-256");

        //If you retrieve the object in parts, and set the ChecksumMode to
enabled, the SDK will automatically validate the part checksum
        for (int partNumber = 1; partNumber <=
objectAttributes.objectParts().totalPartsCount(); partNumber++) {
            MessageDigest sha256Part = MessageDigest.getInstance("SHA-256");
            ResponseInputStream<GetObjectResponse> response =
s3Client.getObject(GetObjectRequest.builder().bucket(BUCKET).key(FILE_NAME).partNumber(partNumber));
            GetObjectResponse getObjectResponse = response.response();
            byte[] buffer = new byte[CHUNK_SIZE];
            int read = response.read(buffer);
            while (read != -1) {

```

```

        out.write(buffer, 0, read);
        sha256FullObject.update(buffer, 0, read);
        sha256Part.update(buffer, 0, read);
        read = response.read(buffer);
    }
    byte[] sha256PartBytes = sha256Part.digest();
    sha256ChecksumOfChecksums.update(sha256PartBytes);
    // Optionally, you can do an additional manual validation again the
part checksum if needed in addition to the SDK check
    String base64PartChecksum =
Base64.getEncoder().encodeToString(sha256PartBytes);
    String base64PartChecksumFromObjectAttributes =
objectAttributes.objectParts().parts().get(partNumber - 1).checksumSHA256();
    if (!base64PartChecksum.equals(getObjectResponse.checksumSHA256()))
|| !base64PartChecksum.equals(base64PartChecksumFromObjectAttributes)) {
        throw new IOException("Part checksum didn't match for the
part");
    }
    System.out.println(partNumber + " " + base64PartChecksum);
}
// Before finalizing, do the final checksum validation.
String base64FullObject =
Base64.getEncoder().encodeToString(sha256FullObject.digest());
String base64ChecksumOfChecksums =
Base64.getEncoder().encodeToString(sha256ChecksumOfChecksums.digest());
if (fullObjectChecksum != null && !
fullObjectChecksum.equals(base64FullObject)) {
    throw new IOException("Failed checksum validation for full
object");
}
System.out.println(fullObjectChecksum);
String base64ChecksumOfChecksumFromAttributes =
objectAttributes.checksum().checksumSHA256();
if (base64ChecksumOfChecksumFromAttributes != null && !
base64ChecksumOfChecksums.equals(base64ChecksumOfChecksumFromAttributes)) {
    throw new IOException("Failed checksum validation for full object
checksum of checksums");
}
System.out.println(base64ChecksumOfChecksumFromAttributes);
out.flush();
} catch (IOException | NoSuchAlgorithmException e) {
    // Cleanup bad file
    file.delete();
    e.printStackTrace();
}
}

public static void validateExistingFileAgainstS3Checksum(S3Client s3Client) {
    System.out.println("Starting existing file validation");
    File file = new File("DOWNLOADED_" + FILE_NAME);
    GetObjectAttributesResponse
        objectAttributes =
s3Client.getObjectAttributes(GetObjectAttributesRequest.builder().bucket(BUCKET).key(FILE_NAME)
        .objectAttributes(ObjectAttributes.OBJECT_PARTS,
ObjectAttributes.CHECKSUM).build());
    try (InputStream in = new FileInputStream(file)) {
        MessageDigest sha256ChecksumOfChecksums =
MessageDigest.getInstance("SHA-256");
        MessageDigest sha256Part = MessageDigest.getInstance("SHA-256");
        byte[] buffer = new byte[CHUNK_SIZE];
        int currentPart = 0;
        int partBreak =
objectAttributes.objectParts().parts().get(currentPart).size();
        int totalRead = 0;
        int read = in.read(buffer);
        while (read != -1) {

```

```
totalRead += read;
if (totalRead >= partBreak) {
    int difference = totalRead - partBreak;
    byte[] partChecksum;
    if (totalRead != partBreak) {
        sha256Part.update(buffer, 0, read - difference);
        partChecksum = sha256Part.digest();
        sha256ChecksumOfChecksums.update(partChecksum);
        sha256Part.reset();
        sha256Part.update(buffer, read - difference, difference);
    } else {
        sha256Part.update(buffer, 0, read);
        partChecksum = sha256Part.digest();
        sha256ChecksumOfChecksums.update(partChecksum);
        sha256Part.reset();
    }
    String base64PartChecksum =
Base64.getEncoder().encodeToString(partChecksum);
    if (!
base64PartChecksum.equals(objectAttributes.objectParts().parts().get(currentPart).checksumSHA256()))
{
    throw new IOException("Part checksum didn't match S3");
}
currentPart++;
System.out.println(currentPart + " " + base64PartChecksum);
if (currentPart <
objectAttributes.objectParts().totalPartsCount()) {
    partBreak +=
objectAttributes.objectParts().parts().get(currentPart - 1).size();
}
} else {
    sha256Part.update(buffer, 0, read);
}
read = in.read(buffer);
}
if (currentPart != objectAttributes.objectParts().totalPartsCount()) {
    currentPart++;
    byte[] partChecksum = sha256Part.digest();
    sha256ChecksumOfChecksums.update(partChecksum);
    String base64PartChecksum =
Base64.getEncoder().encodeToString(partChecksum);
    System.out.println(currentPart + " " + base64PartChecksum);
}

String base64CalculatedChecksumOfChecksums =
Base64.getEncoder().encodeToString(sha256ChecksumOfChecksums.digest());
System.out.println(base64CalculatedChecksumOfChecksums);
System.out.println(objectAttributes.checksum().checksumSHA256());
if (!
base64CalculatedChecksumOfChecksums.equals(objectAttributes.checksum().checksumSHA256())))
{
    throw new IOException("Full object checksum of checksums don't
match S3");
}

} catch (IOException | NoSuchAlgorithmException e) {
    e.printStackTrace();
}
}
}
```

Using the REST API

You can send REST requests to upload an object with a checksum value to verify the integrity of the data with [PutObject](#). You can also retrieve the checksum value for objects using [GetObject](#) or [HeadObject](#).

Using the AWS CLI

You can send a `PUT` request to upload an object of up to 5 GB in a single operation. For more information, see the [PutObject](#) in the *AWS CLI Command Reference*. You can also use `get-object` and `head-object` to retrieve the checksum of an already-uploaded object to verify the integrity of the data.

Using Content-MD5 when uploading objects

Another way to verify the integrity of your object after uploading is to provide an MD5 digest of the object when you upload it. If you calculate the MD5 digest for your object, you can provide the digest with the `PUT` command by using the `Content-MD5` header.

After uploading the object, Amazon S3 calculates the MD5 digest of the object and compares it to the value that you provided. The request succeeds only if the two digests match.

Supplying an MD5 digest isn't required, but you can use it to verify the integrity of the object as part of the upload process.

Using Content-MD5 and the ETag to verify uploaded objects

The entity tag (ETag) for an object represents a specific version of that object. Keep in mind that the ETag reflects changes only to the content of an object, not to its metadata. If only the metadata of an object changes, the ETag remains the same.

Depending on the object, the ETag of the object might be an MD5 digest of the object data:

- If an object is created by the `PUT Object`, `POST Object`, or `Copy` operation, or through the AWS Management Console, and that object is also plaintext or encrypted by server-side encryption with Amazon S3-managed keys (SSE-S3), that object has an ETag that is an MD5 digest of its object data.
- If an object is created by the `PUT Object`, `POST Object`, or `Copy` operation, or through the AWS Management Console, and that object is encrypted by server-side encryption with customer-provided keys (SSE-C) or server-side encryption with AWS Key Management Service (AWS KMS) keys (SSE-KMS), that object has an ETag that is not an MD5 digest of its object data.
- If an object is created by either the `Multipart Upload` or `Part Copy` operation, the object's ETag is not an MD5 digest, regardless of the method of encryption. If an object is larger than 16 MB, the AWS Management Console uploads or copies that object as a multipart upload, and therefore the ETag isn't an MD5 digest.

For objects where the ETag is the `Content-MD5` digest of the object, you can compare the ETag value of the object with a calculated or previously stored `Content-MD5` digest.

Using trailing checksums

When uploading objects to Amazon S3, you can either provide a precalculated checksum for the object or use an AWS SDK to automatically create trailing checksums on your behalf. If you decide to use a trailing checksum, Amazon S3 automatically generates the checksum by using your specified algorithm and uses it to validate the integrity of the object during upload.

To create a trailing checksum when using an AWS SDK, populate the `ChecksumAlgorithm` parameter with your preferred algorithm. The SDK uses that algorithm to calculate the checksum for your object (or object parts) and automatically appends it to the end of your upload request. This behavior saves you time because Amazon S3 performs both the verification and upload of your data in a single pass.

Important

If you're using S3 Object Lambda, all requests to S3 Object Lambda are signed using `s3-object-lambda` instead of `s3`. This behavior affects the signature of trailing checksum values. For more information about S3 Object Lambda, see [Transforming objects with S3 Object Lambda \(p. 271\)](#).

Using part-level checksums for multipart uploads

When objects are uploaded to Amazon S3, they can either be uploaded as a single object or through the multipart upload process. Objects that are larger than 16 MB and uploaded through the console are automatically uploaded using multipart uploads. For more information about multipart uploads, see [Uploading and copying objects using multipart upload \(p. 167\)](#).

When an object is uploaded as a multipart upload, the ETag for the object is not an MD5 digest of the entire object. Amazon S3 calculates the MD5 digest of each individual part as it is uploaded. The MD5 digests are used to determine the ETag for the final object. Amazon S3 concatenates the bytes for the MD5 digests together and then calculates the MD5 digest of these concatenated values. The final step for creating the ETag is when Amazon S3 adds a dash with the total number of parts to the end.

For example, consider an object uploaded with a multipart upload that has an ETag of `C9A5A6878D97B48CC965C1E41859F034-14`. In this case, `C9A5A6878D97B48CC965C1E41859F034` is the MD5 digest of all the digests concatenated together. The `-14` indicates that there are 14 parts associated with this object's multipart upload.

If you've enabled additional checksum values for your multipart object, Amazon S3 calculates the checksum for each individual part by using the specified checksum algorithm. The checksum for the completed object is calculated in the same way that Amazon S3 calculates the MD5 digest for the multipart upload. You can use this checksum to verify the integrity of the object.

To retrieve information about the object, including how many parts make up the entire object, you can use the [GetObjectAttributes](#) operation. With additional checksums, you can also recover information for each individual part that includes each part's checksum value.

Alternatively, you can get an individual part's checksum by using the [GetObject](#) or [HeadObject](#) operation and specifying a part number or byte range that aligns to a single part. With this method, you can use that checksum to validate the individual part without needing to wait for all of the parts to finish uploading before verifying the data integrity. When you use this method, you can also request only the individual parts that failed the integrity test.

Because of how Amazon S3 calculates the checksum for multipart objects, the checksum value for the object might change if you copy it. If you're using an SDK or the REST API and you call [CopyObject](#), Amazon S3 copies any object up to the size limitations of the `CopyObject` API operation. Amazon S3 does this copy as a single action, regardless of whether the object was uploaded in a single request or as part of a multipart upload. With a copy command, the checksum of the object is a direct checksum of the full object. If the object was originally uploaded using a multipart upload, then the checksum value changes even though the data has not.

Note

Objects that are larger than the size limitations of the `CopyObject` API operation must use multipart copy commands.

Important

When you perform some operations using the AWS Management Console, Amazon S3 uses a multipart upload if the object is greater than 16 MB in size. In this case, the checksum is not

a direct checksum of the full object, but rather a calculation based on the checksum values of each individual part.

For example, consider an object 100 MB in size that you uploaded as a single-part direct upload using the REST API. The checksum in this case is a checksum of the entire object. If you later use the console to rename that object, copy it, change the storage class, or edit the metadata, Amazon S3 uses the multipart upload functionality to update the object. As a result, Amazon S3 creates a new checksum value for the object that is calculated based on the checksum values of the individual parts.

The preceding list of console operations is not a complete list of all the possible actions that you can take in the AWS Management Console that result in Amazon S3 updating the object using the multipart upload functionality. Keep in mind that whenever you use the console to act on objects over 16 MB in size, the checksum value might not be the checksum of the entire object.

Deleting Amazon S3 objects

You can delete one or more objects directly from Amazon S3 using the Amazon S3 console, AWS SDKs, AWS Command Line Interface (AWS CLI), or REST API. Because all objects in your S3 bucket incur storage costs, you should delete objects that you no longer need. For example, if you're collecting log files, it's a good idea to delete them when they're no longer needed. You can set up a lifecycle rule to automatically delete objects such as log files. For more information, see [the section called "Setting lifecycle configuration" \(p. 708\)](#).

For information about Amazon S3 features and pricing, see [Amazon S3 pricing](#).

You have the following API options when deleting an object:

- **Delete a single object** — Amazon S3 provides the DELETE API that you can use to delete one object in a single HTTP request.
- **Delete multiple objects** — Amazon S3 provides the Multi-Object Delete API that you can use to delete up to 1,000 objects in a single HTTP request.

When deleting objects from a bucket that is not version-enabled, you provide only the object key name. However, when deleting objects from a version-enabled bucket, you can optionally provide the version ID of the object to delete a specific version of the object.

Programmatically deleting objects from a version-enabled bucket

If your bucket is version-enabled, multiple versions of the same object can exist in the bucket. When working with version-enabled buckets, the delete API enables the following options:

- **Specify a non-versioned delete request** — Specify only the object's key, and not the version ID. In this case, Amazon S3 creates a delete marker and returns its version ID in the response. This makes your object disappear from the bucket. For information about object versioning and the delete marker concept, see [Using versioning in S3 buckets \(p. 638\)](#).
- **Specify a versioned delete request** — Specify both the key and also a version ID. In this case the following two outcomes are possible:
 - If the version ID maps to a specific object version, Amazon S3 deletes the specific version of the object.
 - If the version ID maps to the delete marker of that object, Amazon S3 deletes the delete marker. This makes the object reappear in your bucket.

Deleting objects from an MFA-enabled bucket

When deleting objects from a multi-factor authentication (MFA)-enabled bucket, note the following:

- If you provide an invalid MFA token, the request always fails.
- If you have an MFA-enabled bucket, and you make a versioned delete request (you provide an object key and version ID), the request fails if you don't provide a valid MFA token. In addition, when using the Multi-Object Delete API on an MFA-enabled bucket, if any of the deletes are a versioned delete request (that is, you specify object key and version ID), the entire request fails if you don't provide an MFA token.

However, in the following cases the request succeeds:

- If you have an MFA-enabled bucket, and you make a non-versioned delete request (you are not deleting a versioned object), and you don't provide an MFA token, the delete succeeds.
- If you have a Multi-Object Delete request specifying only non-versioned objects to delete from an MFA-enabled bucket, and you don't provide an MFA token, the deletions succeed.

For information about MFA delete, see [Configuring MFA delete \(p. 648\)](#).

Topics

- [Deleting a single object \(p. 224\)](#)
- [Deleting multiple objects \(p. 231\)](#)

Deleting a single object

You can use the Amazon S3 console or the DELETE API to delete a single existing object from an S3 bucket. For more information about deleting objects in Amazon S3, see [Deleting Amazon S3 objects \(p. 223\)](#).

Because all objects in your S3 bucket incur storage costs, you should delete objects that you no longer need. For example, if you are collecting log files, it's a good idea to delete them when they're no longer needed. You can set up a lifecycle rule to automatically delete objects such as log files. For more information, see [the section called "Setting lifecycle configuration" \(p. 708\)](#).

For information about Amazon S3 features and pricing, see [Amazon S3 pricing](#).

Using the S3 console

Follow these steps to use the Amazon S3 console to delete a single object from a bucket.

To delete 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 **Bucket name** list, choose the name of the bucket that you want to delete an object from.
3. To delete an object in a versioning-enabled bucket with versioning:
 - **Off**, Amazon S3 creates a delete marker. To delete the object, select the object, and choose **delete** and confirm your choice by typing **delete** in the text field.
 - **On**, Amazon S3 will permanently delete the object version. Select the object version that you want to delete, and choose **delete** and confirm your choice by typing **permanently delete** in the text field.

Using the AWS SDKs

The following examples show how you can use the AWS SDKs to delete an object from a bucket. For more information, see [DELETE Object](#) in the *Amazon Simple Storage Service API Reference*

If you have S3 Versioning enabled on the bucket, you have the following options:

- Delete a specific object version by specifying a version ID.
- Delete an object without specifying a version ID, in which case Amazon S3 adds a delete marker to the object.

For more information about S3 Versioning, see [Using versioning in S3 buckets \(p. 638\)](#).

Java

Example Example 1: Deleting an object (non-versioned bucket)

The following example assumes that the bucket is not versioning-enabled and the object doesn't have any version IDs. In the delete request, you specify only the object key and not a version ID.

For instructions on creating and testing a working sample, see [Testing the Amazon S3 Java Code Examples \(p. 1192\)](#).

```
import com.amazonaws.AmazonServiceException;
import com.amazonaws.SdkClientException;
import com.amazonaws.auth.profile.ProfileCredentialsProvider;
import com.amazonaws.regions.Regions;
import com.amazonaws.services.s3.AmazonS3;
import com.amazonaws.services.s3.AmazonS3ClientBuilder;
import com.amazonaws.services.s3.model.DeleteObjectRequest;

import java.io.IOException;

public class DeleteObjectNonVersionedBucket {

    public static void main(String[] args) throws IOException {
        Regions clientRegion = Regions.DEFAULT_REGION;
        String bucketName = "*** Bucket name ***";
        String keyName = "*** Key name ***";

        try {
            AmazonS3 s3Client = AmazonS3ClientBuilder.standard()
                .withCredentials(new ProfileCredentialsProvider())
                .withRegion(clientRegion)
                .build();

            s3Client.deleteObject(new DeleteObjectRequest(bucketName, keyName));
        } catch (AmazonServiceException e) {
            // The call was transmitted successfully, but Amazon S3 couldn't process
            // it, so it returned an error response.
            e.printStackTrace();
        } catch (SdkClientException e) {
            // Amazon S3 couldn't be contacted for a response, or the client
            // couldn't parse the response from Amazon S3.
            e.printStackTrace();
        }
    }
}
```

Example Example 2: Deleting an object (versioned bucket)

The following example deletes an object from a versioned bucket. The example deletes a specific object version by specifying the object key name and version ID.

The example does the following:

1. Adds a sample object to the bucket. Amazon S3 returns the version ID of the newly added object. The example uses this version ID in the delete request.
2. Deletes the object version by specifying both the object key name and a version ID. If there are no other versions of that object, Amazon S3 deletes the object entirely. Otherwise, Amazon S3 only deletes the specified version.

Note

You can get the version IDs of an object by sending a `ListVersions` request.

```
import com.amazonaws.AmazonServiceException;
import com.amazonaws.SdkClientException;
import com.amazonaws.auth.profile.ProfileCredentialsProvider;
import com.amazonaws.regions.Regions;
import com.amazonaws.services.s3.AmazonS3;
import com.amazonaws.services.s3.AmazonS3ClientBuilder;
import com.amazonaws.services.s3.model.BucketVersioningConfiguration;
import com.amazonaws.services.s3.model.DeleteVersionRequest;
import com.amazonaws.services.s3.model.PutObjectResult;

import java.io.IOException;

public class DeleteObjectVersionEnabledBucket {

    public static void main(String[] args) throws IOException {
        Regions clientRegion = Regions.DEFAULT_REGION;
        String bucketName = "*** Bucket name ***";
        String keyName = "*** Key name ***";

        try {
            AmazonS3 s3Client = AmazonS3ClientBuilder.standard()
                .withCredentials(new ProfileCredentialsProvider())
                .withRegion(clientRegion)
                .build();

            // Check to ensure that the bucket is versioning-enabled.
            String bucketVersionStatus =
                s3Client.getBucketVersioningConfiguration(bucketName).getStatus();
            if (!bucketVersionStatus.equals(BucketVersioningConfiguration.ENABLED)) {
                System.out.printf("Bucket %s is not versioning-enabled.", bucketName);
            } else {
                // Add an object.
                PutObjectResult putResult = s3Client.putObject(bucketName, keyName,
                    "Sample content for deletion example.");
                System.out.printf("Object %s added to bucket %s\n", keyName,
                    bucketName);

                // Delete the version of the object that we just created.
                System.out.println("Deleting versioned object " + keyName);
                s3Client.deleteVersion(new DeleteVersionRequest(bucketName, keyName,
                    putResult.getVersionId()));
                System.out.printf("Object %s, version %s deleted\n", keyName,
                    putResult.getVersionId());
            }
        } catch (AmazonServiceException e) {
            // The call was transmitted successfully, but Amazon S3 couldn't process
        }
    }
}
```

```
        // it, so it returned an error response.
        e.printStackTrace();
    } catch (SdkClientException e) {
        // Amazon S3 couldn't be contacted for a response, or the client
        // couldn't parse the response from Amazon S3.
        e.printStackTrace();
    }
}
```

.NET

The following examples show how to delete an object from both versioned and non-versioned buckets. For more information about S3 Versioning, see [Using versioning in S3 buckets \(p. 638\)](#).

Example Deleting an object from a non-versioned bucket

The following C# example deletes an object from a non-versioned bucket. The example assumes that the objects don't have version IDs, so you don't specify version IDs. You specify only the object key.

For information about how to create and test a working sample, see [Running the Amazon S3 .NET Code Examples \(p. 1193\)](#).

```
using Amazon;
using Amazon.S3;
using Amazon.S3.Model;
using System;
using System.Threading.Tasks;

namespace Amazon.DocSamples.S3
{
    class DeleteObjectNonVersionedBucketTest
    {
        private const string bucketName = "*** bucket name ***";
        private const string keyName = "*** object key ***";
        // Specify your bucket region (an example region is shown).
        private static readonly RegionEndpoint bucketRegion = RegionEndpoint.USWest2;
        private static IAmazonS3 client;

        public static void Main()
        {
            client = new AmazonS3Client(bucketRegion);
            DeleteObjectNonVersionedBucketAsync().Wait();
        }
        private static async Task DeleteObjectNonVersionedBucketAsync()
        {
            try
            {
                var deleteObjectRequest = new DeleteObjectRequest
                {
                    BucketName = bucketName,
                    Key = keyName
                };

                Console.WriteLine("Deleting an object");
                await client.DeleteObjectAsync(deleteObjectRequest);
            }
            catch (AmazonS3Exception e)
            {
                Console.WriteLine("Error encountered on server. Message:{0} when
deleting an object", e.Message);
            }
        }
}
```

```
        catch (Exception e)
    {
        Console.WriteLine("Unknown encountered on server. Message:{0}' when
deleting an object", e.Message);
    }
}
```

Example Deleting an object from a versioned bucket

The following C# example deletes an object from a versioned bucket. It deletes a specific version of the object by specifying the object key name and version ID.

The code performs the following tasks:

1. Enables S3 Versioning on a bucket that you specify (if S3 Versioning is already enabled, this has no effect).
2. Adds a sample object to the bucket. In response, Amazon S3 returns the version ID of the newly added object. The example uses this version ID in the delete request.
3. Deletes the sample object by specifying both the object key name and a version ID.

Note

You can also get the version ID of an object by sending a `ListVersions` request.

```
var listResponse = client.ListVersions(new ListVersionsRequest { BucketName =
bucketName, Prefix = keyName });
```

For information about how to create and test a working sample, see [Running the Amazon S3 .NET Code Examples \(p. 1193\)](#).

```
using Amazon;
using Amazon.S3;
using Amazon.S3.Model;
using System;
using System.Threading.Tasks;

namespace Amazon.DocSamples.S3
{
    class DeleteObjectVersion
    {
        private const string bucketName = "*** versioning-enabled bucket name ***";
        private const string keyName = "*** Object Key Name ***";
        // Specify your bucket region (an example region is shown).
        private static readonly RegionEndpoint bucketRegion = RegionEndpoint.USWest2;
        private static IAmazonS3 client;

        public static void Main()
        {
            client = new AmazonS3Client(bucketRegion);
            CreateAndDeleteObjectVersionAsync().Wait();
        }

        private static async Task CreateAndDeleteObjectVersionAsync()
        {
            try
            {
                // Add a sample object.
                string versionID = await PutAnObject(keyName);
```

```
// Delete the object by specifying an object key and a version ID.
DeleteObjectRequest request = new DeleteObjectRequest
{
    BucketName = bucketName,
    Key = keyName,
    VersionId = versionID
};
Console.WriteLine("Deleting an object");
await client.DeleteObjectAsync(request);
}
catch (AmazonS3Exception e)
{
    Console.WriteLine("Error encountered on server. Message:{0}' when
deleting an object", e.Message);
}
catch (Exception e)
{
    Console.WriteLine("Unknown encountered on server. Message:{0}' when
deleting an object", e.Message);
}
}

static async Task<string> PutAnObject(string objectKey)
{
    PutObjectRequest request = new PutObjectRequest
    {
        BucketName = bucketName,
        Key = objectKey,
        ContentBody = "This is the content body!"
    };
    PutObjectResponse response = await client.PutObjectAsync(request);
    return response.VersionId;
}
}
```

PHP

This example shows how to use classes from version 3 of the AWS SDK for PHP to delete an object from a non-versioned bucket. For information about deleting an object from a versioned bucket, see [Using the REST API \(p. 231\)](#).

This example assumes that you are already following the instructions for [Using the AWS SDK for PHP and Running PHP Examples \(p. 1193\)](#) and have the AWS SDK for PHP properly installed. For information about running the PHP examples in this guide, see [Running PHP Examples \(p. 1194\)](#).

The following PHP example deletes an object from a bucket. Because this example shows how to delete objects from non-versioned buckets, it provides only the bucket name and object key (not a version ID) in the delete request.

```
<?php

require 'vendor/autoload.php';

use Aws\S3\S3Client;
use Aws\S3\Exception\S3Exception;

$bucket = '*** Your Bucket Name ***';
$keyname = '*** Your Object Key ***';

$s3 = new S3Client([
    'version' => 'latest',
    'region'  => 'us-east-1'
```

```
]);
// 1. Delete the object from the bucket.
try
{
    echo 'Attempting to delete ' . $keyname . '...' . PHP_EOL;

    $result = $s3->deleteObject([
        'Bucket' => $bucket,
        'Key'     => $keyname
    ]);

    if ($result['DeleteMarker'])
    {
        echo $keyname . ' was deleted or does not exist.' . PHP_EOL;
    } else {
        exit('Error: ' . $keyname . ' was not deleted.' . PHP_EOL);
    }
}
catch (S3Exception $e) {
    exit('Error: ' . $e->getAwsErrorMessage() . PHP_EOL);
}

// 2. Check to see if the object was deleted.
try
{
    echo 'Checking to see if ' . $keyname . ' still exists...' . PHP_EOL;

    $result = $s3->getObject([
        'Bucket' => $bucket,
        'Key'     => $keyname
    ]);

    echo 'Error: ' . $keyname . ' still exists.';
}
catch (S3Exception $e) {
    exit($e->getAwsErrorMessage());
}
```

Javascript

This example shows how to use version 3 of the AWS SDK for JavaScript to delete an object. For more information about AWS SDK for JavaScript see, [Using the AWS SDK for JavaScript \(p. 1196\)](#).

```
import { DeleteObjectCommand } from "@aws-sdk/client-s3";
import { s3Client } from "./libs/s3Client.js" // Helper function that creates Amazon S3 service client module.

export const bucketParams = { Bucket: "BUCKET_NAME", Key: "KEY" };

export const run = async () => {
    try {
        const data = await s3Client.send(new DeleteObjectCommand(bucketParams));
        console.log("Success. Object deleted.", data);
        return data; // For unit tests.
    } catch (err) {
        console.log("Error", err);
    }
};

run();
```

Using the AWS CLI

To delete one object per request, use the [DELETE API](#). For more information, see [DELETE Object](#). For more information about using the CLI to delete an object, see [delete-object](#).

Using the REST API

You can use the AWS SDKs to delete an object. However, if your application requires it, you can send REST requests directly. For more information, see [DELETE Object](#) in the *Amazon Simple Storage Service API Reference*.

Deleting multiple objects

Because all objects in your S3 bucket incur storage costs, you should delete objects that you no longer need. For example, if you are collecting log files, it's a good idea to delete them when they're no longer needed. You can set up a lifecycle rule to automatically delete objects such as log files. For more information, see [the section called "Setting lifecycle configuration" \(p. 708\)](#).

For information about Amazon S3 features and pricing, see [Amazon S3 pricing](#).

You can use the Amazon S3 console or the Multi-Object Delete API to delete multiple objects simultaneously from an S3 bucket.

Using the S3 console

Follow these steps to use the Amazon S3 console to delete multiple objects from a bucket.

To delete objects

1. Sign in to the AWS Management Console and open the Amazon S3 console at <https://console.aws.amazon.com/s3/>.
2. Navigate to the Amazon S3 bucket or folder that contains the objects that you want to delete.
3. Select the check box to the left of the names of the objects that you want to delete.
4. Choose **Actions** and choose **Delete** from the list of options that appears.

Alternatively, choose **Delete** from the options in the upper right.

5. Enter **delete** if asked to confirm that you want to delete these objects.
6. Choose **Delete objects** in the bottom right and Amazon S3 deletes the specified objects.

Warning

- Deleting the specified objects cannot be undone.
- This action deletes all specified objects. When deleting folders, wait for the delete action to finish before adding new objects to the folder. Otherwise, new objects might be deleted as well.
- To delete an object in a bucket with bucket versioning **disabled** or **suspended**, Amazon S3 creates a delete marker. To undo the delete action, delete this delete marker. To confirm this action, type **permanently delete**.
- To delete an object version in a bucket with bucket versioning **enabled**, Amazon S3 will permanently delete the object version. To confirm this action, type **delete**.

Using the AWS SDKs

Amazon S3 provides the Multi-Object Delete API , which you can use to delete multiple objects in a single request. The API supports two modes for the response: *verbose* and *quiet*. By default, the operation uses verbose mode. In verbose mode, the response includes the result of the deletion of

each key that is specified in your request. In quiet mode, the response includes only keys for which the delete operation encountered an error. If all keys are successfully deleted when you're using quiet mode, Amazon S3 returns an empty response. For more information, see [Delete - Multi-Object Delete](#).

To learn more about object deletion, see [Deleting Amazon S3 objects \(p. 223\)](#).

Java

The AWS SDK for Java provides the `AmazonS3Client.deleteObjects()` method for deleting multiple objects. For each object that you want to delete, you specify the key name. If the bucket is versioning-enabled, you have the following options:

- Specify only the object's key name. Amazon S3 adds a delete marker to the object.
- Specify both the object's key name and a version ID to be deleted. Amazon S3 deletes the specified version of the object.

Example

The following example uses the Multi-Object Delete API to delete objects from a bucket that is not version-enabled. The example uploads sample objects to the bucket and then uses the `AmazonS3Client.deleteObjects()` method to delete the objects in a single request. In the `DeleteObjectsRequest`, the example specifies only the object key names because the objects do not have version IDs.

For instructions on creating and testing a working sample, see [Testing the Amazon S3 Java Code Examples \(p. 1192\)](#).

```
import com.amazonaws.AmazonServiceException;
import com.amazonaws.SdkClientException;
import com.amazonaws.regions.Regions;
import com.amazonaws.services.s3.AmazonS3;
import com.amazonaws.services.s3.AmazonS3ClientBuilder;
import com.amazonaws.services.s3.model.DeleteObjectsRequest;
import com.amazonaws.services.s3.model.DeleteObjectsRequest.KeyVersion;
import com.amazonaws.services.s3.model.DeleteObjectsResult;

import java.io.IOException;
import java.util.ArrayList;

public class DeleteMultipleObjectsNonVersionedBucket {

    public static void main(String[] args) throws IOException {
        Regions clientRegion = Regions.DEFAULT_REGION;
        String bucketName = "*** Bucket name ***";

        try {
            AmazonS3 s3Client = AmazonS3ClientBuilder.standard()
                .withRegion(clientRegion)
                .build();

            // Upload three sample objects.
            ArrayList<KeyVersion> keys = new ArrayList<KeyVersion>();
            for (int i = 0; i < 3; i++) {
                String keyName = "delete object example " + i;
                s3Client.putObject(bucketName, keyName, "Object number " + i + " to be
deleted.");
                keys.add(new KeyVersion(keyName));
            }
            System.out.println(keys.size() + " objects successfully created.");
        } catch (AmazonServiceException | SdkClientException e) {
            System.out.println("Error occurred while deleting objects: " + e.getMessage());
        }
    }
}
```

```
// Delete the sample objects.  
DeleteObjectsRequest multiObjectDeleteRequest = new  
DeleteObjectsRequest(bucketName)  
    .withKeys(keys)  
    .withQuiet(false);  
  
// Verify that the objects were deleted successfully.  
DeleteObjectsResult delObjRes =  
s3Client.deleteObjects(multiObjectDeleteRequest);  
int successfulDeletes = delObjRes.getDeletedObjects().size();  
System.out.println(successfulDeletes + " objects successfully deleted.");  
} catch (AmazonServiceException e) {  
    // The call was transmitted successfully, but Amazon S3 couldn't process  
    // it, so it returned an error response.  
    e.printStackTrace();  
} catch (SdkClientException e) {  
    // Amazon S3 couldn't be contacted for a response, or the client  
    // couldn't parse the response from Amazon S3.  
    e.printStackTrace();  
}  
}
```

Example

The following example uses the Multi-Object Delete API to delete objects from a version-enabled bucket. It does the following:

- Creates sample objects and then deletes them, specifying the key name and version ID for each object to delete. The operation deletes only the specified object versions.
 - Creates sample objects and then deletes them by specifying only the key names. Because the example doesn't specify version IDs, the operation adds a delete marker to each object, without deleting any specific object versions. After the delete markers are added, these objects will not appear in the AWS Management Console.
 - Removes the delete markers by specifying the object keys and version IDs of the delete markers. The operation deletes the delete markers, which results in the objects reappearing in the AWS Management Console.

```
import com.amazonaws.AmazonServiceException;
import com.amazonaws.SdkClientException;
import com.amazonaws.auth.profile.ProfileCredentialsProvider;
import com.amazonaws.regions.Regions;
import com.amazonaws.services.s3.AmazonS3;
import com.amazonaws.services.s3.AmazonS3ClientBuilder;
import com.amazonaws.services.s3.model.BucketVersioningConfiguration;
import com.amazonaws.services.s3.model.DeleteObjectsRequest;
import com.amazonaws.services.s3.model.DeleteObjectsRequest.KeyVersion;
import com.amazonaws.services.s3.model.DeleteObjectsResult;
import com.amazonaws.services.s3.model.DeleteObjectsResult.DeletedObject;
import com.amazonaws.services.s3.model.PutObjectResult;

import java.io.IOException;
import java.util.ArrayList;
import java.util.List;

public class DeleteMultipleObjectsVersionEnabledBucket {
    private static AmazonS3 S3_CLIENT;
    private static String VERSIONED_BUCKET_NAME;
```

```

public static void main(String[] args) throws IOException {
    Regions clientRegion = Regions.DEFAULT_REGION;
    VERSIONED_BUCKET_NAME = "*** Bucket name ***";

    try {
        S3_CLIENT = AmazonS3ClientBuilder.standard()
            .withCredentials(new ProfileCredentialsProvider())
            .withRegion(clientRegion)
            .build();

        // Check to make sure that the bucket is versioning-enabled.
        String bucketVersionStatus =
        S3_CLIENT.getBucketVersioningConfiguration(VERSIONED_BUCKET_NAME).getStatus();
        if (!bucketVersionStatus.equals(BucketVersioningConfiguration.ENABLED)) {
            System.out.printf("Bucket %s is not versioning-enabled.", VERSIONED_BUCKET_NAME);
        } else {
            // Upload and delete sample objects, using specific object versions.
            uploadAndDeleteObjectsWithVersions();

            // Upload and delete sample objects without specifying version IDs.
            // Amazon S3 creates a delete marker for each object rather than
            deleting
            // specific versions.
            DeleteObjectsResult unversionedDeleteResult =
            uploadAndDeleteObjectsWithoutVersions();

            // Remove the delete markers placed on objects in the non-versioned
            create/delete method.
            multiObjectVersionedDeleteRemoveDeleteMarkers(unversionedDeleteResult);
        }
    } catch (AmazonServiceException e) {
        // The call was transmitted successfully, but Amazon S3 couldn't process
        // it, so it returned an error response.
        e.printStackTrace();
    } catch (SdkClientException e) {
        // Amazon S3 couldn't be contacted for a response, or the client
        // couldn't parse the response from Amazon S3.
        e.printStackTrace();
    }
}

private static void uploadAndDeleteObjectsWithVersions() {
    System.out.println("Uploading and deleting objects with versions specified.");

    // Upload three sample objects.
    ArrayList<KeyVersion> keys = new ArrayList<KeyVersion>();
    for (int i = 0; i < 3; i++) {
        String keyName = "delete object without version ID example " + i;
        PutObjectResult putResult = S3_CLIENT.putObject(VERSIONED_BUCKET_NAME,
keyName,
            "Object number " + i + " to be deleted.");
        // Gather the new object keys with version IDs.
        keys.add(new KeyVersion(keyName, putResult.getVersionId()));
    }

    // Delete the specified versions of the sample objects.
    DeleteObjectsRequest multiObjectDeleteRequest = new
DeleteObjectsRequest(VERSIONED_BUCKET_NAME)
        .withKeys(keys)
        .withQuiet(false);

    // Verify that the object versions were successfully deleted.
    DeleteObjectsResult delObjRes =
    S3_CLIENT.deleteObjects(multiObjectDeleteRequest);
    int successfulDeletes = delObjRes.getDeletedObjects().size();
}

```

```

        System.out.println(successfulDeletes + " objects successfully deleted");
    }

    private static DeleteObjectsResult uploadAndDeleteObjectsWithoutVersions() {
        System.out.println("Uploading and deleting objects with no versions
specified.");

        // Upload three sample objects.
        ArrayList<KeyVersion> keys = new ArrayList<KeyVersion>();
        for (int i = 0; i < 3; i++) {
            String keyName = "delete object with version ID example " + i;
            S3_CLIENT.putObject(VERSIONED_BUCKET_NAME, keyName, "Object number " + i +
" to be deleted.");
            // Gather the new object keys without version IDs.
            keys.add(new KeyVersion(keyName));
        }

        // Delete the sample objects without specifying versions.
        DeleteObjectsRequest multiObjectDeleteRequest = new
DeleteObjectsRequest(VERSIONED_BUCKET_NAME).withKeys(keys)
            .withQuiet(false);

        // Verify that delete markers were successfully added to the objects.
        DeleteObjectsResult delObjRes =
S3_CLIENT.deleteObjects(multiObjectDeleteRequest);
        int successfulDeletes = delObjRes.getDeletedObjects().size();
        System.out.println(successfulDeletes + " objects successfully marked for
deletion without versions.");
        return delObjRes;
    }

    private static void
multiObjectVersionedDeleteRemoveDeleteMarkers(DeleteObjectsResult response) {
    List<KeyVersion> keyList = new ArrayList<KeyVersion>();
    for (DeletedObject deletedObject : response.getDeletedObjects()) {
        // Note that the specified version ID is the version ID for the delete
marker.
        keyList.add(new KeyVersion(deletedObject.getKey(),
deletedObject.getDeleteMarkerVersionId()));
    }
    // Create a request to delete the delete markers.
    DeleteObjectsRequest deleteRequest = new
DeleteObjectsRequest(VERSIONED_BUCKET_NAME).withKeys(keyList);

    // Delete the delete markers, leaving the objects intact in the bucket.
    DeleteObjectsResult delObjRes = S3_CLIENT.deleteObjects(deleteRequest);
    int successfulDeletes = delObjRes.getDeletedObjects().size();
    System.out.println(successfulDeletes + " delete markers successfully deleted");
}
}

```

.NET

The AWS SDK for .NET provides a convenient method for deleting multiple objects: `DeleteObjects`. For each object that you want to delete, you specify the key name and the version of the object. If the bucket is not versioning-enabled, you specify `null` for the version ID. If an exception occurs, review the `DeleteObjectsException` response to determine which objects were not deleted and why.

Example Deleting multiple objects from a non-versioning bucket

The following C# example uses the multi-object delete API to delete objects from a bucket that is not version-enabled. The example uploads the sample objects to the bucket, and then uses the

DeleteObjects method to delete the objects in a single request. In the DeleteObjectsRequest, the example specifies only the object key names because the version IDs are null.

For information about creating and testing a working sample, see [Running the Amazon S3 .NET Code Examples \(p. 1193\)](#).

```
using Amazon;
using Amazon.S3;
using Amazon.S3.Model;
using System;
using System.Collections.Generic;
using System.Threading.Tasks;

namespace Amazon.DocSamples.S3
{
    class DeleteMultipleObjectsNonVersionedBucketTest
    {
        private const string bucketName = "*** versioning-enabled bucket name ***";
        // Specify your bucket region (an example region is shown).
        private static readonly RegionEndpoint bucketRegion = RegionEndpoint.USWest2;
        private static IAmazonS3 s3Client;

        public static void Main()
        {
            s3Client = new AmazonS3Client(bucketRegion);
            MultiObjectDeleteAsync().Wait();
        }

        static async Task MultiObjectDeleteAsync()
        {
            // Create sample objects (for subsequent deletion).
            var keysAndVersions = await PutObjectsAsync(3);

            // a. multi-object delete by specifying the key names and version IDs.
            DeleteObjectsRequest multiObjectDeleteRequest = new DeleteObjectsRequest
            {
                BucketName = bucketName,
                Objects = keysAndVersions // This includes the object keys and null
version IDs.
            };
            // You can add specific object key to the delete request using the .AddKey.
            // multiObjectDeleteRequest.AddKey("TickerReference.csv", null);
            try
            {
                DeleteObjectsResponse response = await
s3Client.DeleteObjectsAsync(multiObjectDeleteRequest);
                Console.WriteLine("Successfully deleted all the {0} items",
response.DeletedObjects.Count);
            }
            catch (DeleteObjectsException e)
            {
                PrintDeletionErrorStatus(e);
            }
        }

        private static void PrintDeletionErrorStatus(DeleteObjectsException e)
        {
            // var errorResponse = e.ErrorResponse;
            DeleteObjectsResponse errorResponse = e.Response;
            Console.WriteLine("x {0}", errorResponse.DeletedObjects.Count);

            Console.WriteLine("No. of objects successfully deleted = {0}",
errorResponse.DeletedObjects.Count);
            Console.WriteLine("No. of objects failed to delete = {0}",
errorResponse.DeleteErrors.Count);
        }
    }
}
```

```

Console.WriteLine("Printing error data...");
foreach (DeleteError deleteError in errorResponse.DeleteErrors)
{
    Console.WriteLine("Object Key: {0}\t{1}\t{2}", deleteError.Key,
deleteError.Code, deleteError.Message);
}
}

static async Task<List<KeyVersion>> PutObjectsAsync(int number)
{
    List<KeyVersion> keys = new List<KeyVersion>();
    for (int i = 0; i < number; i++)
    {
        string key = "ExampleObject-" + new System.Random().Next();
        PutObjectRequest request = new PutObjectRequest
        {
            BucketName = bucketName,
            Key = key,
            ContentBody = "This is the content body!",
        };

        PutObjectResponse response = await s3Client.PutObjectAsync(request);
        KeyVersion keyVersion = new KeyVersion
        {
            Key = key,
            // For non-versioned bucket operations, we only need object key.
            // VersionId = response.VersionId
        };
        keys.Add(keyVersion);
    }
    return keys;
}
}

```

Example Multi-object deletion for a version-enabled bucket

The following C# example uses the multi-object delete API to delete objects from a version-enabled bucket. The example performs the following actions:

- Creates sample objects and deletes them by specifying the key name and version ID for each object. The operation deletes specific versions of the objects.
 - Creates sample objects and deletes them by specifying only the key names. Because the example doesn't specify version IDs, the operation only adds delete markers. It doesn't delete any specific versions of the objects. After deletion, these objects don't appear in the Amazon S3 console.
 - Deletes the delete markers by specifying the object keys and version IDs of the delete markers. When the operation deletes the delete markers, the objects reappear in the console.

For information about creating and testing a working sample, see [Running the Amazon S3 .NET Code Examples \(p. 1193\)](#).

```
using Amazon;
using Amazon.S3;
using Amazon.S3.Model;
using System;
using System.Collections.Generic;
using System.Threading.Tasks;

namespace Amazon.DocSamples.S3
{
```

```
class DeleteMultipleObjVersionedBucketTest
{
    private const string bucketName = "*** versioning-enabled bucket name ***";
    // Specify your bucket region (an example region is shown).
    private static readonly RegionEndpoint bucketRegion = RegionEndpoint.USWest2;
    private static IAmazonS3 s3Client;

    public static void Main()
    {
        s3Client = new AmazonS3Client(bucketRegion);
        DeleteMultipleObjectsFromVersionedBucketAsync().Wait();
    }

    private static async Task DeleteMultipleObjectsFromVersionedBucketAsync()
    {

        // Delete objects (specifying object version in the request).
        await DeleteObjectVersionsAsync();

        // Delete objects (without specifying object version in the request).
        var deletedObjects = await DeleteObjectsAsync();

        // Additional exercise - remove the delete markers S3 returned in the
        preceding response.
        // This results in the objects reappearing in the bucket (you can
        // verify the appearance/disappearance of objects in the console).
        await RemoveDeleteMarkersAsync(deletedObjects);
    }

    private static async Task<List<DeletedObject>> DeleteObjectsAsync()
    {
        // Upload the sample objects.
        var keysAndVersions2 = await PutObjectsAsync(3);

        // Delete objects using only keys. Amazon S3 creates a delete marker and
        // returns its version ID in the response.
        List<DeletedObject> deletedObjects = await
NonVersionedDeleteAsync(keysAndVersions2);
        return deletedObjects;
    }

    private static async Task DeleteObjectVersionsAsync()
    {
        // Upload the sample objects.
        var keysAndVersions1 = await PutObjectsAsync(3);

        // Delete the specific object versions.
        await VersionedDeleteAsync(keysAndVersions1);
    }

    private static void PrintDeletionReport(DeleteObjectsException e)
    {
        var errorResponse = e.Response;
        Console.WriteLine("No. of objects successfully deleted = {0}",
errorResponse.DeletedObjects.Count);
        Console.WriteLine("No. of objects failed to delete = {0}",
errorResponse.DeleteErrors.Count);
        Console.WriteLine("Printing error data...");
        foreach (var deleteError in errorResponse.DeleteErrors)
        {
            Console.WriteLine("Object Key: {0}\t{1}\t{2}", deleteError.Key,
deleteError.Code, deleteError.Message);
        }
    }

    static async Task VersionedDeleteAsync(List<KeyVersion> keys)
```

```
{  
    // a. Perform a multi-object delete by specifying the key names and version  
    IDs.  
    var multiObjectDeleteRequest = new DeleteObjectsRequest  
    {  
        BucketName = bucketName,  
        Objects = keys // This includes the object keys and specific version  
        IDs.  
    };  
    try  
    {  
        Console.WriteLine("Executing VersionedDelete...");  
        DeleteObjectsResponse response = await  
s3Client.DeleteObjectsAsync(multiObjectDeleteRequest);  
        Console.WriteLine("Successfully deleted all the {0} items",  
response.DeletedObjects.Count);  
    }  
    catch (DeleteObjectsException e)  
    {  
        PrintDeletionReport(e);  
    }  
}  
  
static async Task<List<DeletedObject>> NonVersionedDeleteAsync(List<KeyVersion>  
keys)  
{  
    // Create a request that includes only the object key names.  
    DeleteObjectsRequest multiObjectDeleteRequest = new DeleteObjectsRequest();  
    multiObjectDeleteRequest.BucketName = bucketName;  
  
    foreach (var key in keys)  
    {  
        multiObjectDeleteRequest.AddKey(key.Key);  
    }  
    // Execute DeleteObjects - Amazon S3 add delete marker for each object  
    // deletion. The objects disappear from your bucket.  
    // You can verify that using the Amazon S3 console.  
    DeleteObjectsResponse response;  
    try  
    {  
        Console.WriteLine("Executing NonVersionedDelete...");  
        response = await s3Client.DeleteObjectsAsync(multiObjectDeleteRequest);  
        Console.WriteLine("Successfully deleted all the {0} items",  
response.DeletedObjects.Count);  
    }  
    catch (DeleteObjectsException e)  
    {  
        PrintDeletionReport(e);  
        throw; // Some deletes failed. Investigate before continuing.  
    }  
    // This response contains the DeletedObjects list which we use to delete  
    the delete markers.  
    return response.DeletedObjects;  
}  
  
private static async Task RemoveDeleteMarkersAsync(List<DeletedObject>  
deletedObjects)  
{  
    var keyVersionList = new List<KeyVersion>();  
  
    foreach (var deletedObject in deletedObjects)  
    {  
        KeyVersion keyVersion = new KeyVersion  
        {  
            Key = deletedObject.Key,  
            VersionId = deletedObject.DeleteMarkerVersionId  
        };  
        keyVersionList.Add(keyVersion);  
    }  
    // Remove the delete markers.  
    await s3Client.PutObjectVersionsAsync(bucketName, keyVersionList);  
}
```

```
        };
        keyVersionList.Add(keyVersion);
    }
    // Create another request to delete the delete markers.
    var multiObjectDeleteRequest = new DeleteObjectsRequest
    {
        BucketName = bucketName,
        Objects = keyVersionList
    };

    // Now, delete the delete marker to bring your objects back to the bucket.
    try
    {
        Console.WriteLine("Removing the delete markers .....");
        var deleteObjectResponse = await
s3Client.DeleteObjectsAsync(multiObjectDeleteRequest);
        Console.WriteLine("Successfully deleted all the {0} delete markers",
                           deleteObjectResponse.DeletedObjects.Count);
    }
    catch (DeleteObjectsException e)
    {
        PrintDeletionReport(e);
    }
}

static async Task<List<KeyVersion>> PutObjectsAsync(int number)
{
    var keys = new List<KeyVersion>();

    for (var i = 0; i < number; i++)
    {
        string key = "ObjectToDelete-" + new System.Random().Next();
        PutObjectRequest request = new PutObjectRequest
        {
            BucketName = bucketName,
            Key = key,
            ContentBody = "This is the content body!",
        };

        var response = await s3Client.PutObjectAsync(request);
        KeyVersion keyVersion = new KeyVersion
        {
            Key = key,
            VersionId = response.VersionId
        };

        keys.Add(keyVersion);
    }
    return keys;
}
}
```

PHP

These examples show how to use classes from version 3 of the AWS SDK for PHP to delete multiple objects from versioned and non-versioned Amazon S3 buckets. For more information about versioning, see [Using versioning in S3 buckets \(p. 638\)](#).

The examples assume that you are already following the instructions for [Using the AWS SDK for PHP and Running PHP Examples \(p. 1193\)](#) and have the AWS SDK for PHP properly installed.

Example Deleting multiple objects from a non-versioned bucket

The following PHP example uses the `deleteObjects()` method to delete multiple objects from a bucket that is not version-enabled.

For information about running the PHP examples in this guide, see [Running PHP Examples \(p. 1194\)](#).

```
<?php

require 'vendor/autoload.php';

use Aws\S3\S3Client;

$bucket = '*** Your Bucket Name ***';

$s3 = new S3Client([
    'version' => 'latest',
    'region'  => 'us-east-1'
]);

// 1. Create a few objects.
for ($i = 1; $i <= 3; $i++) {
    $s3->putObject([
        'Bucket' => $bucket,
        'Key'     => "key{$i}",
        'Body'    => "content {$i}",
    ]);
}

// 2. List the objects and get the keys.
$keys = $s3->listObjects([
    'Bucket' => $bucket
]);

// 3. Delete the objects.
foreach ($keys['Contents'] as $key)
{
    $s3->deleteObjects([
        'Bucket' => $bucket,
        'Delete'  => [
            'Objects' => [
                [
                    'Key' => $key['Key']
                ]
            ]
        ]
    ]);
}
```

Example Deleting multiple objects from a version-enabled bucket

The following PHP example uses the `deleteObjects()` method to delete multiple objects from a version-enabled bucket.

For information about running the PHP examples in this guide, see [Running PHP Examples \(p. 1194\)](#).

```
<?php

require 'vendor/autoload.php';

use Aws\S3\S3Client;
```

```

$bucket = '*** Your Bucket Name ***';
$keyname = '*** Your Object Key ***';

$s3 = new S3Client([
    'version' => 'latest',
    'region'  => 'us-east-1'
]);

// 1. Enable object versioning for the bucket.
$s3->putBucketVersioning([
    'Bucket' => $bucket,
    'VersioningConfiguration' => [
        'Status' => 'Enabled'
    ]
]);

// 2. Create a few versions of an object.
for ($i = 1; $i <= 3; $i++) {
    $s3->putObject([
        'Bucket' => $bucket,
        'Key'     => $keyname,
        'Body'    => "content {$i}",
    ]);
}

// 3. List the objects versions and get the keys and version IDs.
$versions = $s3->listObjectVersions(['Bucket' => $bucket]);

// 4. Delete the object versions.
$deletedResults = 'The following objects were deleted successfully:' . PHP_EOL;
$deleted = false;
$errorResults = 'The following objects could not be deleted:' . PHP_EOL;
$errors = false;

foreach ($versions['Versions'] as $version)
{
    $result = $s3->deleteObjects([
        'Bucket'  => $bucket,
        'Delete'   => [
            'Objects' => [
                [
                    'Key'    => $version['Key'],
                    'VersionId' => $version['VersionId']
                ]
            ]
        ]
    ]);

    if (isset($result['Deleted']))
    {
        $deleted = true;

        $deletedResults .= "Key: {$result['Deleted'][0]['Key']}, ".
                          "VersionId: {$result['Deleted'][0]['VersionId']} " . PHP_EOL;
    }

    if (isset($result['Errors']))
    {
        $errors = true;

        $errorResults .= "Key: {$result['Errors'][0]['Key']}, ".
                        "VersionId: {$result['Errors'][0]['VersionId']}, ".
                        "Message: {$result['Errors'][0]['Message']} " . PHP_EOL;
    }
}

```

```
if ($deleted)
{
    echo $deletedResults;
}

if ($errors)
{
    echo $errorResults;
}

// 5. Suspend object versioning for the bucket.
$s3->putBucketVersioning([
    'Bucket' => $bucket,
    'VersioningConfiguration' => [
        'Status' => 'Suspended'
    ]
]);

```

Using the REST API

You can use the AWS SDKs to delete multiple objects using the Multi-Object Delete API. However, if your application requires it, you can send REST requests directly.

For more information, see [Delete Multiple Objects](#) in the *Amazon Simple Storage Service API Reference*.

Organizing, listing, and working with your objects

In Amazon S3, you can use prefixes to organize your storage. A prefix is a logical grouping of the objects in a bucket. The prefix value is similar to a directory name that enables you to store similar data under the same directory in a bucket. When you programmatically upload objects, you can use prefixes to organize your data.

In the Amazon S3 console, prefixes are called folders. You can view all your objects and folders in the S3 console by navigating to a bucket. You can also view information about each object, including object properties.

For more information about listing and organizing your data in Amazon S3, see the following topics.

Topics

- [Organizing objects using prefixes \(p. 243\)](#)
- [Listing object keys programmatically \(p. 245\)](#)
- [Organizing objects in the Amazon S3 console using folders \(p. 254\)](#)
- [Viewing an object overview in the Amazon S3 console \(p. 256\)](#)
- [Viewing object properties in the Amazon S3 console \(p. 256\)](#)

Organizing objects using prefixes

You can use prefixes to organize the data that you store in Amazon S3 buckets. A prefix is a string of characters at the beginning of the object key name. A prefix can be any length, subject to the maximum length of the object key name (1,024 bytes). You can think of prefixes as a way to organize your data in a similar way to directories. However, prefixes are not directories.

Searching by prefix limits the results to only those keys that begin with the specified prefix. The delimiter causes a list operation to roll up all the keys that share a common prefix into a single summary list result.

The purpose of the prefix and delimiter parameters is to help you organize and then browse your keys hierarchically. To do this, first pick a delimiter for your bucket, such as slash (/), that doesn't occur in any of your anticipated key names. You can use another character as a delimiter. There is nothing unique about the slash (/) character, but it is a very common prefix delimiter. Next, construct your key names by concatenating all containing levels of the hierarchy, separating each level with the delimiter.

For example, if you were storing information about cities, you might naturally organize them by continent, then by country, then by province or state. Because these names don't usually contain punctuation, you might use slash (/) as the delimiter. The following examples use a slash (/) delimiter.

- Europe/France/Nouvelle-Aquitaine/Bordeaux
- North America/Canada/Quebec/Montreal
- North America/USA/Washington/Bellevue
- North America/USA/Washington/Seattle

If you stored data for every city in the world in this manner, it would become awkward to manage a flat key namespace. By using `Prefix` and `Delimiter` with the list operation, you can use the hierarchy that you've created to list your data. For example, to list all the states in USA, set `Delimiter='/'` and `Prefix='North America/USA/'`. To list all the provinces in Canada for which you have data, set `Delimiter='/'` and `Prefix='North America/Canada/'`.

Listing objects using prefixes and delimiters

If you issue a list request with a delimiter, you can browse your hierarchy at only one level, skipping over and summarizing the (possibly millions of) keys nested at deeper levels. For example, assume that you have a bucket (`DOC-EXAMPLE-BUCKET`) with the following keys:

```
sample.jpg  
photos/2006/January/sample.jpg  
photos/2006/February/sample2.jpg  
photos/2006/February/sample3.jpg  
photos/2006/February/sample4.jpg
```

The sample bucket has only the `sample.jpg` object at the root level. To list only the root level objects in the bucket, you send a GET request on the bucket with the slash (/) delimiter character. In response, Amazon S3 returns the `sample.jpg` object key because it does not contain the / delimiter character. All other keys contain the delimiter character. Amazon S3 groups these keys and returns a single `CommonPrefixes` element with the prefix value `photos/`, which is a substring from the beginning of these keys to the first occurrence of the specified delimiter.

Example

```
<ListBucketResult xmlns="http://s3.amazonaws.com/doc/2006-03-01/">  
  <Name>DOC-EXAMPLE-BUCKET</Name>  
  <Prefix></Prefix>  
  <Marker></Marker>  
  <MaxKeys>1000</MaxKeys>  
  <Delimiter>/</Delimiter>  
  <IsTruncated>false</IsTruncated>  
  <Contents>  
    <Key>sample.jpg</Key>  
    <LastModified>2011-07-24T19:39:30.000Z</LastModified>  
    <ETag>"d1a7fb5eab1c16cb4f7cf341cf188c3d"</ETag>  
    <Size>6</Size>  
    <Owner>
```

```
<ID>75cc57f09aa0c8caeab4f8c24e99d10f8e7faeefbf76c078efc7c6caea54ba06a</ID>
<DisplayName>displayname</DisplayName>
</Owner>
<StorageClass>STANDARD</StorageClass>
</Contents>
<CommonPrefixes>
  <Prefix>photos/</Prefix>
</CommonPrefixes>
</ListBucketResult>
```

For more information about listing object keys programmatically, see [Listing object keys programmatically \(p. 245\)](#).

Listing object keys programmatically

In Amazon S3, keys can be listed by prefix. You can choose a common prefix for the names of related keys and mark these keys with a special character that delimits hierarchy. You can then use the list operation to select and browse keys hierarchically. This is similar to how files are stored in directories within a file system.

Amazon S3 exposes a list operation that lets you enumerate the keys contained in a bucket. Keys are selected for listing by bucket and prefix. For example, consider a bucket named "dictionary" that contains a key for every English word. You might make a call to list all the keys in that bucket that start with the letter "q". List results are always returned in UTF-8 binary order.

Both the SOAP and REST list operations return an XML document that contains the names of matching keys and information about the object identified by each key.

Note

SOAP support over HTTP is deprecated, but SOAP is still available over HTTPS. New Amazon S3 features are not supported for SOAP. Instead of using SOAP, we recommend that you use either the REST API or the AWS SDKs.

Groups of keys that share a prefix terminated by a special delimiter can be rolled up by that common prefix for the purposes of listing. This enables applications to organize and browse their keys hierarchically, much like how you would organize your files into directories in a file system.

For example, to extend the dictionary bucket to contain more than just English words, you might form keys by prefixing each word with its language and a delimiter, such as "French/logical". Using this naming scheme and the hierarchical listing feature, you could retrieve a list of only French words. You could also browse the top-level list of available languages without having to iterate through all the lexicographically intervening keys. For more information about this aspect of listing, see [Organizing objects using prefixes \(p. 243\)](#).

REST API

If your application requires it, you can send REST requests directly. You can send a GET request to return some or all of the objects in a bucket or you can use selection criteria to return a subset of the objects in a bucket. For more information, see [GET Bucket \(List Objects\) Version 2](#) in the *Amazon Simple Storage Service API Reference*.

List implementation efficiency

List performance is not substantially affected by the total number of keys in your bucket. It's also not affected by the presence or absence of the `prefix`, `marker`, `maxkeys`, or `delimiter` arguments.

Iterating through multipage results

As buckets can contain a virtually unlimited number of keys, the complete results of a list query can be extremely large. To manage large result sets, the Amazon S3 API supports pagination to split them into multiple responses. Each list keys response returns a page of up to 1,000 keys with an indicator

indicating if the response is truncated. You send a series of list keys requests until you have received all the keys. AWS SDK wrapper libraries provide the same pagination.

Examples

The following code examples show how to list objects in an S3 bucket.

.NET

AWS SDK for .NET

Tip

To learn how to set up and run this example, see [GitHub](#).

```
/// <summary>
/// Shows how to list the objects in an Amazon S3 bucket.
/// </summary>
/// <param name="client">An initialized Amazon S3 client object.</param>
/// <param name="bucketName">The name of the bucket for which to list
/// the contents.</param>
/// <returns>A boolean value indicating the success or failure of the
/// copy operation.</returns>
public static async Task<bool> ListBucketContentsAsync(IAmazonS3 client,
string bucketName)
{
    try
    {
        var request = new ListObjectsV2Request
        {
            BucketName = bucketName,
            MaxKeys = 5,
        };

        Console.WriteLine("-----");
        Console.WriteLine($"Listing the contents of {bucketName}:");
        Console.WriteLine("-----");

        var response = new ListObjectsV2Response();

        do
        {
            response = await client.ListObjectsV2Async(request);

            response.S3Objects
                .ForEach(obj => Console.WriteLine($"{obj.Key,-35}{obj.LastModified.ToShortDateString(),10}{obj.Size,10}"));

            // If the response is truncated, set the request
ContinuationToken
                // from the NextContinuationToken property of the response.
                request.ContinuationToken = response.NextContinuationToken;
        }
        while (response.IsTruncated);

        return true;
    }
    catch (AmazonS3Exception ex)
    {
        Console.WriteLine($"Error encountered on server.
Message:{ex.Message}' getting list of objects.");
        return false;
    }
}
```

```
}
```

- For API details, see [ListObjects in AWS SDK for .NET API Reference](#).

C++

SDK for C++

Tip

To learn how to set up and run this example, see [GitHub](#).

```
bool AwsDoc::S3::ListObjects(const Aws::String& bucketName,
    const Aws::String& region)
{
    Aws::Client::ClientConfiguration config;

    if (!region.empty())
    {
        config.region = region;
    }

    Aws::S3::S3Client s3_client(config);

    Aws::S3::Model::ListObjectsRequest request;
    request.WithBucket(bucketName);

    auto outcome = s3_client.ListObjects(request);

    if (outcome.IsSuccess())
    {
        std::cout << "Objects in bucket '" << bucketName << "'":
        << std::endl << std::endl;

        Aws::Vector<Aws::S3::Model::Object> objects =
            outcome.GetResult().GetContents();

        for (Aws::S3::Model::Object& object : objects)
        {
            std::cout << object.GetKey() << std::endl;
        }

        return true;
    }
    else
    {
        std::cout << "Error: ListObjects: " <<
            outcome.GetError().GetMessage() << std::endl;

        return false;
    }
}

int main()
{
    Aws::SDKOptions options;
    Aws::InitAPI(options);
    {

        //TODO: Name of a bucket in your account.
        //The bucket must have at least one object in it. One way to achieve
        //this is to configure and run put_object.cpp's executable first.
        const Aws::String bucket_name = "DOC-EXAMPLE-BUCKET";
    }
}
```

```
//TODO: Set to the AWS Region in which the bucket was created.  
Aws::String region = "us-east-1";  
  
if (!AwsDoc::S3::ListObjects(bucket_name, region))  
{  
    return 1;  
}  
  
}  
Aws::ShutdownAPI(options);  
  
return 0;  
}
```

- For API details, see [ListObjects in AWS SDK for C++ API Reference](#).

Go

SDK for Go V2

Tip

To learn how to set up and run this example, see [GitHub](#).

```
// List objects in the bucket.  
// n.b. object keys in Amazon S3 do not begin with '/'. You do not need to lead  
your  
// prefix with it.  
fmt.Println("Listing the objects in the bucket:")  
listObjsResponse, err := client.ListObjectsV2(context.TODO(),  
&s3.ListObjectsV2Input{  
    Bucket: aws.String(name),  
    Prefix: aws.String(""),  
})  
  
if err != nil {  
    panic("Couldn't list bucket contents")  
}  
  
for _, object := range listObjsResponse.Contents {  
    fmt.Printf("%s (%d bytes, class %v) \n", *object.Key, object.Size,  
    object.StorageClass)  
}
```

- For API details, see [ListObjects in AWS SDK for Go API Reference](#).

Java

SDK for Java 2.x

Tip

To learn how to set up and run this example, see [GitHub](#).

```
public static void listBucketObjects(S3Client s3, String bucketName ) {  
  
try {  
    ListObjectsRequest listObjects = ListObjectsRequest  
        .builder()
```

```
.bucket(bucketName)
.build();

ListObjectsResponse res = s3.listObjects(listObjects);
List<S3Object> objects = res.contents();
for (S3Object myValue : objects) {
    System.out.print("\n The name of the key is " + myValue.key());
    System.out.print("\n The object is " + calKb(myValue.size()) + "
KBs");
    System.out.print("\n The owner is " + myValue.owner());
}

} catch (S3Exception e) {
    System.err.println(e.awsErrorDetails().errorMessage());
    System.exit(1);
}
}

//convert bytes to kbs.
private static long calKb(Long val) {
    return val/1024;
}
```

- For API details, see [ListObjects in AWS SDK for Java 2.x API Reference](#).

JavaScript

SDK for JavaScript V3

Tip

To learn how to set up and run this example, see [GitHub](#).

Create the client.

```
// Create service client module using ES6 syntax.
import { S3Client } from "@aws-sdk/client-s3";
// Set the AWS Region.
const REGION = "us-east-1";
// Create an Amazon S3 service client object.
const s3Client = new S3Client({ region: REGION });
export { s3Client };
```

List the objects.

```
// Import required AWS SDK clients and commands for Node.js.
import { ListObjectsCommand } from "@aws-sdk/client-s3";
import { s3Client } from "./libs/s3Client.js"; // Helper function that creates an
Amazon S3 service client module.

// Create the parameters for the bucket
export const bucketParams = { Bucket: "BUCKET_NAME" };

export const run = async () => {
    try {
        const data = await s3Client.send(new ListObjectsCommand(bucketParams));
        console.log("Success", data);
        return data; // For unit tests.
    } catch (err) {
        console.log("Error", err);
    }
};
```

```
run();
```

List 1000 or more objects.

```
// Import required AWS SDK clients and commands for Node.js.
import { ListObjectsCommand } from "@aws-sdk/client-s3";
import { s3Client } from "./libs/s3Client.js"; // Helper function that creates an
Amazon S3 service client module.

// Create the parameters for the bucket
export const bucketParams = { Bucket: "BUCKET_NAME" };

export async function run() {
    // Declare truncated as a flag that the while loop is based on.
    let truncated = true;
    // Declare a variable to which the key of the last element is assigned to in the
    response.
    let pageMarker;
    // while loop that runs until 'response.truncated' is false.
    while (truncated) {
        try {
            const response = await s3Client.send(new ListObjectsCommand(bucketParams));
            // return response; //For unit tests
            response.Contents.forEach((item) => {
                console.log(item.Key);
            });
            // Log the key of every item in the response to standard output.
            truncated = response.IsTruncated;
            // If truncated is true, assign the key of the last element in the response
            to the pageMarker variable.
            if (truncated) {
                pageMarker = response.Contents.slice(-1)[0].Key;
                // Assign the pageMarker value to bucketParams so that the next iteration
                starts from the new pageMarker.
                bucketParams.Marker = pageMarker;
            }
            // At end of the list, response.truncated is false, and the function exits
            the while loop.
        } catch (err) {
            console.log("Error", err);
            truncated = false;
        }
    }
}
run();
```

- For API details, see [ListObjects in AWS SDK for JavaScript API Reference](#).

Kotlin

SDK for Kotlin

Note

This is prerelease documentation for a feature in preview release. It is subject to change.

Tip

To learn how to set up and run this example, see [GitHub](#).

```
suspend fun listBucketObjects(bucketName: String) {
```

```
val request = ListObjectsRequest {
    bucket = bucketName
}

S3Client { region = "us-east-1" }.use { s3 ->

    val response = s3.listObjects(request)
    response.contents?.forEach { myObject ->
        println("The name of the key is ${myObject.key}")
        println("The object is ${calKb(myObject.size)} KBs")
        println("The owner is ${myObject.owner}")
    }
}

private fun calKb(intValue: Long): Long {
    return intValue / 1024
}
```

- For API details, see [ListObjects in AWS SDK for Kotlin API reference](#).

PHP

SDK for PHP

Tip

To learn how to set up and run this example, see [GitHub](#).

List objects in a bucket.

```
$s3client = new Aws\S3\S3Client(['region' => 'us-west-2', 'version' => 'latest']);

try {
    $contents = $s3client->listObjects([
        'Bucket' => $bucket_name,
    ]);
    echo "The contents of your bucket are: \n";
    foreach ($contents['Contents'] as $content) {
        echo $content['Key'] . "\n";
    }
} catch (Exception $exception) {
    echo "Failed to list objects in $bucket_name with error: " . $exception-
>getMessage();
    exit("Please fix error with listing objects before continuing.");
}
```

- For API details, see [ListObjects in AWS SDK for PHP API Reference](#).

Python

SDK for Python (Boto3)

Tip

To learn how to set up and run this example, see [GitHub](#).

```
class ObjectWrapper:
    def __init__(self, s3_object):
        self.object = s3_object
```

```
        self.key = self.object.key

    @staticmethod
    def list(bucket, prefix=None):
        """
            Lists the objects in a bucket, optionally filtered by a prefix.

        :param bucket: The bucket to query.
        :param prefix: When specified, only objects that start with this prefix are
        listed.
        :return: The list of objects.
        """
        try:
            if not prefix:
                objects = list(bucket.objects.all())
            else:
                objects = list(bucket.objects.filter(Prefix=prefix))
            logger.info("Got objects %s from bucket '%s',
                        [o.key for o in objects], bucket.name)
        except ClientError:
            logger.exception("Couldn't get objects for bucket '%s'.", bucket.name)
            raise
        else:
            return objects
```

- For API details, see [ListObjects in AWS SDK for Python \(Boto3\) API Reference](#).

Ruby

SDK for Ruby

Tip

To learn how to set up and run this example, see [GitHub](#).

```
require "aws-sdk-s3"

# Wraps Amazon S3 bucket actions.
class BucketListObjectsWrapper
  attr_reader :bucket

  # @param bucket [Aws::S3::Bucket] An existing Amazon S3 bucket.
  def initialize(bucket)
    @bucket = bucket
  end

  # Lists object in a bucket.
  #
  # @param max_objects [Integer] The maximum number of objects to list.
  # @return [Integer] The number of objects listed.
  def list_objects(max_objects)
    count = 0
    puts "The objects in #{@bucket.name} are:"
    @bucket.objects.each do |obj|
      puts "\t#{obj.key}"
      count += 1
      break if count == max_objects
    end
    count
  rescue Aws::Errors::ServiceError => e
    puts "Couldn't list objects in bucket #{@bucket.name}. Here's why: #{e.message}"
    0
  end
```

```
end

def run_demo
    bucket_name = "doc-example-bucket"

    wrapper = BucketListObjectsWrapper.new(Aws::S3::Bucket.new(bucket_name))
    count = wrapper.list_objects(25)
    puts "Listed #{count} objects."
end

run_demo if $PROGRAM_NAME == __FILE__
```

- For API details, see [ListObjects in AWS SDK for Ruby API Reference](#).

Rust

SDK for Rust

Note

This documentation is for an SDK in preview release. The SDK is subject to change and should not be used in production.

Tip

To learn how to set up and run this example, see [GitHub](#).

```
pub async fn list_objects(client: &Client, bucket_name: &str) -> Result<(), Error> {
    let objects = client.list_objects_v2().bucket(bucket_name).send().await?;
    println!("Objects in bucket:");
    for obj in objects.contents().unwrap_or_default() {
        println!("{}: {}", obj.key().unwrap());
    }
    Ok(())
}
```

- For API details, see [ListObjects in AWS SDK for Rust API reference](#).

Swift

SDK for Swift

Note

This is prerelease documentation for an SDK in preview release. It is subject to change.

Tip

To learn how to set up and run this example, see [GitHub](#).

```
public func listBucketFiles(bucket: String) async throws -> [String] {
    let input = ListObjectsV2Input(
        bucket: bucket
    )
    let output = try await client.listObjectsV2(input: input)
    var names: [String] = []

    guard let objList = output.contents else {
        return []
    }
```

```
        for obj in objList {
            if let objName = obj.key {
                names.append(objName)
            }
        }

        return names
    }
```

- For API details, see [ListObjects in AWS SDK for Swift API reference](#).

Organizing objects in the Amazon S3 console using folders

In Amazon S3, buckets and objects are the primary resources, and objects are stored in buckets. Amazon S3 has a flat structure instead of a hierarchy like you would see in a file system. However, for the sake of organizational simplicity, the Amazon S3 console supports the *folder* concept as a means of grouping objects. It does this by using a shared name prefix for objects (that is, objects have names that begin with a common string). Object names are also referred to as *key names*.

For example, you can create a folder on the console named `photos` and store an object named `myphoto.jpg` in it. The object is then stored with the key name `photos/myphoto.jpg`, where `photos/` is the prefix.

Here are two more examples:

- If you have three objects in your bucket—`logs/date1.txt`, `logs/date2.txt`, and `logs/date3.txt`—the console will show a folder named `logs`. If you open the folder in the console, you will see three objects: `date1.txt`, `date2.txt`, and `date3.txt`.
- If you have an object named `photos/2017/example.jpg`, the console will show you a folder named `photos` containing the folder `2017`. The folder `2017` will contain the object `example.jpg`.

You can have folders within folders, but not buckets within buckets. You can upload and copy objects directly into a folder. Folders can be created, deleted, and made public, but they cannot be renamed. Objects can be copied from one folder to another.

Important

When you use the Amazon S3 console to create a folder, Amazon S3 creates a 0-byte object with a key that's set to the folder name that you provided. For example, if you create a folder named `photos` in your bucket, the Amazon S3 console creates a 0-byte object with the key `photos/`. The console creates this object to support the idea of folders.

The Amazon S3 console treats all objects that have a forward slash (/) character as the last (trailing) character in the key name as a folder (for example, `examplekeyname/`). You can't upload an object that has a key name with a trailing / character by using the Amazon S3 console. However, you can upload objects that are named with a trailing / with the Amazon S3 API by using the AWS CLI, AWS SDKs, or REST API.

An object that is named with a trailing / appears as a folder in the Amazon S3 console. The Amazon S3 console does not display the content and metadata for such an object. When you use the console to copy an object named with a trailing /, a new folder is created in the destination location, but the object's data and metadata are not copied.

Topics

- [Creating a folder \(p. 255\)](#)
- [Making folders public \(p. 255\)](#)
- [Deleting folders \(p. 255\)](#)

Creating a folder

This section describes how to use the Amazon S3 console to create a folder.

Important

If your bucket policy prevents uploading objects to this bucket without tags, metadata, or access control list (ACL) grantees, you will not be able to create a folder using this configuration. Instead, upload an empty folder and specify these settings in the upload configuration.

To create a folder

1. Sign in to the AWS Management Console and open the Amazon S3 console at <https://console.aws.amazon.com/s3/>.
2. In the **Buckets** list, choose the name of the bucket that you want to create a folder in.
3. If your bucket policy prevents uploading objects to this bucket without encryption, you must choose **Enable** under **Server-side encryption**.
4. Choose **Create folder**.
5. Enter a name for the folder (for example, `favorite-pics`). Then choose **Create folder**.

Making folders public

We recommend blocking all public access to your Amazon S3 folders and buckets unless you specifically require a public folder or bucket. When you make a folder public, anyone on the internet can view all the objects that are grouped in that folder.

In the Amazon S3 console, you can make a folder public. You can also make a folder public by creating a bucket policy that limits access by prefix. For more information, see [Identity and access management in Amazon S3 \(p. 394\)](#).

Warning

After you make a folder public in the Amazon S3 console, you can't make it private again. Instead, you must set permissions on each individual object in the public folder so that the objects have no public access. For more information, see [Configuring ACLs \(p. 562\)](#).

Deleting folders

This section explains how to use the Amazon S3 console to delete folders from an S3 bucket.

For information about Amazon S3 features and pricing, see [Amazon S3](#).

To delete folders from an S3 bucket

1. Sign in to the AWS Management Console and open the Amazon S3 console at <https://console.aws.amazon.com/s3/>.
2. In the **Buckets** list, choose the name of the bucket that you want to delete folders from.
3. In the **Objects** list, select the check box next to the folders and objects that you want to delete.
4. Choose **Delete**.
5. On the **Delete objects** page, verify that the names of the folders you selected for deletion are listed.
6. In the **Delete objects** box, enter `delete`, and choose **Delete objects**.

Warning

This action deletes all specified objects. When deleting folders, wait for the delete action to finish before adding new objects to the folder. Otherwise, new objects might be deleted as well.

Viewing an object overview in the Amazon S3 console

You can use the Amazon S3 console to view an overview of an object. The object overview in the console provides all the essential information for an object in one place.

To open the overview pane for 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 **Buckets** list, choose the name of the bucket that contains the object.
 3. In the **Objects** list, choose the name of the object for which you want an overview.
- The object overview opens.
4. To download the object, choose **Object actions**, and then choose **Download**. To copy the path of the object to the clipboard, under **Object URL**, choose the URL.
 5. If versioning is enabled on the bucket, choose **Versions** to list the versions of the object.
 - To download an object version, select the check box next to the version ID, choose **Actions**, and then choose **Download**.
 - To delete an object version, select the check box next to the version ID, and choose **Delete**.

Important

You can undelete an object only if it was deleted as the latest (current) version. You can't undelete a previous version of an object that was deleted.

Viewing object properties in the Amazon S3 console

You can use the Amazon S3 console to view the properties of an object, including storage class, encryption settings, tags, and metadata.

To view the properties 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 **Buckets** list, choose the name of the bucket that contains the object.
3. In the **Objects** list, choose the name of the object you want to view properties for.

The **Object overview** for your object opens. You can scroll down to view the object properties.

4. On the **Object overview** page, you can configure the following properties for the object.

Note

If you change the **Storage Class**, **Encryption**, or **Metadata** properties, a new object is created to replace the old one. If S3 Versioning is enabled, a new version of the object is created, and the existing object becomes an older version. The role that changes the property also becomes the owner of the new object or (object version).

- a. **Storage class** – Each object in Amazon S3 has a storage class associated with it. The storage class that you choose to use depends on how frequently you access the object. The default storage class for S3 objects is STANDARD. You choose which storage class to use when you upload an object. For more information about storage classes, see [Using Amazon S3 storage classes \(p. 688\)](#).

To change the storage class after you upload an object, choose **Storage class**. Choose the storage class that you want, and then choose **Save**.

- b. **Server-side encryption settings** – You can use server-side encryption to encrypt your S3 objects. For more information, see [Specifying server-side encryption with AWS KMS \(SSE-KMS\) \(p. 342\)](#) or [Specifying Amazon S3 encryption \(p. 357\)](#).
- c. **Metadata** – Each object in Amazon S3 has a set of name-value pairs that represents its metadata. For information about adding metadata to an S3 object, see [Editing object metadata in the Amazon S3 console \(p. 157\)](#).
- d. **Tags** – You categorize storage by adding tags to an S3 object. For more information, see [Categorizing your storage using tags \(p. 825\)](#).
- e. **Object lock legal hold and rentention** – You can prevent an object from being deleted. For more information, see [Using S3 Object Lock \(p. 680\)](#).

Using presigned URLs

All objects and buckets are private by default. However, you can use a presigned URL to optionally share objects or allow your customers/users to upload objects to buckets without AWS security credentials or permissions.

You can use presigned URLs to generate a URL that can be used to access your Amazon S3 buckets. When you create a presigned URL, you associate it with a specific action. You can share the URL, and anyone with access to it can perform the action embedded in the URL as if they were the original signing user. The URL will expire and no longer work when it reaches its expiration time.

Important

For all Regions that launched after March 20, 2019, if a request arrives at the wrong Amazon S3 location, Amazon S3 returns an HTTP 400 Bad Request error.

Limiting presigned URL capabilities

The capabilities of a presigned URL are limited by the permissions of the user who created it. In essence, presigned URLs are a bearer token that grants access to those who possess them. As such, we recommend that you protect them appropriately.

If you want to restrict the use of presigned URLs and all S3 access to particular network paths, you can write AWS Identity and Access Management (IAM) policies that require a particular network path. You can set these policies on the IAM principal that makes the call, the Amazon S3 bucket, or both.

A network-path restriction on the principal requires the user of those credentials to make requests from the specified network. A restriction on the bucket limits access to that bucket only to requests that originate from the specified network. Realize that these restrictions also apply outside of the presigned URL scenario.

The IAM global condition that you use depends on the type of endpoint. If you are using the public endpoint for Amazon S3, use `aws:SourceIp`. If you are using a VPC endpoint to Amazon S3, use `aws:SourceVpc` or `aws:SourceVpce`.

The following IAM policy statement requires the principal to access AWS only from the specified network range. With this policy statement in place, all access is required to originate from that range. This includes the case of someone using a presigned URL for S3.

```
{  
    "Sid": "NetworkRestrictionForIAMPPrincipal",  
    "Effect": "Deny",  
    "Action": "",  
    "Resource": "",
```

```
    "Condition": {  
        "NotIpAddressIfExists": { "aws:SourceIp": "IP-address" },  
        "BoolIfExists": { "aws:ViaAWSService": "false" }  
    }  
}
```

Who can create a presigned URL

Anyone with valid security credentials can create a presigned URL. But for someone to successfully access an object, the presigned URL must be created by someone who has permission to perform the operation that the presigned URL is based upon.

The following are credentials that you can use to create a presigned URL:

- IAM instance profile: Valid up to 6 hours.
- AWS Security Token Service: Valid up to 36 hours when signed with permanent credentials, such as the credentials of the AWS account root user or an IAM user.
- IAM user: Valid up to 7 days when using AWS Signature Version 4.

To create a presigned URL that's valid for up to 7 days, first designate IAM user credentials (the access key and secret key) to the SDK that you're using. Then, generate a presigned URL using AWS Signature Version 4.

Note

- If you created a presigned URL using a temporary token, the URL expires when the token expires, even if the URL was created with a later expiration time.
- Because presigned URLs grant access to your Amazon S3 buckets to whoever has the URL, we recommend that you protect them appropriately.

When does Amazon S3 check the expiration date and time of a presigned URL?

Amazon S3 checks the expiration date and time of a signed URL at the time of the HTTP request. For example, if a client begins to download a large file immediately before the expiration time, the download should complete even if the expiration time passes during the download. If the connection drops and the client tries to restart the download after the expiration time passes, the download will fail.

For more information about using a presigned URL to share or upload objects, see the following topics.

Topics

- [Sharing objects using presigned URLs \(p. 258\)](#)
- [Generating a presigned URL to upload an object \(p. 262\)](#)
- [Deleting an object using a presigned URL with the AWS SDK for JavaScript \(p. 270\)](#)

Sharing objects using presigned URLs

By default, all S3 objects are private. Only the object owner has permission to access them. However, the object owner can optionally share objects with others by creating a presigned URL, using their own security credentials, to grant time-limited permission to download the objects.

When you create a presigned URL for your object, you must provide your security credentials and then specify a bucket name, an object key, an HTTP method (GET to download the object), and an expiration

date and time. The presigned URLs are valid only for the specified duration. If you created a presigned URL using a temporary token, then the URL expires when the token expires, even if the URL was created with a later expiration time.

Anyone who receives the presigned URL can then access the object. For example, if you have a video in your bucket and both the bucket and the object are private, you can share the video with others by generating a presigned URL. Because presigned URLs grant access to your Amazon S3 buckets to whoever has the URL, we recommend that you protect them appropriately. For more details about protecting presigned URLs, see [Limiting presigned URL capabilities \(p. 257\)](#).

For more information about who can create a presigned URL, see [Who can create a presigned URL \(p. 258\)](#).

Generating a presigned URL to share an object

You can generate a presigned URL for an object without writing any code by using the S3 console or AWS Explorer for Visual Studio. You can also generate a presigned URL programmatically using the AWS SDKs for Java, .NET, [Ruby](#), [PHP](#), [Node.js](#), [Python](#), and [Go](#).

Using the S3 console

You can use the AWS Management Console to generate a presigned URL for an object by following these steps.

Note

In the Amazon S3 console, the maximum expiration time for a presigned URL is 12 hours from the time of creation.

To generate a presigned URL using the AWS Management Console

1. Sign in to the AWS Management Console and open the Amazon S3 console at <https://console.aws.amazon.com/s3/>.
2. In the **Buckets** list, choose the name of the bucket that contains the object that you want a presigned URL for.
3. In the **Objects** list, select the object that you want to create a presigned URL for.
4. On the **Actions** menu, choose **Share with a presigned URL**.
5. Specify how long you want the presigned URL to be valid.
6. Choose **Create presigned URL**.
7. When a confirmation appears, the URL is automatically copied to your clipboard. You will see a button to copy the presigned URL if you need to copy it again.

Using AWS Explorer for Visual Studio

If you are using Visual Studio, you can generate a presigned URL for an object without writing any code by using AWS Explorer for Visual Studio. Anyone with this URL can download the object. For more information, go to [Using Amazon S3 from AWS Explorer](#).

Note

With AWS Explorer for Visual Studio, the maximum expiration time for a presigned URL is 7 days from the time of creation.

For instructions on how to install the AWS Explorer, see [Developing with Amazon S3 using the AWS SDKs, and explorers \(p. 1184\)](#).

Using the AWS SDKs

The following examples generate a presigned URL that you can give to others so that they can retrieve an object.

Note

With the AWS SDKs, the maximum expiration time for a presigned URL is 7 days from the time of creation.

Java

Example

The following example generates a presigned URL that you can give to others so that they can retrieve an object from an S3 bucket. For more information, see [Sharing objects using presigned URLs \(p. 258\)](#).

For instructions on creating and testing a working sample, see [Testing the Amazon S3 Java Code Examples \(p. 1192\)](#).

```
import com.amazonaws.AmazonServiceException;
import com.amazonaws.HttpMethod;
import com.amazonaws.SdkClientException;
import com.amazonaws.auth.profile.ProfileCredentialsProvider;
import com.amazonaws.regions.Regions;
import com.amazonaws.services.s3.AmazonS3;
import com.amazonaws.services.s3.AmazonS3ClientBuilder;
import com.amazonaws.services.s3.model.GeneratePresignedUrlRequest;

import java.io.IOException;
import java.net.URL;
import java.time.Instant;

public class GeneratePresignedURL {

    public static void main(String[] args) throws IOException {
        Regions clientRegion = Regions.DEFAULT_REGION;
        String bucketName = "*** Bucket name ***";
        String objectKey = "*** Object key ***";

        try {
            AmazonS3 s3Client = AmazonS3ClientBuilder.standard()
                .withRegion(clientRegion)
                .withCredentials(new ProfileCredentialsProvider())
                .build();

            // Set the presigned URL to expire after one hour.
            java.util.Date expiration = new java.util.Date();
            long expTimeMillis = Instant.now().toEpochMilli();
            expTimeMillis += 1000 * 60 * 60;
            expiration.setTime(expTimeMillis);

            // Generate the presigned URL.
            System.out.println("Generating pre-signed URL.");
            GeneratePresignedUrlRequest generatePresignedUrlRequest =
                new GeneratePresignedUrlRequest(bucketName, objectKey)
                    .withMethod(HttpMethod.GET)
                    .withExpiration(expiration);
            URL url = s3Client.generatePresignedUrl(generatePresignedUrlRequest);

            System.out.println("Pre-Signed URL: " + url.toString());
        } catch (AmazonServiceException e) {
            // The call was transmitted successfully, but Amazon S3 couldn't process
            // it, so it returned an error response.
            e.printStackTrace();
        } catch (SdkClientException e) {
            // Amazon S3 couldn't be contacted for a response, or the client
            // couldn't parse the response from Amazon S3.
        }
    }
}
```

```
        e.printStackTrace();
    }
}
```

.NET

Example

For instructions on creating and testing a working sample, see [Running the Amazon S3 .NET Code Examples \(p. 1193\)](#).

```
using Amazon;
using Amazon.S3;
using Amazon.S3.Model;
using System;

namespace Amazon.DocSamples.S3
{
    class GenPresignedURLTest
    {
        private const string bucketName = "*** bucket name ***";
        private const string objectKey = "*** object key ***";
        // Specify how long the presigned URL lasts, in hours
        private const double timeoutDuration = 12;
        // Specify your bucket region (an example region is shown).
        private static readonly RegionEndpoint bucketRegion = RegionEndpoint.USWest2;
        private static IAmazonS3 s3Client;

        public static void Main()
        {
            s3Client = new AmazonS3Client(bucketRegion);
            string urlString = GeneratePreSignedURL(timeoutDuration);
        }
        static string GeneratePreSignedURL(double duration)
        {
            string urlString = "";
            try
            {
                GetPreSignedUrlRequest request1 = new GetPreSignedUrlRequest
                {
                    BucketName = bucketName,
                    Key = objectKey,
                    Expires = DateTime.UtcNow.AddHours(duration)
                };
                urlString = s3Client.GetPreSignedURL(request1);
            }
            catch (AmazonS3Exception e)
            {
                Console.WriteLine("Error encountered on server. Message:'{0}' when
writing an object", e.Message);
            }
            catch (Exception e)
            {
                Console.WriteLine("Unknown encountered on server. Message:'{0}' when
writing an object", e.Message);
            }
            return urlString;
        }
    }
}
```

PHP

For information about using AWS SDK for PHP Version 3 to generate a presigned URL, see [Amazon S3 pre-signed URL with AWS SDK for PHP Version 3](#) in the *AWS SDK for PHP Developer Guide*.

Python

Generate a presigned URL to share an object by using the SDK for Python (Boto3). For example, use a Boto3 client and the `generate_presigned_url` function to generate a presigned URL that GETs an object.

```
import boto3
url = boto3.client('s3').generate_presigned_url(
    ClientMethod='get_object',
    Params={'Bucket': 'BUCKET_NAME', 'Key': 'OBJECT_KEY'},
    ExpiresIn=3600)
```

For more information about using SDK for Python (Boto3) to generate a presigned URL, see [Python](#) in the *AWS SDK for Python (Boto) API Reference*.

Using the AWS CLI

To generate a presigned URL to share an object using the AWS CLI, see [presign](#) in the *AWS CLI Command Reference*.

Note

With the AWS CLI, the maximum expiration time for a presigned URL is 7 days from the time of creation.

Generating a presigned URL to upload an object

A presigned URL gives you access to the object identified in the URL, provided that the creator of the presigned URL has permissions to access that object. That is, if you receive a presigned URL to upload an object, you can upload the object only if the creator of the presigned URL has the necessary permissions to upload that object.

All objects and buckets by default are private. The presigned URLs are useful if you want your user/customer to be able to upload a specific object to your bucket, but you don't require them to have AWS security credentials or permissions.

When you create a presigned URL, you must provide your security credentials and then specify a bucket name, an object key, an HTTP method (PUT for uploading objects), and an expiration date and time. The presigned URLs are valid only for the specified duration. That is, you must start the action before the expiration date and time.

If the action consists of multiple steps, such as a multipart upload, all steps must be started before the expiration. Otherwise, you will receive an error when Amazon S3 tries to start a step with an expired URL.

You can use the presigned URL multiple times, up to the expiration date and time.

Access to presigned URLs

Because presigned URLs grant access to your Amazon S3 buckets to whoever has the URL, we recommend that you protect them appropriately. For more information about protecting presigned URLs, see [Limiting presigned URL capabilities \(p. 257\)](#).

Anyone with valid security credentials can create a presigned URL. However, for you to successfully upload an object, the presigned URL must be created by someone who has permission to perform the operation that the presigned URL is based upon. For more information, see [Who can create a presigned URL \(p. 258\)](#).

Generating a presigned URL for uploading objects

You can generate a presigned URL programmatically using the AWS SDKs for .NET, Java, Ruby, JavaScript, PHP, and Python.

You can use the AWS SDK to generate a presigned URL that you or anyone that you give the URL to can use to upload an object to Amazon S3. When you use the URL to upload an object, Amazon S3 creates the object in the specified bucket. If an object with the same key that is specified in the presigned URL already exists in the bucket, Amazon S3 replaces the existing object with the uploaded object.

Using AWS Explorer for Visual Studio

If you're using Microsoft Visual Studio, you can generate a presigned object URL without writing any code by using AWS Explorer in the AWS Toolkit for Visual Studio. Anyone who receives a valid presigned URL can then programmatically upload an object. For more information, see [Using Amazon S3 from AWS Explorer](#) in the *AWS Toolkit for Visual Studio User Guide*.

For instructions on installing AWS Explorer, see [Developing with Amazon S3 using the AWS SDKs, and explorers \(p. 1184\)](#).

Using the AWS SDKs

The following examples show how to upload objects using presigned URLs.

The following code examples show how to create a presigned URL for S3 and upload an object.

Go

SDK for Go V2

Tip

To learn how to set up and run this example, see [GitHub](#).

```
// Get a presigned URL for the object.  
// In order to get a presigned URL for an object, you must  
// create a Presignclient  
fmt.Println("Create Presign client")  
presignClient := s3.NewPresignClient(&client)  
  
presignResult, err := presignClient.PresignGetObject(context.TODO(),  
&s3.GetObjectInput{  
    Bucket: aws.String(name),  
    Key:    aws.String("path/myfile.jpg"),  
})  
  
if err != nil {  
    panic("Couldn't get presigned URL for GetObject")  
}  
  
fmt.Printf("Presigned URL For object: %s\n", presignResult.URL)
```

Java

SDK for Java 2.x

Tip

To learn how to set up and run this example, see [GitHub](#).

```
public static void signBucket(S3Presigner presigner, String bucketName, String keyName) {

    try {
        PutObjectRequest objectRequest = PutObjectRequest.builder()
            .bucket(bucketName)
            .key(keyName)
            .contentType("text/plain")
            .build();

        PutObjectPresignRequest presignRequest =
PutObjectPresignRequest.builder()
            .signatureDuration(Duration.ofMinutes(10))
            .putObjectRequest(objectRequest)
            .build();

        PresignedPutObjectRequest presignedRequest =
presigner.presignPutObject(presignRequest);
        String myURL = presignedRequest.url().toString();
        System.out.println("Presigned URL to upload a file to: " +myURL);
        System.out.println("Which HTTP method needs to be used when uploading a
file: " + presignedRequest.httpRequest().method());

        // Upload content to the Amazon S3 bucket by using this URL.
        URL url = presignedRequest.url();

        // Create the connection and use it to upload the new object by using
the presigned URL.
        HttpURLConnection connection = (HttpURLConnection)
url.openConnection();
        connection.setDoOutput(true);
        connection.setRequestProperty("Content-Type", "text/plain");
        connection.setRequestMethod("PUT");
        OutputStreamWriter out = new
OutputStreamWriter(connection.getOutputStream());
        out.write("This text was uploaded as an object by using a presigned
URL.");
        out.close();

        connection.getResponseCode();
        System.out.println("HTTP response code is " +
connection.getResponseCode());

    } catch (S3Exception | IOException e) {
        e.printStackTrace();
    }
}
```

JavaScript

SDK for JavaScript V3

Tip

To learn how to set up and run this example, see [GitHub](#).

Create the client.

```
// Create service client module using ES6 syntax.
import { S3Client } from "@aws-sdk/client-s3";
// Set the AWS Region.
const REGION = "us-east-1";
```

```
// Create an Amazon S3 service client object.  
const s3Client = new S3Client({ region: REGION });  
export { s3Client };
```

Create a presigned URL to upload an object to a bucket.

```
// Import the required AWS SDK clients and commands for Node.js  
import {  
    CreateBucketCommand,  
    DeleteObjectCommand,  
    PutObjectCommand,  
    DeleteBucketCommand }  
from "@aws-sdk/client-s3";  
import { s3Client } from "./libs/s3Client.js"; // Helper function that creates an  
// Amazon S3 service client module.  
import { getSignedUrl } from "@aws-sdk/s3-request-presigner";  
import fetch from "node-fetch";  
  
// Set parameters  
// Create a random name for the Amazon Simple Storage Service (Amazon S3) bucket  
and key  
export const bucketParams = {  
    Bucket: `test-bucket-${Math.ceil(Math.random() * 10 ** 10)}`,  
    Key: `test-object-${Math.ceil(Math.random() * 10 ** 10)}`,  
    Body: "BODY"  
};  
export const run = async () => {  
    try {  
        // Create an S3 bucket.  
        console.log(`Creating bucket ${bucketParams.Bucket}`);  
        await s3Client.send(new CreateBucketCommand({ Bucket: bucketParams.Bucket }));  
        console.log(`Waiting for "${bucketParams.Bucket}" bucket creation...`);  
    } catch (err) {  
        console.log("Error creating bucket", err);  
    }  
    try {  
        // Create a command to put the object in the S3 bucket.  
        const command = new PutObjectCommand(bucketParams);  
        // Create the presigned URL.  
        const signedUrl = await getSignedUrl(s3Client, command, {  
            expiresIn: 3600,  
        });  
        console.log(  
            `\nPutting "${bucketParams.Key}" using signedUrl with body  
"${bucketParams.Body}" in v3`  
        );  
        console.log(signedUrl);  
        const response = await fetch(signedUrl, {method: 'PUT', body:  
        bucketParams.Body});  
        console.log(  
            `\nResponse returned by signed URL: ${await response.text()}\n`  
        );  
    } catch (err) {  
        console.log("Error creating presigned URL", err);  
    }  
    try {  
        // Delete the object.  
        console.log(`\nDeleting object "${bucketParams.Key}" from bucket`);  
        await s3Client.send(  
            new DeleteObjectCommand({ Bucket: bucketParams.Bucket, Key:  
            bucketParams.Key })  
        );  
    } catch (err) {  
        console.log("Error deleting object", err);  
    }  
};
```

```
        }
    try {
        // Delete the S3 bucket.
        console.log(`\nDeleting bucket ${bucketParams.Bucket}`);
        await s3Client.send(
            new DeleteBucketCommand({ Bucket: bucketParams.Bucket })
        );
    } catch (err) {
        console.log("Error deleting bucket", err);
    }
};

run();
```

Create a presigned URL to download an object from a bucket.

```
// Import the required AWS SDK clients and commands for Node.js
import {
    CreateBucketCommand,
    PutObjectCommand,
    GetObjectCommand,
    DeleteObjectCommand,
    DeleteBucketCommand
} from "@aws-sdk/client-s3";
import { s3Client } from "./libs/s3Client.js"; // Helper function that creates an
// Amazon S3 service client module.
import { getSignedUrl } from "@aws-sdk/s3-request-presigner";
const fetch = require("node-fetch");

// Set parameters
// Create a random names for the S3 bucket and key.
export const bucketParams = {
    Bucket: `test-bucket-${Math.ceil(Math.random() * 10 ** 10)}`,
    Key: `test-object-${Math.ceil(Math.random() * 10 ** 10)}`,
    Body: "BODY"
};

export const run = async () => {
    // Create an S3 bucket.
    try {
        console.log(`Creating bucket ${bucketParams.Bucket}`);
        const data = await s3Client.send(
            new CreateBucketCommand({ Bucket: bucketParams.Bucket })
        );
        return data; // For unit tests.
        console.log(`Waiting for "${bucketParams.Bucket}" bucket creation...\n`);
    } catch (err) {
        console.log("Error creating bucket", err);
    }
    // Put the object in the S3 bucket.
    try {
        console.log(`Putting object "${bucketParams.Key}" in bucket`);
        const data = await s3Client.send(
            new PutObjectCommand({
                Bucket: bucketParams.Bucket,
                Key: bucketParams.Key,
                Body: bucketParams.Body,
            })
        );
        return data; // For unit tests.
    } catch (err) {
        console.log("Error putting object", err);
    }
    // Create a presigned URL.
    try {
```

```
// Create the command.
const command = new GetObjectCommand(bucketParams);

// Create the presigned URL.
const signedUrl = await getSignedUrl(s3Client, command, {
    expiresIn: 3600,
});
console.log(
    `\nGetting "${bucketParams.Key}" using signedUrl with body
"${bucketParams.Body}" in v3`
);
console.log(signedUrl);
const response = await fetch(signedUrl);
console.log(
    `\nResponse returned by signed URL: ${await response.text()}\n`
);
} catch (err) {
    console.log("Error creating presigned URL", err);
}
// Delete the object.
try {
    console.log(`\nDeleting object "${bucketParams.Key}" from bucket`);
    const data = await s3Client.send(
        new DeleteObjectCommand({ Bucket: bucketParams.Bucket, Key:
    bucketParams.Key })
    );
    return data; // For unit tests.
} catch (err) {
    console.log("Error deleting object", err);
}
// Delete the S3 bucket.
try {
    console.log(`\nDeleting bucket ${bucketParams.Bucket}`);
    const data = await s3Client.send(
        new DeleteBucketCommand({ Bucket: bucketParams.Bucket, Key:
    bucketParams.Key })
    );
    return data; // For unit tests.
} catch (err) {
    console.log("Error deleting object", err);
}
};

run();
```

- For more information, see [AWS SDK for JavaScript Developer Guide](#).

Python

SDK for Python (Boto3)

Tip

To learn how to set up and run this example, see [GitHub](#).

Generate a presigned URL that can perform an S3 action for a limited time. Use the Requests package to make a request with the URL.

```
import argparse
import logging
import boto3
from botocore.exceptions import ClientError
import requests

logger = logging.getLogger(__name__)
```

```
def generate_presigned_url(s3_client, client_method, method_parameters,
                           expires_in):
    """
    Generate a presigned Amazon S3 URL that can be used to perform an action.

    :param s3_client: A Boto3 Amazon S3 client.
    :param client_method: The name of the client method that the URL performs.
    :param method_parameters: The parameters of the specified client method.
    :param expires_in: The number of seconds the presigned URL is valid for.
    :return: The presigned URL.
    """
    try:
        url = s3_client.generate_presigned_url(
            ClientMethod=client_method,
            Params=method_parameters,
            ExpiresIn=expires_in
        )
        logger.info("Got presigned URL: %s", url)
    except ClientError:
        logger.exception(
            "Couldn't get a presigned URL for client method '%s'.", client_method)
        raise
    return url

def usage_demo():
    logging.basicConfig(level=logging.INFO, format='%(levelname)s: %(message)s')

    print('*'*88)
    print("Welcome to the Amazon S3 presigned URL demo.")
    print('*'*88)

    parser = argparse.ArgumentParser()
    parser.add_argument('bucket', help="The name of the bucket.")
    parser.add_argument(
        'key', help="For a GET operation, the key of the object in Amazon S3. For a "
    "PUT operation, the name of a file to upload.")
    parser.add_argument(
        'action', choices=('get', 'put'), help="The action to perform.")
    args = parser.parse_args()

    s3_client = boto3.client('s3')
    client_action = 'get_object' if args.action == 'get' else 'put_object'
    url = generate_presigned_url(
        s3_client, client_action, {'Bucket': args.bucket, 'Key': args.key}, 1000)

    print("Using the Requests package to send a request to the URL.")
    response = None
    if args.action == 'get':
        response = requests.get(url)
    elif args.action == 'put':
        print("Putting data to the URL.")
        try:
            with open(args.key, 'r') as object_file:
                object_text = object_file.read()
            response = requests.put(url, data=object_text)
        except FileNotFoundError:
            print(f"Couldn't find {args.key}. For a PUT operation, the key must be "
the "
f"name of a file that exists on your computer.")

    if response is not None:
        print("Got response:")
```

```
print(f"Status: {response.status_code}")
print(response.text)

print('*'*88)

if __name__ == '__main__':
    usage_demo()
```

Generate a presigned POST request to upload a file.

```
class BucketWrapper:
    def __init__(self, bucket):
        self.bucket = bucket
        self.name = bucket.name

    def generate_presigned_post(self, object_key, expires_in):
        """
        Generate a presigned Amazon S3 POST request to upload a file.
        A presigned POST can be used for a limited time to let someone without an
        AWS
        account upload a file to a bucket.

        :param object_key: The object key to identify the uploaded object.
        :param expires_in: The number of seconds the presigned POST is valid.
        :return: A dictionary that contains the URL and form fields that contain
                required access data.
        """
        try:
            response = self.bucket.meta.client.generate_presigned_post(
                Bucket=self.bucket.name, Key=object_key, ExpiresIn=expires_in)
            logger.info("Got presigned POST URL: %s", response['url'])
        except ClientError:
            logger.exception(
                "Couldn't get a presigned POST URL for bucket '%s' and object
                '%s'", self.bucket.name, object_key)
            raise
        return response
```

Ruby

SDK for Ruby

Tip

To learn how to set up and run this example, see [GitHub](#).

```
require "aws-sdk-s3"
require "net/http"

# Creates a presigned URL that can be used to upload content to an object.
#
# @param bucket [Aws::S3::Bucket] An existing Amazon S3 bucket.
# @param object_key [String] The key to give the uploaded object.
# @return [URI, nil] The parsed URI if successful; otherwise nil.
def get_presigned_url(bucket, object_key)
    url = bucket.object(object_key).presigned_url(:put)
    puts "Created presigned URL: #{url}."
    URI(url)
```

```
rescue Aws::Errors::ServiceError => e
  puts "Couldn't create presigned URL for #{bucket.name}:#{object_key}. Here's why:
  #{e.message}"
end

def run_demo
  bucket_name = "doc-example-bucket"
  object_key = "my-file.txt"
  object_content = "This is the content of my-file.txt."

  bucket = Aws::S3::Bucket.new(bucket_name)
  presigned_url = get_presigned_url(bucket, object_key)
  return unless presigned_url

  response = Net::HTTP.start(presigned_url.host) do |http|
    http.send_request("PUT", presigned_url.request_uri, object_content,
"content_type" => "")
  end

  case response
  when Net::HTTPSuccess
    puts "Content uploaded!"
  else
    puts response.value
  end
end

run_demo if $PROGRAM_NAME == __FILE__
```

Deleting an object using a presigned URL with the AWS SDK for JavaScript

Example

The following AWS SDK for JavaScript example uses a presigned URL to delete an object.

For more information about using presigned URLs, see [Using presigned URLs \(p. 257\)](#).

```
// Import the required AWS SDK clients and commands for Node.js
import {
  CreateBucketCommand,
  DeleteObjectCommand,
  PutObjectCommand,
  DeleteBucketCommand
} from "@aws-sdk/client-s3";
import { s3Client } from "./libs/s3Client.js"; // Helper function that creates an Amazon S3
service client module.
import { getSignedUrl } from "@aws-sdk/s3-request-presigner";
import fetch from "node-fetch";

// Set parameters
// Create a random name for the Amazon Simple Storage Service (Amazon S3) bucket and key
export const bucketParams = {
  Bucket: `test-bucket-${Math.ceil(Math.random() * 10 ** 10)}`,
  Key: `test-object-${Math.ceil(Math.random() * 10 ** 10)}`,
  Body: "BODY"
};
export const run = async () => {
  try {
```

```

// Create an S3 bucket.
console.log(`Creating bucket ${bucketParams.Bucket}`);
await s3Client.send(new CreateBucketCommand({ Bucket: bucketParams.Bucket }));
console.log(`Waiting for "${bucketParams.Bucket}" bucket creation...`);
} catch (err) {
  console.log("Error creating bucket", err);
}
try {
  // Create a command to put the object in the S3 bucket.
  const command = new PutObjectCommand(bucketParams);
  // Create the presigned URL.
  const signedUrl = await getSignedUrl(s3Client, command, {
    expiresIn: 3600,
  });
  console.log(
    `\nPutting "${${bucketParams.Key}}" using signedUrl with body "${${bucketParams.Body}}" in
v3`;
  );
  console.log(signedUrl);
  const response = await fetch(signedUrl, {method: 'PUT', body: bucketParams.Body});
  console.log(
    `\nResponse returned by signed URL: ${await response.text()}\n`;
  );
} catch (err) {
  console.log("Error creating presigned URL", err);
}
try {
  // Delete the object.
  console.log(`\nDeleting object "${${bucketParams.Key}}" from bucket`);
  await s3Client.send(
    new DeleteObjectCommand({ Bucket: bucketParams.Bucket, Key: bucketParams.Key })
  );
} catch (err) {
  console.log("Error deleting object", err);
}
try {
  // Delete the S3 bucket.
  console.log(`\nDeleting bucket ${${bucketParams.Bucket}}`);
  await s3Client.send(
    new DeleteBucketCommand({ Bucket: bucketParams.Bucket })
  );
} catch (err) {
  console.log("Error deleting bucket", err);
}
};

run();

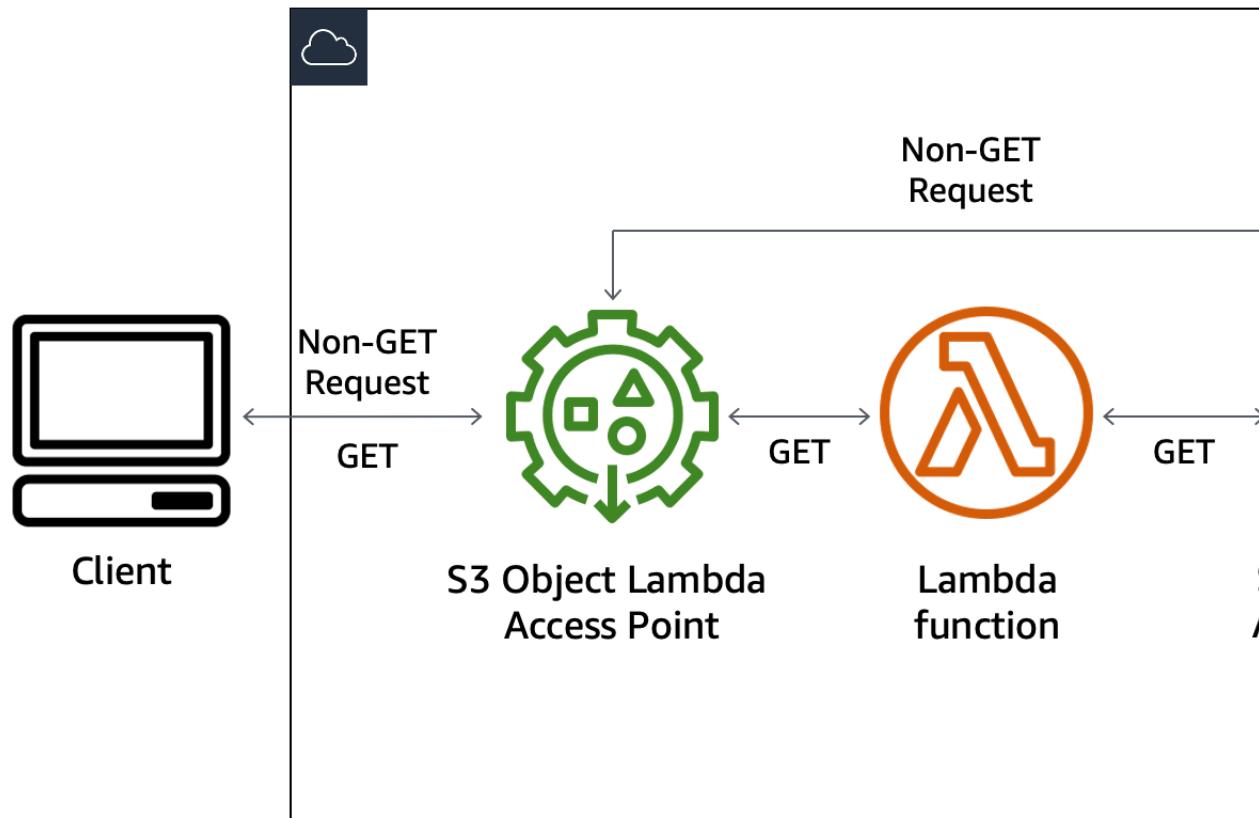
```

Transforming objects with S3 Object Lambda

With S3 Object Lambda you can add your own code to Amazon S3 GET requests to modify and process data as it is returned to an application. You can use custom code to modify the data returned by standard S3 GET requests to filter rows, dynamically resize images, redact confidential data, and more. Powered by AWS Lambda functions, your code runs on infrastructure that is fully managed by AWS, eliminating the need to create and store derivative copies of your data or to run proxies, all with no changes required to applications.

S3 Object Lambda uses AWS Lambda functions to automatically process the output of a standard S3 GET request. AWS Lambda is a serverless compute service that runs customer-defined code without requiring management of underlying compute resources. You can author and execute your own custom Lambda

functions, tailoring data transformation to your specific use cases. You can configure a Lambda function and attach it to an S3 Object Lambda service endpoint and S3 will automatically call your function. Then any data retrieved using an S3 GET request through the S3 Object Lambda endpoint will return a transformed result back to the application. All other requests will be processed as normal, as illustrated in the following diagram.



The topics in this section describe how to work with Object Lambda access points.

Topics

- [Creating Object Lambda Access Points \(p. 273\)](#)
- [Using Amazon S3 Object Lambda Access Points \(p. 277\)](#)
- [Using an AWS CloudFormation template to automate S3 Object Lambda setup \(p. 278\)](#)
- [Configuring IAM policies for Object Lambda access points \(p. 281\)](#)
- [Writing and debugging AWS Lambda functions for Amazon S3 Object Lambda Access Points \(p. 284\)](#)
- [Using AWS built Lambda functions \(p. 296\)](#)

- [Best practices and guidelines for S3 Object Lambda \(p. 298\)](#)
- [Security considerations for S3 Object Lambda access points \(p. 299\)](#)
- [S3 Object Lambda tutorials \(p. 300\)](#)

Creating Object Lambda Access Points

An Object Lambda access point is associated with exactly one standard access point and thus one Amazon S3 bucket. To create an Object Lambda access point, you need the following resources:

- An IAM policy
- An Amazon S3 bucket
- A standard S3 access point
- An AWS Lambda function

The following sections describe how to create an Object Lambda access point using the AWS Management Console and AWS CLI.

Create an Object Lambda access point

For information about how to create an Object Lambda access point using the REST API, see [CreateAccessPointForObjectLambda](#) in the *Amazon Simple Storage Service API Reference*.

Using the S3 console

To create an Object Lambda access point using the console

1. Sign in to the AWS Management Console and open the Amazon S3 console at <https://console.aws.amazon.com/s3/>.
2. In the navigation pane on the left side of the console, choose **Object Lambda access points**.
3. On the **Object Lambda access points** page, choose **Create Object Lambda access point**.
4. For **Object Lambda access point name**, enter the name you want to use for the access point.

As with standard access points, there are rules for naming. For more information, see [Rules for naming Amazon S3 access points \(p. 306\)](#).

5. For **Supporting access point**, enter or browse to the standard access point that you want to use. The access point must be in the same AWS Region as the objects you want to transform.
6. For **Invoke Lambda function**, you can choose to use a prebuilt function or enter the Amazon Resource Name (ARN) of an AWS Lambda function in your AWS account.

For more information about prebuilt functions, see [Using AWS built Lambda functions \(p. 296\)](#).

7. (Optional) For **Range and part number**, you must enable this option in order to process GET requests with range and part number headers. Selecting this option confirms that your Lambda function is able to recognize and process these requests. For more information about range headers and part numbers, see [Working with Range and partNumber headers \(p. 293\)](#).
8. (Optional) Under **Payload**, add JSON text to provide your Lambda function with additional information. A payload is optional JSON that you can provide to your Lambda function as input. You can configure payloads with different parameters for different Object Lambda access points that invoke the same Lambda function, thereby extending the flexibility of your Lambda function.
9. (Optional) For **Request metrics**, choose **enable** or **disable** to add Amazon S3 monitoring to your Object Lambda access point. Request metrics are billed at the standard CloudWatch rate.

10. (Optional) Under **Object Lambda access point policy**, set a resource policy. This resource policy grants GetObject permission for the specified Object Lambda access point.
11. Choose **Create Object Lambda access point**.

Using the AWS CLI

To create an Object Lambda access point using an AWS CloudFormation template

1. Download the AWS Lambda function deployment package `s3objectlambda_deployment_package.zip` at [S3 Object Lambda default configuration](#).
2. Upload the package to an Amazon S3 bucket that has object versioning enabled.

```
aws s3api put-object --bucket Amazon S3 bucket name --key  
s3objectlambda_deployment_package.zip --body release/  
s3objectlambda_deployment_package.zip
```

3. Download the AWS CloudFormation template `s3objectlambda_defaultconfig.yaml` at [S3 Object Lambda default configuration](#).
4. Deploy the template to your AWS account.

```
aws cloudformation deploy --template-file s3objectlambda_defaultconfig.yaml \  
--stack-name AWS CloudFormation stack name --parameter-overrides  
ObjectLambdaAccessPointName=Object Lambda access point name \  
SupportingAccessPointName=Amazon S3 access point S3BucketName=Amazon S3 bucket \  
LambdaFunctionS3BucketName=Amazon S3 bucket containing your Lambda package \  
LambdaFunctionsS3Key=Lambda object key LambdaFunctionS3ObjectVersion=Lambda object \  
version LambdaFunctionRuntime=Lambda function runtime --capabilities capability_IAM
```

For more information about modifying the AWS CloudFormation template's default configuration, see [the section called "Getting started with an AWS CloudFormation template" \(p. 278\)](#).

To create an Object Lambda access point using the AWS CLI

The following example creates an Object Lambda access point named `my-object-lambda-ap` for the bucket `DOC-EXAMPLE-BUCKET1` in the account `111122223333`. This example assumes that a standard access point named `example-ap` has already been created. For information about creating a standard access point, see [the section called "Creating access points" \(p. 306\)](#).

This example uses the AWS prebuilt function `compress`. For example AWS Lambda functions, see [the section called "Using AWS built functions" \(p. 296\)](#).

1. Create a bucket. In this example, we will use `DOC-EXAMPLE-BUCKET1`. For information about creating buckets, see [the section called "Creating a bucket" \(p. 119\)](#).
2. Create a standard access point and attach it to your bucket. In this example, we will use `example-ap`. For information about creating standard access points, see [the section called "Creating access points" \(p. 306\)](#).
3. Create a Lambda function in your account that you would like to use to transform your Amazon S3 object. See [Using Lambda with the AWS CLI](#) in the *AWS Lambda Developer Guide*. You can also use an AWS prebuilt Lambda function.
4. Create a JSON configuration file named `my-olap-configuration.json`. In this configuration, provide the supporting access point and Lambda function ARN created in the previous steps.

Example

```
{
```

```

        "SupportingAccessPoint" : "arn:aws:s3:us-east-1:111122223333:accesspoint/example-
ap",
        "TransformationConfigurations": [
            {
                "Actions" : ["GetObject"],
                "ContentTransformation" : {
                    "AwsLambda": {
                        "FunctionPayload" : "{\"compressionType\":\"gzip\"}",
                        "FunctionArn" : "arn:aws:lambda:us-east-1:111122223333:function/
compress"
                    }
                }
            }
        ]
    }
}

```

- Run `create-access-point-for-object-lambda` to create your Object Lambda access point.

```
aws s3control create-access-point-for-object-lambda --account-id 111122223333 --name my-object-lambda-ap --configuration file://my-olap-configuration.json
```

- (Optional) Create a JSON policy file named `my-olap-policy.json`.

This resource policy grants `GetObject` permission for account `444455556666` to the specified Object Lambda access point.

Example

```
{
    "Version": "2008-10-17",
    "Statement": [
        {
            "Sid": "Grant account 444455556666 GetObject access",
            "Effect": "Allow",
            "Action": "s3-object-lambda:GetObject",
            "Principal": {
                "AWS": "arn:aws:iam::444455556666:root"
            },
            "Resource": "your-object-lambda-access-point-arn"
        }
    ]
}
```

- (Optional) Run `put-access-point-policy-for-object-lambda` to set your resource policy.

```
aws s3control put-access-point-policy-for-object-lambda --account-id 111122223333 --name my-object-lambda-ap --policy file://my-olap-policy.json
```

- (Optional) Specify a payload.

A payload is optional JSON that you can provide to your AWS Lambda function as input. You can configure payloads with different parameters for different Object Lambda access points that invoke the same Lambda function, thereby extending the flexibility of your Lambda function.

The following Object Lambda access point configuration shows a payload with two parameters.

```
{
    "SupportingAccessPoint": "AccessPointArn",
    "CloudWatchMetricsEnabled": false,
    "TransformationConfigurations": [
        {
            "Actions": ["GetObject"],
            "ContentTransformation": {
                "AwsLambda": {

```

```
        "FunctionArn": "FunctionArn",  
        "FunctionPayload": "{\"res-x\": \"100\", \"res-y\": \"100\"}"  
    }  
}  
]  
}
```

The following Object Lambda access point configuration shows a payload with one parameter, and with `GetObject-Range` and `GetObject-PartNumber` enabled.

```
{  
    "SupportingAccessPoint": "AccessPointArn",  
    "CloudWatchMetricsEnabled": false,  
    "AllowedFeatures": ["GetObject-Range", "GetObject-PartNumber"],  
    "TransformationConfigurations": [  
        {  
            "Actions": ["GetObject"],  
            "ContentTransformation": {  
                "AwsLambda": {  
                    "FunctionArn": "FunctionArn",  
                    "FunctionPayload": "{\"compression-amount\": \"5\"}"  
                }  
            }  
        }  
    ]  
}
```

Important

When you're using Object Lambda access points, make sure that the payload does not contain any confidential information.

Using the AWS CloudFormation console and template

You can create an Object Lambda access point using the default configuration provided by Amazon S3. You can download an AWS CloudFormation template and Lambda function source code from the [GitHub repository](#) and deploy these resources to set up a functional Object Lambda access point.

To upload the Lambda function deployment package

1. Download the AWS Lambda function deployment package `s3objectlambda_deployment_package.zip` at [S3 Object Lambda default configuration](#).
2. Upload the package to an Amazon S3 bucket that has object versioning enabled.

To create an Object Lambda access point using the AWS CloudFormation console

1. Download the AWS CloudFormation template `s3objectlambda_defaultconfig.yaml` at [S3 Object Lambda default configuration](#).
2. Sign in to the AWS Management Console and open the AWS CloudFormation console at <https://console.aws.amazon.com/cloudformation>.
3. On the AWS CloudFormation page, choose **Create stack**.
4. For **Prerequisite - Prepare template**, choose **Template is ready**.
5. For **Specify template**, choose **Upload a template file** and upload `s3objectlambda_defaultconfig.yaml`.
6. Choose **Next**.
7. On the **Specify stack details** page, enter a name for the stack.
8. Choose **Next**.
9. In the Parameters section, specify the following parameters that are defined in the stack template:

- a. Enter the Amazon S3 bucket name where you uploaded the deployment package for **LambdaFunctionS3BucketName**.
- b. Enter the Amazon S3 object key where you uploaded the deployment package for **LambdaFunctionS3Key**.
- c. Enter the Amazon S3 object version where you uploaded the deployment package for **LambdaFunctionS3ObjectVersion**.
- d. Enter your preferred runtime for the Lambda function for **LambdaFunctionRuntime**. The available choices are nodejs14.x, python3.9, java11.
- e. Enter a name for your Object Lambda access point for **ObjectLambdaAccessPointName**.
- f. Enter the Amazon S3 bucket name that will be associated with your Object Lambda access point for **S3BucketName**.
- g. Enter the name of your supporting access point for **SupportingAccessPointName**.

Note

This is an access point associated with the Amazon S3 bucket that you chose in the previous step. If you do not have any access points associated with your Amazon S3 bucket, you can configure the template to create one for you by selecting **true** for **CreateNewSupportingAccessPoint**.

10. Choose **Next**.

11. On the **Configure stack options** page, choose **Next**.

For more information on the optional settings on this page, see [Setting AWS CloudFormation stack options](#) in the *AWS CloudFormation User Guide*.

12. On the **Review** page, choose **Create stack**.

For more information about modifying the AWS CloudFormation template's default configuration, see [the section called "Getting started with an AWS CloudFormation template" \(p. 278\)](#).

For more information about configuring Object Lambda access points using AWS CloudFormation without the template, see [AWS::S3ObjectLambda::AccessPoint](#) in the *AWS CloudFormation User Guide*.

Using AWS Cloud Development Kit (AWS CDK)

For more information about configuring Object Lambda access points using the AWS CDK, see [AWS::S3ObjectLambda Construct Library](#) in the *AWS Cloud Development Kit (AWS CDK) API Reference*.

Using Amazon S3 Object Lambda Access Points

Making requests through S3 Object Lambda access points works the same as making requests through other access points. For more information about how to make requests through an access point, see [Using access points \(p. 310\)](#). You can make requests through Object Lambda access points by using the AWS Management Console, AWS Command Line Interface (AWS CLI), AWS SDKs, or Amazon S3 REST API.

Important

The Amazon Resource Names (ARNs) for Object Lambda access points use a service name of s3-object-lambda. Thus, Object Lambda access point ARNs begin with `arn:aws::s3-object-lambda`, instead of `arn:aws::s3`, which is used with other access points.

How to find the ARN for your Object Lambda access point

To use an Object Lambda access point with the AWS CLI or AWS SDKs, you need to know the Amazon Resource Name (ARN) of the Object Lambda access point. The following examples show how to find the ARN for an Object Lambda access point by using the AWS Management Console or AWS CLI.

AWS Management Console

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 **Object Lambda Access Points**.
3. Select the option button next to the Object Lambda access point whose ARN you want to copy.
4. Choose **Copy ARN**.

AWS CLI

1. To retrieve a list of the Object Lambda access points that are associated with your AWS account, run the following command. Before running the command, replace the account ID **111122223333** with your AWS account ID.

```
aws s3control list-access-points-for-object-lambda --account-id 111122223333
```

2. Review the command output to find the Object Lambda access point ARN that you want to use. The output of the previous command should look similar to the following example.

```
{  
    "ObjectLambdaAccessPointList": [  
        {  
            "Name": "my-object-lambda-ap",  
            "ObjectLambdaAccessPointArn": "arn:aws:s3-object-lambda:us-east-1:111122223333:accesspoint/my-object-lambda-ap"  
        },  
        ...  
    ]  
}
```

Using an AWS CloudFormation template to automate S3 Object Lambda setup

You can use an AWS CloudFormation template to quickly create an Object Lambda access point. The AWS CloudFormation template automatically creates relevant resources, configures IAM roles, and sets up a Lambda function that automatically handles requests through the Object Lambda access point. With the AWS CloudFormation template, you can implement best practices, improve your security posture, and reduce errors caused by manual processes.

This [GitHub repository](#) contains the AWS CloudFormation template and AWS Lambda function source code. For instructions on how to use the template, see [Creating Object Lambda Access Points](#).

The Lambda function provided in the template does not run any transformation, and returns your objects as-is from your Amazon S3 bucket. You can clone the function and add your own transformation code to modify and process data as it is returned to an application.

Modifying the template

Creating a new Supporting Access Point

You can create a new supporting Access Point by passing the following parameter as part of the `aws cloudformation deploy` command to create when deploying the template.

```
CreateNewSupportingAccessPoint=true
```

Configuring a function payload

You can configure a payload to provide supplemental data to the Lambda function by passing the following parameter as part of the `aws cloudformation deploy` command when deploying the template.

```
LambdaFunctionPayload="format=csv"
```

Enabling Amazon CloudWatch monitoring

You can enable CloudWatch monitoring by passing the following parameter as part of the `aws cloudformation deploy` command when deploying the template.

```
EnableCloudWatchMonitoring=true
```

This will enable your Object Lambda access point for Amazon S3 request metrics and create two CloudWatch alarms to monitor client-side and server-side errors.

Note

Amazon CloudWatch usage will incur additional costs. For more information on Amazon S3 request metrics, see [Monitoring and logging access points](#)

Configuring provisioned concurrency

To optimize latency, you can configure provisioned concurrency for the Lambda function backing the Object Lambda access point by editing the template to include the following lines under `Resources`.

```
LambdaFunctionVersion:  
  Type: AWS::Lambda::Version  
  Properties:  
    FunctionName: !Ref LambdaFunction  
    ProvisionedConcurrencyConfig:  
      ProvisionedConcurrentExecutions: Integer
```

Note

You will incur additional charges for provisioning concurrency. For more information, see [Managing Lambda provisioned concurrency](#) in the *AWS Lambda Developer Guide*.

Modifying the Lambda function

Changing header values

By default, the Lambda function forwards all headers, except Content-Length and ETag, from the pre-signed URL request to the `GetObject` client. Based on your transformation code in the Lambda function, you can choose to send new header values to the `GetObject` client.

You can update your Lambda function to send new header values by passing them in the `WriteGetObjectResponse` API.

For example, if your Lambda function translates text in Amazon S3 objects to a different language, you can pass a new value in the `Content-Language` header. You can do this by modifying the `writeResponse` function as below.

```
async function writeResponse (s3Client: S3, requestContext: GetObjectContext,  
  transformedObject: Buffer,  
  headers: Headers): Promise<PromiseResult<{}, AWSError>> {  
  const { algorithm, digest } = getChecksum(transformedObject);  
  
  return s3Client.writeGetObjectResponse({  
    RequestRoute: requestContext.outputRoute,
```

```
RequestToken: requestContext.outputToken,
Body: transformedObject,
Metadata: {
  'body-checksum-algorithm': algorithm,
  'body-checksum-digest': digest
},
...headers,
ContentLanguage: 'my-new-language'
}).promise();
}
```

For a full list of supported headers, see [WriteGetObjectResponse](#) in the *Amazon Simple Storage Service API Reference*.

Returning metadata headers

You can update your Lambda function to send new header values by passing them in the [WriteGetObjectResponse](#) API.

```
async function writeResponse (s3Client: S3, requestContext: GetObjectContext,
transformedObject: Buffer,
headers: Headers): Promise<PromiseResult<{}, AWSError>> {
const { algorithm, digest } = getChecksum(transformedObject);

return s3Client.writeGetObjectResponse({
  RequestRoute: requestContext.outputRoute,
  RequestToken: requestContext.outputToken,
  Body: transformedObject,
  Metadata: {
    'body-checksum-algorithm': algorithm,
    'body-checksum-digest': digest,
    'my-new-header': 'my-new-value'
  },
  ...headers
}).promise();
}
```

Returning a new status code

You can return a custom status code to the `GetObject` client by passing it in the [WriteGetObjectResponse](#) API.

```
async function writeResponse (s3Client: S3, requestContext: GetObjectContext,
transformedObject: Buffer,
headers: Headers): Promise<PromiseResult<{}, AWSError>> {
const { algorithm, digest } = getChecksum(transformedObject);

return s3Client.writeGetObjectResponse({
  RequestRoute: requestContext.outputRoute,
  RequestToken: requestContext.outputToken,
  Body: transformedObject,
  Metadata: {
    'body-checksum-algorithm': algorithm,
    'body-checksum-digest': digest
  },
  ...headers,
  StatusCode: Integer
}).promise();
}
```

For a full list of supported status codes, see [WriteGetObjectResponse](#) in the *Amazon Simple Storage Service API Reference*.

Applying Range and partNumber to the source object

By default, the Object Lambda access point created by the AWS CloudFormation template can handle range and part number parameters. The Lambda function applies the range or part number requested to the transformed object. To do so, it needs to download the whole object and run the transformation. In some cases, your transformed object ranges might map exactly to your source object ranges. This means that requesting byte range A-B on your source object and running the transformation may produce the same result as requesting for the whole object, running the transformation, and returning byte range A-B on the transformed object.

In such cases, you can change the Lambda function implementation to apply the range or part number directly to the source object. This improves the overall function latency and memory required. For more information, see [Working with Range and partNumber headers](#).

Disabling Range and partNumber handling

By default, the Object Lambda access point created by the AWS CloudFormation template can handle range and part number parameters. If you do not need this, you can disable it by removing the following lines from the template.

```
AllowedFeatures:  
  - GetObject-Range  
  - GetObject-PartNumber
```

Transforming large objects

By default, the Lambda function processes the entire object in memory before it can start streaming the response to S3 Object Lambda. You can modify the function to stream the response as it performs the transformation. This helps reduce the transformation latency and the Lambda function memory size. For an example implementation, see the [Stream compressed content example](#).

Configuring IAM policies for Object Lambda access points

Amazon S3 access points support AWS Identity and Access Management (IAM) resource policies that you can use to control the use of the access point by resource, user, or other conditions. For step-by-step examples, see [Tutorial: Transforming data for your application with S3 Object Lambda \(p. 27\)](#) and [Tutorial: Detecting and redacting PII data with S3 Object Lambda and Amazon Comprehend \(p. 41\)](#).

In the case of a single AWS account, the following four resources must have permissions granted to work with Object Lambda access points:

- The IAM user or role
- The bucket and associated standard access point
- The Object Lambda access point
- The AWS Lambda function

Make sure to resolve security warnings, errors, general warnings, and suggestions from AWS Identity and Access Management Access Analyzer before you save your policy. IAM Access Analyzer runs policy checks to validate your policy against IAM [policy grammar](#) and [best practices](#). These checks generate findings and provide actionable recommendations to help you author policies that are functional and conform to security best practices. To learn more about validating policies using IAM Access Analyzer, see [IAM Access Analyzer policy validation](#) in the *IAM User Guide*. To view a list of the warnings, errors, and suggestions that are returned by IAM Access Analyzer, see [IAM Access Analyzer policy check reference](#).

These examples assume that you have the following resources:

- An Amazon S3 bucket with following Amazon Resource Name (ARN):

`arn:aws:s3:::DOC-EXAMPLE-BUCKET1`

The following S3 bucket policy example delegates access control for a bucket to the bucket's access points. This policy allows full access to all access points owned by the bucket owner's account. Thus, all access to this bucket is controlled by the policies attached to its access points. Users can read from the bucket only through the S3 Access Point, which means that operations can be invoked only through access points. For more information, see [Delegating access control to access points \(p. 302\)](#).

Example – bucket policy delegating access control to access points

```
{
    "Version": "2012-10-17",
    "Statement" : [
        {
            "Effect": "Allow",
            "Principal" : { "AWS": "account-ARN" },
            "Action" : "*",
            "Resource" : [
                "arn:aws:s3:::DOC-EXAMPLE-BUCKET1",
                "arn:aws:s3:::DOC-EXAMPLE-BUCKET1/*"
            ],
            "Condition": {
                "StringEquals" : { "s3:DataAccessPointAccount" : "Bucket owner's account
ID" }
            }
        }
    ]
}
```

- An Amazon S3 standard access point on this bucket with the following ARN:

`arn:aws:s3:us-east-1:111122223333:accesspoint/my-access-point`

- An Object Lambda access point with the following ARN:

`arn:aws:s3-object-lambda:us-east-1:111122223333:accesspoint/my-object-lambda-
ap`

- An AWS Lambda function with the following ARN:

`arn:aws:lambda:us-east-1:111122223333:function:MyObjectLambdaFunction`

Note

If using a Lambda function from your account you must include the function version in your policy statement. For example, `arn:aws:lambda:us-
east-1:111122223333:function:MyObjectLambdaFunction:$LATEST`

The following IAM policy grants a user permission to the Lambda function, standard access point and the S3 Object Lambda access point.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "AllowLambdaInvocation",
            "Action": [
                "lambda:InvokeFunction"
            ],
            "Effect": "Allow",
            "Resource": [
                "arn:aws:s3:us-east-1:111122223333:accesspoint/my-access-point",
                "arn:aws:s3-object-lambda:us-east-1:111122223333:accesspoint/my-object-lambda-
ap",
                "arn:aws:lambda:us-east-1:111122223333:function:MyObjectLambdaFunction"
            ]
        }
    ]
}
```

```

    "Resource": "arn:aws:lambda:us-east-1:111122223333:function:MyObjectLambdaFunction:$LATEST",
    "Condition": {
        "ForAnyValue:StringEquals": {
            "aws:CalledVia": [
                "s3-object-lambda.amazonaws.com"
            ]
        }
    },
{
    "Sid": "AllowStandardAccessPointAccess",
    "Action": [
        "s3:Get*",
        "s3>List*"
    ],
    "Effect": "Allow",
    "Resource": "arn:aws:s3:us-east-1:111122223333:accesspoint/my-access-point/*",
    "Condition": {
        "ForAnyValue:StringEquals": {
            "aws:CalledVia": [
                "s3-object-lambda.amazonaws.com"
            ]
        }
    },
{
    "Sid": "AllowObjectLambdaAccess",
    "Action": [
        "s3-object-lambda:Get*",
        "s3-object-lambda>List*"
    ],
    "Effect": "Allow",
    "Resource": "arn:aws:s3-object-lambda:us-east-1:111122223333:accesspoint/my-object-lambda-ap"
}
]
}

```

Lambda execution role

Your Lambda function needs permission to send data to S3 Object Lambda when requests are made to an Object Lambda access point. This is provided by enabling the `s3-object-lambda:WriteGetObjectResponse` permission on your Lambda function's execution role. You can create a new execution role or update an existing one.

To create an execution role in the IAM console

1. Open the [Roles page](#) in the IAM console.
2. Choose **Create role**.
3. Under **Common use cases**, choose **Lambda**.
4. Choose **Next: Permissions**.
5. Under **Attach permissions policies**, choose the AWS managed policy [AmazonS3ObjectLambdaExecutionRolePolicy](#).
6. Choose **Next: Tags**.
7. Choose **Next: Review**.
8. For **Role name**, enter `s3-object-lambda-role`.
9. Choose **Create role**.
10. Apply the newly created `s3-object-lambda-role` as your Lambda function's execution role.

For detailed instructions, see [Creating a role for an AWS service \(console\)](#) in the *IAM User Guide*.

To update your Lambda function's execution role

Add the following statement to the execution role that is used by the Lambda function.

```
{  
    "Sid": "AllowObjectLambdaAccess",  
    "Action": [ "s3-object-lambda:WriteGetObjectResponse" ],  
    "Effect": "Allow",  
    "Resource": "*"  
}
```

For more information about execution roles, see [Lambda execution role](#) in the *AWS Lambda Developer Guide*.

Using context keys with Object Lambda access points

With S3 Object Lambda, GET requests will automatically invoke Lambda functions and all other requests will be forwarded to Amazon S3. S3 Object Lambda will evaluate context keys such as `s3-object-lambda:TlsVersion` or `s3-object-lambda:AuthType` related to the connection or signing of the request. All other context keys, such as `s3:prefix`, are evaluated by Amazon S3.

Writing and debugging AWS Lambda functions for Amazon S3 Object Lambda Access Points

This section details how to write and debug Lambda functions for use with S3 Object Lambda access points.

Topics

- [Working with WriteGetObjectResponse \(p. 284\)](#)
- [Debugging S3 Object Lambda \(p. 292\)](#)
- [Working with Range and partNumber headers \(p. 293\)](#)
- [Event context format and usage \(p. 295\)](#)

Working with WriteGetObjectResponse

S3 Object Lambda includes a new Amazon S3 API operation, `WriteGetObjectResponse`, which enables the Lambda function to provide customized data and response headers to the `GetObject` caller. `WriteGetObjectResponse` affords the Lambda author extensive control over the status code, response headers, and response body, based on their processing needs. You can use `WriteGetObjectResponse` to respond with the whole transformed object, portions of the transformed object, or other responses based on the context of your application. The following section shows unique examples of using the `WriteGetObjectResponse` API operation.

- **Example 1:** Respond with an HTTP status code 403 (Forbidden)
- **Example 2:** Respond with a transformed image
- **Example 3:** Stream compressed content

Example 1:

You can use `WriteGetObjectResponse` to respond with the HTTP status code 403 (Forbidden) based on the content of the object.

Java

```
package com.amazon.s3.objectlambda;

import com.amazonaws.services.lambda.runtime.Context;
import com.amazonaws.services.lambda.runtime.events.S3ObjectLambdaEvent;
import com.amazonaws.services.s3.AmazonS3;
import com.amazonaws.services.s3.AmazonS3Client;
import com.amazonaws.services.s3.model.WriteGetObjectResponseRequest;

import java.io.ByteArrayInputStream;
import java.net.URI;
import java.net.http.HttpClient;
import java.net.http.HttpRequest;
import java.net.http.HttpResponse;

public class Example1 {

    public void handleRequest(S3ObjectLambdaEvent event, Context context) throws
Exception {
        AmazonS3 s3Client = AmazonS3Client.builder().build();

        // Check to see if the request contains all of the necessary information.
        // If it does not, send a 4XX response and a custom error code and message.
        // Otherwise, retrieve the object from S3 and stream it
        // to the client unchanged.
        var tokenIsNotPresent = !
event.getUserRequest().getHeaders().containsKey("requiredToken");
        if (tokenIsNotPresent) {
            s3Client.writeGetObjectResponse(new WriteGetObjectResponseRequest()
                .withRequestRoute(event.outputRoute())
                .withRequestToken(event.outputToken())
                .withStatusCode(403)
                .withContentLength(0L).withInputStream(new ByteArrayInputStream(new
byte[0]))
                .withErrorCode("MissingRequiredToken")
                .withErrorMessage("The required token was not present in the
request."));
            return;
        }

        // Prepare the presigned URL for use and make the request to S3.
        HttpClient httpClient = HttpClient.newBuilder().build();
        var presignedResponse = httpClient.send(
            HttpRequest.newBuilder(new URI(event.inputS3Url())).GET().build(),
            HttpResponse.BodyHandlers.ofInputStream());

        // Stream the original bytes back to the caller.
        s3Client.writeGetObjectResponse(new WriteGetObjectResponseRequest()
            .withRequestRoute(event.outputRoute())
            .withRequestToken(event.outputToken())
            .withInputStream(presignedResponse.body()));
    }
}
```

Python

```
import boto3
import requests
```

```

def handler(event, context):
    s3 = boto3.client('s3')

    """
        Retrieve the operation context object from the event. This object indicates where
        the WriteGetObjectResponse request
            should be delivered and contains a presigned URL in 'inputS3Url' where we can
        download the requested object from.
        The 'userRequest' object has information related to the user who made this
        'GetObject' request to
            S3 Object Lambda.
    """
    get_context = event["getObjectContext"]
    user_request_headers = event["userRequest"]["headers"]

    route = get_context["outputRoute"]
    token = get_context["outputToken"]
    s3_url = get_context["inputS3Url"]

    # Check for the presence of a 'CustomHeader' header and deny or allow based on that
    # header.
    is_token_present = "SuperSecretToken" in user_request_headers

    if is_token_present:
        # If the user presented our custom 'SuperSecretToken' header, we send the
        # requested object back to the user.
        response = requests.get(s3_url)
        s3.write_get_object_response(RequestRoute=route, RequestToken=token,
                                     Body=response.content)
    else:
        # If the token is not present, we send an error back to the user.
        s3.write_get_object_response(RequestRoute=route, RequestToken=token,
                                     StatusCode=403,
                                     ErrorCode="NoSuperSecretTokenFound", ErrorMessage="The request was not secret
                                     enough.")

    # Gracefully exit the Lambda function.
    return { 'status_code': 200 }

```

Node.js

```

const { S3 } = require('aws-sdk');
const axios = require('axios').default;

exports.handler = async (event) => {
    const s3 = new S3();

    // Retrieve the operation context object from the event. This object indicates
    // where the WriteGetObjectResponse request
    // should be delivered and contains a presigned URL in 'inputS3Url' where we can
    // download the requested object from.
    // The 'userRequest' object has information related to the user who made this
    'GetObject' request to S3 Object Lambda.
    const { userRequest, getObjectContext } = event;
    const { outputRoute, outputToken, inputS3Url } = getObjectContext;

    // Check for the presence of a 'CustomHeader' header and deny or allow based on
    // that header.
    const isTokenPresent = Object
        .keys(userRequest.headers)
        .includes("SuperSecretToken");

    if (!isTokenPresent) {

```

```

        // If the token is not present, we send an error back to the user. The 'await'
in front of the request
        // indicates that we want to wait for this request to finish sending before
moving on.
        await s3.writeGetObjectResponse({
            RequestRoute: outputRoute,
            RequestToken: outputToken,
            StatusCode: 403,
            ErrorCode: "NoSuperSecretTokenFound",
            ErrorMessage: "The request was not secret enough.",
        }).promise();
    } else {
        // If the user presented our custom 'SuperSecretToken' header, we send the
requested object back to the user.
        // Again, note the presence of 'await'.
        const presignedResponse = await axios.get(inputs3Url);
        await s3.writeGetObjectResponse({
            RequestRoute: outputRoute,
            RequestToken: outputToken,
            Body: presignedResponse.data,
        }).promise();
    }

    // Gracefully exit the Lambda function.
    return { statusCode: 200 };
}

```

Example 2:

When performing an image transformation, you might find that you need all the bytes of the source object before you can start processing them. In this case, your `WriteGetObjectResponse` request returns the whole object to the requesting application in one call.

Java

```

package com.amazon.s3.objectlambda;

import com.amazonaws.services.lambda.runtime.Context;
import com.amazonaws.services.lambda.runtime.events.S3ObjectLambdaEvent;
import com.amazonaws.services.s3.AmazonS3;
import com.amazonaws.services.s3.AmazonS3Client;
import com.amazonaws.services.s3.model.WriteGetObjectResponseRequest;

import javax.imageio.ImageIO;
import java.awt.image.BufferedImage;
import java.awt.Image;
import java.io.ByteArrayInputStream;
import java.io.ByteArrayOutputStream;
import java.net.URI;
import java.net.http.HttpClient;
import java.net.http.HttpRequest;
import java.net.http.HttpResponse;

public class Example2 {

    private static final int HEIGHT = 250;
    private static final int WIDTH = 250;

    public void handleRequest(S3ObjectLambdaEvent event, Context context) throws
Exception {
        AmazonS3 s3Client = AmazonS3Client.builder().build();
        HttpClient httpClient = HttpClient.newBuilder().build();

```

```

// Prepare the presigned URL for use and make the request to S3.
var presignedResponse = httpClient.send(
    HttpRequest.newBuilder(new URI(event.inputS3Url())).GET().build(),
    HttpResponse.BodyHandlers.ofInputStream());

// The entire image is loaded into memory here so that we can resize it.
// Once the resizing is completed, we write the bytes into the body
// of the WriteGetObjectResponse.
var originalImage = ImageIO.read(presignedResponse.body());
var resizingImage = originalImage.getScaledInstance(WIDTH, HEIGHT,
Image.SCALE_DEFAULT);
var resizedImage = new BufferedImage(WIDTH, HEIGHT,
BufferedImage.TYPE_INT_RGB);
resizedImage.createGraphics().drawImage(resizingImage, 0, 0, WIDTH, HEIGHT,
null);

var baos = new ByteArrayOutputStream();
ImageIO.write(resizedImage, "png", baos);

// Stream the bytes back to the caller.
s3Client.writeGetObjectResponse(new WriteGetObjectResponseRequest()
    .withRequestRoute(event.outputRoute())
    .withRequestToken(event.outputToken())
    .withInputStream(new ByteArrayInputStream(baos.toByteArray())));
}
}

```

Python

```

import boto3
import requests
import io
from PIL import Image

def handler(event, context):
    """
        Retrieve the operation context object from the event. This object indicates where
        the WriteGetObjectResponse request
        should be delivered and has a presigned URL in 'inputS3Url' where we can download
        the requested object from.
        The 'userRequest' object has information related to the user who made this
        'GetObject' request to
        S3 Object Lambda.
    """
    get_context = event["getObjectContext"]
    route = get_context["outputRoute"]
    token = get_context["outputToken"]
    s3_url = get_context["inputS3Url"]

    """
        In this case, we're resizing '.png' images that are stored in S3 and are accessible
        through the presigned URL
        'inputS3Url'.
    """
    image_request = requests.get(s3_url)
    image = Image.open(io.BytesIO(image_request.content))
    image.thumbnail((256,256), Image.ANTIALIAS)

    transformed = io.BytesIO()
    image.save(transformed, "png")

    # Send the resized image back to the client.
    s3 = boto3.client('s3')

```

```
s3.writeGetObjectResponse(Body=transformed.getvalue(), RequestRoute=route,  
RequestToken=token)  
  
# Gracefully exit the Lambda function.  
return { 'status_code': 200 }
```

Node.js

```
const { S3 } = require('aws-sdk');  
const axios = require('axios').default;  
const sharp = require('sharp');  
  
exports.handler = async (event) => {  
    const s3 = new S3();  
  
    // Retrieve the operation context object from event. This object indicates where  
    // the WriteGetObjectResponse request  
    // should be delivered and has a presigned URL in 'inputS3Url' where we can  
    // download the requested object from.  
    const { getObjectContext } = event;  
    const { outputRoute, outputToken, inputS3Url } = getObjectContext;  
  
    // In this case, we're resizing '.png' images that are stored in S3 and are  
    // accessible through the presigned URL  
    // 'inputS3Url'.  
    const { data } = await axios.get(inputS3Url, { responseType: 'arraybuffer' });  
  
    // Resize the image.  
    const resized = await sharp(data)  
        .resize({ width: 256, height: 256 })  
        .toBuffer();  
  
    // Send the resized image back to the client.  
    await s3.writeGetObjectResponse({  
        RequestRoute: outputRoute,  
        RequestToken: outputToken,  
        Body: resized,  
    }).promise();  
  
    // Gracefully exit the Lambda function.  
    return { statusCode: 200 };  
}
```

Example 3:

When you're compressing objects, compressed data is produced incrementally. Consequently, you can use your `WriteGetObjectResponse` request to return the compressed data as soon as it's ready. As shown in this example, you don't need to know the length of the completed transformation.

Java

```
package com.amazon.s3.objectlambda;  
  
import com.amazonaws.services.lambda.runtime.events.S3ObjectLambdaEvent;  
import com.amazonaws.services.lambda.runtime.Context;  
import com.amazonaws.services.s3.AmazonS3;  
import com.amazonaws.services.s3.AmazonS3Client;  
import com.amazonaws.services.s3.model.WriteGetObjectResponseRequest;  
  
import java.net.URI;  
import java.net.http.HttpClient;
```

```
import java.net.http.HttpRequest;
import java.net.http.HttpResponse;

public class Example3 {

    public void handleRequest(S3ObjectLambdaEvent event, Context context) throws
Exception {
        AmazonS3 s3Client = AmazonS3Client.builder().build();
        HttpClient httpClient = HttpClient.newBuilder().build();

        // Request the original object from S3.
        var presignedResponse = httpClient.send(
            HttpRequest.newBuilder(new URI(event.inputS3Url())).GET().build(),
            HttpResponse.BodyHandlers.ofInputStream());

        // Consume the incoming response body from the presigned request,
        // apply our transformation on that data, and emit the transformed bytes
        // into the body of the WriteGetObjectResponse request as soon as they're
        ready.
        // This example compresses the data from S3, but any processing pertinent
        // to your application can be performed here.
        var bodyStream = new GZIPCompressingInputStream(presignedResponse.body());

        // Stream the bytes back to the caller.
        s3Client.writeGetObjectResponse(new WriteGetObjectResponseRequest()
            .withRequestRoute(event.outputRoute())
            .withRequestToken(event.outputToken())
            .withInputStream(bodyStream));
    }

}
```

Python

```
import boto3
import requests
import zlib
from botocore.config import Config

"""
A helper class to work with content iterators. Takes an interator and compresses the
bytes that come from it. It
implements 'read' and '__iter__' so that the SDK can stream the response.
"""
class Compress:
    def __init__(self, content_iter):
        self.content = content_iter
        self.compressed_obj = zlib.compressobj()

    def read(self, _size):
        for data in self.__iter__():
            return data

    def __iter__(self):
        while True:
            data = next(self.content)
            chunk = self.compressed_obj.compress(data)
            if not chunk:
                break

            yield chunk

        yield self.compressed_obj.flush()
```

```

def handler(event, context):
    """
        Setting the 'payload_signing_enabled' property to False allows us to send a
        streamed response back to the client.
        in this scenario, a streamed response means that the bytes are not buffered into
        memory as we're compressing them,
        but instead are sent straight to the user.
    """
    my_config = Config(
        region_name='eu-west-1',
        signature_version='s3v4',
        s3={
            "payload_signing_enabled": False
        }
    )
    s3 = boto3.client('s3', config=my_config)

    """
        Retrieve the operation context object from the event. This object indicates where
        the WriteGetObjectResponse request
        should be delivered and has a presigned URL in 'inputS3Url' where we can download
        the requested object from.
        The 'userRequest' object has information related to the user who made this
        'GetObject' request to S3 Object Lambda.
    """
    get_context = event["getObjectContext"]
    route = get_context["outputRoute"]
    token = get_context["outputToken"]
    s3_url = get_context["inputS3Url"]

    # Compress the 'get' request stream.
    with requests.get(s3_url, stream=True) as r:
        compressed = Compress(r.iter_content())

    # Send the stream back to the client.
    s3.write_get_object_response(Body=compressed, RequestRoute=route,
                                RequestToken=token, ContentType="text/plain",
                                ContentEncoding="gzip")

    # Gracefully exit the Lambda function.
    return {'status_code': 200}

```

Node.js

```

const { S3 } = require('aws-sdk');
const axios = require('axios').default;
const zlib = require('zlib');

exports.handler = async (event) => {
    const s3 = new S3();

    // Retrieve the operation context object from the event. This object indicates
    // where the WriteGetObjectResponse request
    // should be delivered and has a presigned URL in 'inputS3Url' where we can
    // download the requested object from.
    const { getObjectContext } = event;
    const { outputRoute, outputToken, inputS3Url } = getObjectContext;

    // Download the object from S3 and process it as a stream, because it might be a
    // huge object and we don't want to
    // buffer it in memory. Note the use of 'await' because we want to wait for
    'writeGetObjectResponse' to finish

```

```
// before we can exit the Lambda function.
await axios({
  method: 'GET',
  url: inputS3Url,
  responseType: 'stream',
}).then(
  // Gzip the stream.
  response => response.data.pipe(zlib.createGzip())
).then(
  // Finally send the gzip-ed stream back to the client.
  stream => s3.writeGetObjectResponse({
    RequestRoute: outputRoute,
    RequestToken: outputToken,
    Body: stream,
    ContentType: "text/plain",
    ContentEncoding: "gzip",
  }).promise()
);

// Gracefully exit the Lambda function.
return { statusCode: 200 };
}
```

Note

Although S3 Object Lambda allows up to 60 seconds to send a complete response to the caller through the `WriteGetObjectResponse` request, the actual amount of time available might be less. For instance, your Lambda function timeout might be less than 60 seconds. In other cases, the caller might have more stringent timeouts.

For the original caller to receive a non-500 (Internal Server Error) response, the `WriteGetObjectResponse` call must be completed. If the Lambda function returns, with an exception or otherwise, before the `WriteGetObjectResponse` API operation is called, the original caller receives a 500 response. Exceptions thrown during the time it takes to complete the response result in truncated responses to the caller. If the Lambda function receives a 200 (OK) response from the `WriteGetObjectResponse` API call, then the original caller has sent the complete request. The Lambda function's response, whether an exception is thrown or not, is ignored by S3 Object Lambda.

When calling this API operation, Amazon S3 requires the route and request token from the event context. For more information, see [Event context format and usage \(p. 295\)](#).

These parameters are required to connect the `WriteGetObjectResult` response with the original caller. Even though it is always appropriate to retry 500 responses, note that because the request token is a single-use token, subsequent attempts to use it might result in 400 (Bad Request) responses. Although the call to `WriteGetObjectResponse` with the route and request tokens does not need to be made from the invoked Lambda function, it must be made by an identity in the same account. The call also must be completed before the Lambda function finishes execution.

Debugging S3 Object Lambda

GetObject requests to S3 Object Lambda access points might result in a new error response when something goes wrong with the Lambda function invocation or execution. These errors follow the same format as standard Amazon S3 errors. For information about S3 Object Lambda errors, see [S3 Object Lambda Error Code List](#) in the *Amazon Simple Storage Service API Reference*.

For more information about general Lambda function debugging, see [Monitoring and troubleshooting Lambda applications](#) in the *AWS Lambda Developer Guide*.

For information about standard Amazon S3 errors, see [Error Responses](#) in the *Amazon Simple Storage Service API Reference*.

You can enable request metrics in Amazon CloudWatch for your Object Lambda access points. These metrics can be used to monitor the operational performance of your access point.

To get more granular logging about requests made to your Object Lambda access points, you can enable AWS CloudTrail data events. For more information, see [Logging data events for trails](#) in the [AWS CloudTrail User Guide](#).

Working with Range and partNumber headers

When working with large objects, you can use the `Range` HTTP header to download a specified byte range from an object. You can use concurrent connections to Amazon S3 to fetch different byte ranges from within the same object. You can also specify the `partNumber` parameter (an integer between 1 and 10,000), which effectively performs a "ranged" GET request for the specified part from the object. For more information, see [GetObject Request Syntax](#) in the [Amazon Simple Storage Service API Reference](#).

Because there are multiple ways that you might want to handle a request that includes the `Range` or `partNumber` parameters, S3 Object Lambda doesn't apply these parameters to the transformed object. Instead, your Lambda function must implement this functionality as needed for your application.

To use the `Range` and `partNumber` parameters with S3 Object Lambda, you enable these parameters for your Object Lambda access point, then write a Lambda function that can handle requests that include these parameters. The following steps describe how to accomplish this.

Step 1: Configure your Object Lambda access point

By default, Object Lambda access points respond with an HTTP 501 (Not Implemented) error to any `GetObject` request that contains a `Range` or `partNumber` parameter, either in the headers or query parameters. To enable an Object Lambda access point to accept such requests, you must update your Object Lambda access point configuration by using the AWS Management Console, AWS Command Line Interface (AWS CLI), or AWS SDKs. For more information about updating your Object Lambda access point configuration, see [Creating Object Lambda Access Points \(p. 273\)](#).

Step 2: Implement Range or partNumber handling in your Lambda function

When your Object Lambda access point invokes your Lambda function with a ranged `GetObject` request, the `Range` or `partNumber` parameter is included in the event context. The location of the parameter in the event context depends on which parameter was used and how it was included in the original request to the Object Lambda access point, as explained in the following table.

Parameter	Event context location
Range (header)	<code>userRequest.headers.Range</code>
Range (query parameter)	<code>userRequest.url (query parameter Range)</code>
<code>partNumber</code>	<code>userRequest.url (query parameter partNumber)</code>

Important

The presigned URL provided does not contain the `Range` or `partNumber` parameter from the original request. If you want to retrieve only the specified range from Amazon S3, you must add the parameter to the presigned URL.

After you extract the `Range` or `partNumber` value, you can take one of the following approaches based on your application's needs:

- A. Map the requested `Range` or `partNumber` to the transformed object (recommended)

The most reliable way to handle Range or partNumber requests is to retrieve the full object from S3, transform the object, and then apply the requested Range or partNumber parameters to the transformed object. To do this, use the provided presigned URL to fetch the entire object from Amazon S3 and then process the object as needed. For an example Lambda function that processes a Range parameter in this way, see [this sample](#) in the AWS Samples GitHub repository.

B. Map the requested Range or partNumber to the presigned URL

In some cases, your Lambda function can map the requested Range or partNumber directly to the presigned URL to retrieve only part of the object from Amazon S3. This approach is appropriate only if your transformation meets both of the following criteria:

1. Your transformation function can be applied to partial object ranges.
2. The Range or partNumber parameter acts on the same data in the original object as in the transformed object.

For example, a transformation function that converts all characters in an ASCII-encoded object to uppercase meets both of the preceding criteria. The transform can be applied to part of an object, and applying the Range or partNumber parameter before the transformation achieves the same result as applying the parameter after the transformation.

By contrast, a function that reverses the characters in an ASCII-encoded object doesn't meet these criteria. Such a function meets criterion 1, because it can be applied to partial object ranges. However, it doesn't meet criterion 2, because applying the Range parameter before the transformation achieves different results than applying the parameter after the transformation.

For example, consider a request to apply the function to the first three characters of an object with the contents abcdefg. Applying the Range parameter before the transformation retrieves only abc and then reverses the data, returning cba. But if the parameter is applied after the transformation, the function retrieves the entire object, reverses it, and then applies the Range parameter, returning gfe. Because these results are different, this function should not apply the Range parameter when retrieving the object from Amazon S3. Instead, it should retrieve the entire object, perform the transformation, and only then apply the Range parameter.

Warning

In many cases, applying the Range or partNumber parameter to the presigned URL will result in unexpected behavior by the Lambda function or the requesting client. Unless you are sure that your application will work properly when retrieving only a partial object from Amazon S3, we recommend that you retrieve and transform full objects as described earlier in approach A.

If your application meets the criteria described earlier, you can simplify your AWS Lambda function and minimize data-transfer costs by fetching only the requested object range and then running your transformation on that range.

The following Java code example demonstrates how to retrieve the Range header from the GetObject request and add it to the presigned URL that Lambda can use to retrieve the requested range from Amazon S3.

```
private HttpRequest.Builder applyRangeHeader(ObjectLambdaEvent event, HttpRequest.Builder presignedRequest) {
    var header = event.getUserRequest().getHeaders().entrySet().stream()
        .filter(e -> e.getKey().toLowerCase(Locale.ROOT).equals("range"))
        .findFirst();

    // Add check in the query string itself.
    header.ifPresent(entry -> presignedRequest.header(entry.getKey(), entry.getValue()));
    return presignedRequest;
}
```

Event context format and usage

S3 Object Lambda provides context about the request being made in the event passed to your Lambda function. The following shows an example request and field descriptions.

```
{
    "xAmzRequestId": "requestId",
    "getObjectContext": {
        "inputS3Url": "https://my-s3-ap-111122223333.s3-accesspoint.us-east-1.amazonaws.com/example?X-Amz-Security-Token=<snip>",
        "outputRoute": "io-use1-001",
        "outputToken": "OutputToken"
    },
    "configuration": {
        "accessPointArn": "arn:aws:s3-object-lambda:us-east-1:111122223333:accesspoint/example-object-lambda-ap",
        "supportingAccessPointArn": "arn:aws:s3:us-east-1:111122223333:accesspoint/example-ap",
        "payload": "{}"
    },
    "userRequest": {
        "url": "https://object-lambda-111122223333.s3-object-lambda.us-east-1.amazonaws.com/example",
        "headers": {
            "Host": "object-lambda-111122223333.s3-object-lambda.us-east-1.amazonaws.com",
            "Accept-Encoding": "identity",
            "X-Amz-Content-SHA256": "e3b0c44298fc1example"
        }
    },
    "userIdentity": {
        "type": "AssumedRole",
        "principalId": "principalId",
        "arn": "arn:aws:sts::111122223333:assumed-role/Admin/example",
        "accountId": "111122223333",
        "accessKeyId": "accessKeyId",
        "sessionContext": {
            "attributes": {
                "mfaAuthenticated": "false",
                "creationDate": "Wed Mar 10 23:41:52 UTC 2021"
            },
            "sessionIssuer": {
                "type": "Role",
                "principalId": "principalId",
                "arn": "arn:aws:iam::111122223333:role/Admin",
                "accountId": "111122223333",
                "userName": "Admin"
            }
        }
    },
    "protocolVersion": "1.00"
}
```

- **xAmzRequestId** – The Amazon S3 request ID for this request. We recommend that you log this value to help with debugging.
- **getObjectContext** – The input and output details for connections to Amazon S3 and S3 Object Lambda.
 - **inputS3Url** – A presigned URL that can be used to fetch the original object from Amazon S3. The URL is signed using the original caller's identity, and their permissions will apply when the URL is used. If there are signed headers in the URL, the Lambda function must include these in the call to Amazon S3, except for the **Host** header.

- `outputRoute` – A routing token that is added to the S3 Object Lambda URL when the Lambda function calls `WriteGetObjectResponse`.
- `outputToken` – An opaque token used by S3 Object Lambda to match the `WriteGetObjectResponse` call with the original caller.
- `configuration` – Configuration information about the Object Lambda access point.
 - `accessPointArn` – The Amazon Resource Name (ARN) of the Object Lambda access point that received this request.
 - `supportingAccessPointArn` – The ARN of the supporting access point that is specified in the Object Lambda access point configuration.
 - `payload` – Custom data that is applied to the Object Lambda access point configuration. S3 Object Lambda treats this data as an opaque string, so it might need to be decoded before use.
- `userRequest` – Information about the original call to S3 Object Lambda.
 - `url` – The decoded URL of the request as received by S3 Object Lambda, excluding any authorization-related query parameters.
 - `headers` – A map of string to strings containing the HTTP headers and their values from the original call, excluding any authorization-related headers. If the same header appears multiple times, the values from each instance of the same header are combined into a comma-delimited list. The case of the original headers is retained in this map.
 - `userIdentity` – Details about the identity that made the call to S3 Object Lambda. For more information, see [Logging data events for trails](#) in the *AWS CloudTrail User Guide*.
 - `type` – The type of identity.
 - `accountId` – The AWS account to which the identity belongs.
 - `userName` – The friendly name of the identity that made the call.
 - `principalId` – The unique identifier for the identity that made the call.
 - `arn` – The ARN of the principal who made the call. The last section of the ARN contains the user or role that made the call.
 - `sessionContext` – If the request was made with temporary security credentials, this element provides information about the session that was created for those credentials.
 - `invokedBy` – The name of the AWS service that made the request, such as Amazon EC2 Auto Scaling or AWS Elastic Beanstalk.
 - `sessionIssuer` – If the request was made with temporary security credentials, this element provides information about how the credentials were obtained.
 - `protocolVersion` – The version ID of the context provided. The format of this field is `{Major Version}.{Minor Version}`. The minor version numbers are always two-digit numbers. Any removal or change to the semantics of a field necessitates a major version bump and requires active opt-in. Amazon S3 can add new fields at any time, at which point you might experience a minor version bump. Because of the nature of software rollouts, you might see multiple minor versions in use at once.

Using AWS built Lambda functions

AWS provides some prebuilt Lambda functions that you can use with S3 Object Lambda to detect and redact personally identifiable information (PII) and decompress S3 objects. These Lambda functions are available in the AWS Serverless Application Repository and can be selected through the AWS Management Console when you create your Object Lambda access point.

For more information on how to deploy serverless applications from the AWS Serverless Application Repository, see [Deploying Applications](#) in the *AWS Serverless Application Repository Developer Guide*.

Example 1: PII Access Control

This Lambda function uses Amazon Comprehend, a natural language processing (NLP) service using machine learning to find insights and relationships in text. It automatically detects personally identifiable information (PII) such as names, addresses, dates, credit card numbers, and social security numbers from documents in your Amazon S3 bucket. If you have documents in your bucket that include PII, you can configure the PII Access Control S3 Object Lambda function to detect these PII entity types and restrict access to unauthorized users.

To get started, simply deploy the following Lambda function in your account and add the ARN in your Object Lambda access point configuration.

ARN:

```
arn:aws:serverlessrepo:us-east-1:839782855223:applications/  
ComprehendPiiAccessControlS3ObjectLambda
```

You can add the view this function on the AWS Management Console using the following SAR link: [ComprehendPiiAccessControlS3ObjectLambda](#).

To view this function on GitHub see [Amazon Comprehend S3 Object Lambda](#).

Example 2: PII Redaction

This Lambda function uses Amazon Comprehend, a natural language processing (NLP) service using machine learning to find insights and relationships in text. It automatically redacts personally identifiable information (PII) such as names, addresses, dates, credit card numbers, and social security numbers from documents in your Amazon S3 bucket. If you have documents in your bucket that include information such as credit card numbers or bank account information, you can configure the PII Redaction S3 Object Lambda function to detect PII and then return a copy of these documents in which PII entity types are redacted.

To get started, simply deploy the following Lambda function in your account and add the ARN in your Object Lambda access point configuration.

ARN:

```
arn:aws:serverlessrepo:us-east-1:839782855223:applications/  
ComprehendPiiRedactionS3ObjectLambda
```

You can add the view this function on the AWS Management Console using the following SAR link: [ComprehendPiiRedactionS3ObjectLambda](#).

To view this function on GitHub see [Amazon Comprehend S3 Object Lambda](#).

Example 3: Decompression

The Lambda function S3ObjectLambdaDecompression, is equipped to decompress objects stored in S3 in one of six compressed file formats including bzip2, gzip, snappy, zlib, zstandard and ZIP. To get started, simply deploy the following Lambda function in your account and add the ARN in your Object Lambda access point configuration.

ARN:

```
arn:aws:serverlessrepo:eu-west-1:123065155563:applications/S3ObjectLambdaDecompression
```

You can add the view this function on the AWS Management Console using the following SAR link: [S3ObjectLambdaDecompression](#).

To view this function on GitHub see [S3 Object Lambda Decompression](#).

Best practices and guidelines for S3 Object Lambda

When using S3 Object Lambda, follow these best practices and guidelines to optimize operations and performance.

Topics

- [Working with S3 Object Lambda \(p. 298\)](#)
- [AWS Services used in connection with S3 Object Lambda \(p. 298\)](#)
- [Working with Range and partNumber GET headers \(p. 298\)](#)
- [Working with AWS CLI and SDKs \(p. 299\)](#)

Working with S3 Object Lambda

S3 Object Lambda only support processing GetObject requests. Any non-GET requests, such as ListObjects or HeadObject, will not invoke Lambda and return standard, non-transformed API responses. You can create a maximum of 1,000 Object Lambda access points per AWS account per Region. The AWS Lambda function that you use must be in the same AWS account and Region as the Object Lambda access point.

S3 Object Lambda allows up to 60 seconds to stream a complete response to its caller. Your function is also subject to Lambda default quotas. For more information, see [Lambda quotas](#) in the *AWS Lambda Developer Guide*. Using S3 Object Lambda invokes your specified Lambda function and you are responsible for ensuring that any data overwritten or deleted from S3 by your specified Lambda function or application is intended and correct.

You can only use S3 Object Lambda to perform operations on objects. You cannot use them to perform other Amazon S3 operations, such as modifying or deleting buckets. For a complete list of S3 operations that support access points see, [Access point compatibility with AWS services \(p. 315\)](#).

In addition to this list, S3 Object Lambda access points do not support [POST Object](#), [Copy](#) (as the source), or [Select Object Content](#).

AWS Services used in connection with S3 Object Lambda

S3 Object Lambda connects Amazon S3, AWS Lambda, and optionally, other AWS services of your choosing to deliver objects relevant to requesting applications. All AWS services used in connection with S3 Object Lambda will continue to be governed by their respective Service Level Agreements (SLA). For example, in the event that any AWS service does not meet its Service Commitment, you will be eligible to receive a Service Credit as documented in the service's SLA.

Working with Range and partNumber GET headers

When working with large objects you can use the Range HTTP header to download a specified byte-range from an object fetching only the specified portion. You can use concurrent connections to Amazon S3 to fetch different byte ranges from within the same object. You can also use partNumber (integer between 1 and 10,000) which effectively performs a 'ranged' GET request for the specified part from the object. For more information, see [GetObject Request Syntax](#) in the Amazon Simple Storage Service API Reference.

When receiving a GET request, S3 Object Lambda invokes your specified Lambda function first, hence if your GET request contains range or part number parameters, you must ensure that your Lambda

function is equipped to recognize and manage these parameters. Because there can be multiple entities connected in such a setup (requesting client and services like Lambda, S3, other) it is advised that all involved entities interpret the requested range (or partNumber) in a uniform manner. This ensures that the ranges the application is expecting match with the ranges your Lambda function is processing. When building a function to handle requests with range headers test all combinations of response sizes, original object sizes, and request range sizes that your application plans to use.

By default, S3 Object Lambda access points will respond with a 501 to any GetObject request that contains a range or part number parameter, either in the headers or query parameters. You can confirm that your Lambda function is prepared to handle range or part requests by updating your Object Lambda access point configuration through the AWS Management Console or the AWS CLI.

Working with AWS CLI and SDKs

AWS CLI S3 subcommands (cp, mv and sync) and use of Transfer Manager is not supported in conjunction with S3 Object Lambda.

Security considerations for S3 Object Lambda access points

S3 Object Lambda allows customers the ability to perform custom transformations on data as it leaves S3 using the scale and flexibility of AWS Lambda as a compute platform. S3 and Lambda remain secure by default, but special consideration by the Lambda author is required in order to maintain this security. S3 Object Lambda requires that all access be made by authenticated principals (no anonymous access) and over HTTPS.

To mitigate this risk we recommend that the Lambda execution role be carefully scoped to the smallest set of privileges possible. Additionally, the Lambda should make its S3 accesses via the provided pre-signed URL whenever possible.

Configuring IAM policies

S3 access points support AWS Identity and Access Management (IAM) resource policies that allow you to control the use of the access point by resource, user, or other conditions. For more information, see [Configuring IAM policies for Object Lambda access points \(p. 281\)](#).

Encryption behavior

Since Object Lambda access point use both Amazon S3 and AWS Lambda there are differences in encryption behavior. For more information about default S3 encryption behavior, see [Setting default server-side encryption behavior for Amazon S3 buckets \(p. 132\)](#).

- When using S3 server-side encryption with Object Lambda access points the object will be decrypted before being sent to AWS Lambda where it will be processed unencrypted up to the original caller (in case of a GET).
- To prevent the key being logged, S3 will reject GET requests for objects encrypted via server-side encryption using customer provided keys. The Lambda function may still retrieve these objects provided it has access to the client provided key.
- When using S3 client-side encryption with Object Lambda access points make sure Lambda has access to the key to decrypt and reencrypt the object.

Access points security

S3 Object Lambda uses two access points, an Object Lambda access point and a standard S3 access point, referred to as the supporting access point. When you make a request to an Object Lambda access

point, S3 either invokes Lambda on your behalf or delegates the request to the supporting access point, depending upon the S3 Object Lambda configuration. When Lambda is invoked for GetObject, S3 will generate a pre-signed URL to your object on your behalf through the supporting access point. Your Lambda function will receive this URL as input when invoked.

You may set your Lambda function to use this URL to retrieve the original object, instead of invoking S3 directly. This model allows you to apply better security boundaries to your objects. You can limit direct object access through S3 buckets or S3 access points to a limited set of IAM roles or users. This also protects your Lambda functions from being subject to the Confused Deputy problem, where a misconfigured function with different permissions than your GetObject invoker could allow or deny access to objects when it should not.

Object Lambda Access Point public access

S3 Object Lambda does not allow anonymous or public access because Amazon S3 needs to authorize your identity to complete any S3 Object Lambda request. When invoking GetObject requests through an Object Lambda access point, you need the `lambda:InvokeFunction` permission for the configured Lambda function. Similarly, when invoking other APIs through an Object Lambda access point, you need to have the required `s3:*` permissions.

Without these permissions, requests to invoke Lambda or delegate to S3 will fail as a 403 Forbidden error. All access must be made by authenticated principals. If you require public access, `Lambda@Edge` can be used as a possible alternative. For more information, see [Customizing at the edge with Lambda@Edge](#) in the *Amazon CloudFront Developer Guide*.

Object Lambda Access Point IPs

The describe-managed-prefix-lists subnets support Gateway VPC endpoints and are related to the routing table of VPC endpoints. The missing ranges belong to Amazon S3, but are not supported by Gateway VPC endpoints. For more information about describe-managed-prefix-lists see, [DescribePrefixLists](#) in the *Amazon EC2 API Reference* and [AWS IP address ranges](#) in the *AWS General Reference*.

S3 Object Lambda tutorials

The following tutorials present complete end-to-end procedures for some S3 Object Lambda tasks.

- [Tutorial: Transforming data for your application with S3 Object Lambda \(p. 27\)](#)
- [Tutorial: Detecting and redacting PII data with S3 Object Lambda and Amazon Comprehend \(p. 41\)](#)

For S3 Object Lambda tutorials, see [Tutorial: Transforming data for your application with S3 Object Lambda \(p. 27\)](#) and [Tutorial: Detecting and redacting PII data with S3 Object Lambda and Amazon Comprehend \(p. 41\)](#).

For more information about standard access points, see [Managing data access with Amazon S3 access points \(p. 301\)](#).

For information about working with buckets, see [Buckets overview \(p. 114\)](#). For information about working with objects, see [Amazon S3 objects overview \(p. 149\)](#).

Managing data access with Amazon S3 access points

Amazon S3 access points simplify data access for any AWS service or customer application that stores data in S3. Access points are named network endpoints that are attached to buckets that you can use to perform S3 object operations, such as `GetObject` and `PutObject`. Each access point has distinct permissions and network controls that S3 applies for any request that is made through that access point. Each access point enforces a customized access point policy that works in conjunction with the bucket policy that is attached to the underlying bucket. You can configure any access point to accept requests only from a virtual private cloud (VPC) to restrict Amazon S3 data access to a private network. You can also configure custom block public access settings for each access point.

Note

- You can only use access points to perform operations on objects. You can't use access points to perform other Amazon S3 operations, such as modifying or deleting buckets. For a complete list of S3 operations that support access points, see [Access point compatibility with AWS services \(p. 315\)](#).
- Access points work with some, but not all, AWS services and features. For example, you can't configure Cross-Region Replication to operate through an access point. For a complete list of AWS services that are compatible with S3 access points, see [Access point compatibility with AWS services \(p. 315\)](#).

This section explains how to work with Amazon S3 access points. For information about working with buckets, see [Buckets overview \(p. 114\)](#). For information about working with objects, see [Amazon S3 objects overview \(p. 149\)](#).

Topics

- [Configuring IAM policies for using access points \(p. 301\)](#)
- [Creating access points \(p. 306\)](#)
- [Using access points \(p. 310\)](#)
- [Access points restrictions and limitations \(p. 318\)](#)

Configuring IAM policies for using access points

Amazon S3 access points support AWS Identity and Access Management (IAM) resource policies that allow you to control the use of the access point by resource, user, or other conditions. For an application or user to be able to access objects through an access point, both the access point and the underlying bucket must permit the request.

Important

Adding an S3 access point to a bucket doesn't change the bucket's behavior when accessed through the existing bucket name or ARN. All existing operations against the bucket will continue to work as before. Restrictions that you include in an access point policy apply only to requests made through that access point.

Condition keys

S3 access points introduce three new condition keys that can be used in IAM policies to control access to your resources:

s3:DataAccessPointArn

This is a string that you can use to match on an access point ARN. The following example matches all access points for AWS account 123456789012 in Region us-west-2:

```
"Condition" : {
    "StringLike": {
        "s3:DataAccessPointArn": "arn:aws:s3:us-west-2:123456789012:accesspoint/*"
    }
}
```

s3:DataAccessPointAccount

This is a string operator that you can use to match on the account ID of the owner of an access point. The following example matches all access points owned by AWS account 123456789012.

```
"Condition" : {
    "StringEquals": {
        "s3:DataAccessPointAccount": "123456789012"
    }
}
```

s3:AccessPointNetworkOrigin

This is a string operator that you can use to match on the network origin, either Internet or VPC. The following example matches only access points with a VPC origin.

```
"Condition" : {
    "StringEquals": {
        "s3:AccessPointNetworkOrigin": "VPC"
    }
}
```

For more information about using condition keys with Amazon S3, see [Actions, resources, and condition keys for Amazon S3 \(p. 431\)](#).

Delegating access control to access points

You can delegate access control for a bucket to the bucket's access points. The following example bucket policy allows full access to all access points owned by the bucket owner's account. Thus, all access to this bucket is controlled by the policies attached to its access points. We recommend configuring your buckets this way for all use cases that don't require direct access to the bucket.

Example Bucket policy delegating access control to access points

```
{
    "Version": "2012-10-17",
    "Statement" : [
{
    "Effect": "Allow",
```

```
    "Principal" : { "AWS": "*" },
    "Action" : "*",
    "Resource" : [ "Bucket ARN", "Bucket ARN/*"],
    "Condition": {
        "StringEquals" : { "s3:DataAccessPointAccount" : "Bucket owner's account ID" }
    }
}]}
```

Access point policy examples

The following examples demonstrate how to create IAM policies to control requests made through an access point.

Note

Permissions granted in an access point policy are only effective if the underlying bucket also allows the same access. You can accomplish this in two ways:

1. (Recommended) Delegate access control from the bucket to the access point as described in [Delegating access control to access points \(p. 302\)](#).
2. Add the same permissions contained in the access point policy to the underlying bucket's policy. The first access point policy example demonstrates how to modify the underlying bucket policy to allow the necessary access.

Example Access point policy grant

The following access point policy grants IAM user Alice in account 123456789012 permissions to GET and PUT objects with the prefix Alice/ through access point my-access-point in account 123456789012.

```
{ "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "AWS": "arn:aws:iam::123456789012:user/Alice"
      },
      "Action": ["s3:GetObject", "s3:PutObject"],
      "Resource": "arn:aws:s3:us-west-2:123456789012:accesspoint/my-access-point/object/
Alice/*"
    }
  ]
}
```

Note

For the access point policy to effectively grant access to Alice, the underlying bucket must also allow the same access to Alice. You can delegate access control from the bucket to the access point as described in [Delegating access control to access points \(p. 302\)](#). Or, you can add the following policy to the underlying bucket to grant the necessary permissions to Alice. Note that the Resource entry differs between the access point and bucket policies.

```
{ "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
```

```
        "AWS": "arn:aws:iam::123456789012:user/Alice"
    },
    "Action": ["s3:GetObject", "s3:PutObject"],
    "Resource": "arn:aws:s3:::awsexamplebucket1/Alice/*"
}
}
```

Example Access point policy with tag condition

The following access point policy grants IAM user Bob in account 123456789012 permissions to GET objects through access point `my-access-point` in account 123456789012 that have the tag key `data` set with a value of `finance`.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Principal": {
                "AWS": "arn:aws:iam::123456789012:user/Bob"
            },
            "Action": "s3:GetObject",
            "Resource": "arn:aws:s3:us-west-2:123456789012:accesspoint/my-access-point/object/*",
            "Condition": {
                "StringEquals": {
                    "s3:ExistingObjectTag/data
```

Example Access point policy allowing bucket listing

The following access point policy allows IAM user Charles in account 123456789012 permission to view the objects contained in the bucket underlying access point `my-access-point` in account 123456789012.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Principal": {
                "AWS": "arn:aws:iam::123456789012:user/Charles"
            },
            "Action": "s3>ListBucket",
            "Resource": "arn:aws:s3:us-west-2:123456789012:accesspoint/my-access-point"
        }
    ]
}
```

Example Service control policy

The following service control policy requires all new access points to be created with a VPC network origin. With this policy in place, users in your organization can't create new access points that are accessible from the internet.

```
{
    "Version": "2012-10-17",
```

```
"Statement": [
{
    "Effect": "Deny",
    "Principal": "*",
    "Action": "s3>CreateAccessPoint",
    "Resource": "*",
    "Condition": {
        "StringNotEquals": {
            "s3:AccessPointNetworkOrigin": "VPC"
        }
    }
}]}
```

Example Bucket policy to limit S3 operations to VPC network origins

The following bucket policy limits access to all S3 object operations for bucket `examplebucket` to access points with a VPC network origin.

Important

Before using a statement like this example, make sure you don't need to use features that aren't supported by access points, such as Cross-Region Replication.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Deny",
            "Principal": "*",
            "Action": [
                "s3:AbortMultipartUpload",
                "s3:BypassGovernanceRetention",
                "s3>DeleteObject",
                "s3>DeleteObjectTagging",
                "s3>DeleteObjectVersion",
                "s3>DeleteObjectVersionTagging",
                "s3:GetObject",
                "s3:GetObjectAcl",
                "s3:GetObjectLegalHold",
                "s3:GetObjectRetention",
                "s3:GetObjectTagging",
                "s3:GetObjectVersion",
                "s3:GetObjectVersionAcl",
                "s3:GetObjectVersionTagging",
                "s3>ListMultipartUploadParts",
                "s3:PutObject",
                "s3:PutObjectAcl",
                "s3:PutObjectLegalHold",
                "s3:PutObjectRetention",
                "s3:PutObjectTagging",
                "s3:PutObjectVersionAcl",
                "s3:PutObjectVersionTagging",
                "s3:RestoreObject"
            ],
            "Resource": "arn:aws:s3:::examplebucket/*",
            "Condition": {
                "StringNotEquals": {
                    "s3:AccessPointNetworkOrigin": "VPC"
                }
            }
        }
    ]
}
```

Creating access points

Amazon S3 provides functionality for creating and managing access points. You can create S3 access points using the AWS Management Console, AWS Command Line Interface (AWS CLI), AWS SDKs, or Amazon S3 REST API.

By default, you can create up to 10,000 access points per Region for each of your AWS accounts. If you need more than 10,000 access points for a single account in a single Region, you can request a service quota increase. For more information about service quotas and requesting an increase, see [AWS Service Quotas](#) in the *AWS General Reference*.

Note

Because you might want to publicize your access point name so that other users can use the access point, avoid including sensitive information in the access point name. Access point names are published in a publicly accessible database known as the Domain Name System (DNS).

Rules for naming Amazon S3 access points

Access point names must meet the following conditions:

- Must be unique within a single AWS account and Region
- Must comply with DNS naming restrictions
- Must begin with a number or lowercase letter
- Must be between 3 and 50 characters long
- Can't begin or end with a dash
- Can't contain underscores, uppercase letters, or periods
- Can't end with the suffix `-s3alias`. This suffix is reserved for access point alias names. For more information, see [Using a bucket-style alias for your access point \(p. 314\)](#).

To create an access point, see the topics below.

Topics

- [Creating an access point \(p. 306\)](#)
- [Creating access points restricted to a virtual private cloud \(p. 308\)](#)
- [Managing public access to access points \(p. 309\)](#)

Creating an access point

An access point is associated with exactly one Amazon S3 bucket. Before you begin, make sure that you have created a bucket that you want to use with this access point. For more information about creating buckets, see [Creating, configuring, and working with Amazon S3 buckets \(p. 114\)](#). Amazon S3 access points support AWS Identity and Access Management (IAM) resource policies that allow you to control the use of the access point by resource, user, or other conditions. For more information, see [Configuring IAM policies for using access points \(p. 301\)](#).

Make sure to resolve security warnings, errors, general warnings, and suggestions from AWS Identity and Access Management Access Analyzer before you save your policy. IAM Access Analyzer runs policy checks to validate your policy against IAM [policy grammar](#) and [best practices](#). These checks generate findings and provide actionable recommendations to help you author policies that are functional and conform to security best practices. To learn more about validating policies using IAM Access Analyzer, see [IAM Access](#)

[Analyzer policy validation](#) in the *IAM User Guide*. To view a list of the warnings, errors, and suggestions that are returned by IAM Access Analyzer, see [IAM Access Analyzer policy check reference](#).

By default, you can create up to 10,000 access points per Region for each of your AWS accounts. If you need more than 10,000 access points for a single account in a single Region, you can request a service quota increase. For more information about service quotas and requesting an increase, see [AWS Service Quotas](#) in the *AWS General Reference*.

The following examples demonstrate how to create an access point with the AWS CLI and the S3 console. For more information about how to create access points using the REST API, see [CreateAccessPoint](#) in the *Amazon Simple Storage Service API Reference*.

Using the S3 console

To create an access point

1. Sign in to the AWS Management Console and open the Amazon S3 console at <https://console.aws.amazon.com/s3/>.
2. In the navigation pane on the left side of the console, choose **Access points**.
3. On the access points page, choose **Create access point**.
4. In the **Access point name** field, enter your desired name for the access point. For more information about naming access points, see [Rules for naming Amazon S3 access points \(p. 306\)](#).
5. In the **Bucket name** field, enter the name of a bucket in your account to which you want to attach the access point, for example **DOC-EXAMPLE-BUCKET1**. Optionally, you can choose **Browse S3** to browse and search buckets in your account. If you choose **Browse S3**, select the desired bucket and choose **Choose path** to populate the **Bucket name** field with that bucket's name.
6. (Optional) Choose **View** to view the contents of the specified bucket in a new browser window.
7. Select a **Network origin**. If you choose **Virtual private cloud (VPC)**, enter the **VPC ID** that you want to use with the access point.

For more information about network origins for access points, see [Creating access points restricted to a virtual private cloud \(p. 308\)](#).

8. Under **Block Public Access settings for this Access Point**, select the block public access settings that you want to apply to the access point. All block public access settings are enabled by default for new access points, and we recommend that you leave all settings enabled unless you know you have a specific need to disable any of them. Amazon S3 currently doesn't support changing an access point's block public access settings after the access point has been created.

For more information about using Amazon S3 Block Public Access with access points, see [Managing public access to access points \(p. 309\)](#).

9. (Optional) Under **Access Point policy - optional**, specify the access point policy. Make sure to resolve security warnings, errors, general warnings, and suggestions before you save your policy. For more information about specifying an access point policy, see [Access point policy examples \(p. 303\)](#).
10. Choose **Create access point**.

Using the AWS CLI

The following example creates an access point named `example-ap` for bucket `example-bucket` in account `123456789012`. To create the access point, you send a request to Amazon S3, specifying the access point name, the name of the bucket that you want to associate the access point with, and the account ID for the AWS account that owns the bucket. For information about naming rules, see [the section called "Rules for naming Amazon S3 access points" \(p. 306\)](#).

```
aws s3control create-access-point --name example-ap --account-id 123456789012 --bucket example-bucket
```

Creating access points restricted to a virtual private cloud

When you create an access point, you can choose to make the access point accessible from the internet, or you can specify that all requests made through that access point must originate from a specific virtual private cloud (VPC). An access point that's accessible from the internet is said to have a network origin of `Internet`. It can be used from anywhere on the internet, subject to any other access restrictions in place for the access point, underlying bucket, and related resources, such as the requested objects. An access point that's only accessible from a specified VPC has a network origin of `VPC`, and Amazon S3 rejects any request made to the access point that doesn't originate from that VPC.

Important

You can only specify an access point's network origin when you create the access point. After you create the access point, you can't change its network origin.

To restrict an access point to VPC-only access, you include the `VpcConfiguration` parameter with the request to create the access point. In the `VpcConfiguration` parameter, you specify the VPC ID that you want to be able to use the access point. If a request is made through the access point, the request must originate from the VPC or Amazon S3 will reject it.

You can retrieve an access point's network origin using the AWS CLI, AWS SDKs, or REST APIs. If an access point has a VPC configuration specified, its network origin is `VPC`. Otherwise, the access point's network origin is `Internet`.

Example

Example: Create an access point that's restricted to VPC access

The following example creates an access point named `example-vpc-ap` for bucket `example-bucket` in account `123456789012` that allows access only from the `vpc-1a2b3c` VPC. The example then verifies that the new access point has a network origin of `VPC`.

AWS CLI

```
aws s3control create-access-point --name example-vpc-ap --account-id 123456789012 --bucket example-bucket --vpc-configuration VpcId=vpc-1a2b3c
```



```
aws s3control get-access-point --name example-vpc-ap --account-id 123456789012

{
    "Name": "example-vpc-ap",
    "Bucket": "example-bucket",
    "NetworkOrigin": "VPC",
    "VpcConfiguration": {
        "VpcId": "vpc-1a2b3c"
    },
    "PublicAccessBlockConfiguration": {
        "BlockPublicAcls": true,
        "IgnorePublicAcls": true,
        "BlockPublicPolicy": true,
        "RestrictPublicBuckets": true
    },
    "CreationDate": "2019-11-27T00:00:00Z"
}
```

To use an access point with a VPC, you must modify the access policy for your VPC endpoint. VPC endpoints allow traffic to flow from your VPC to Amazon S3. They have access control policies that control how resources within the VPC are allowed to interact with Amazon S3. Requests from your VPC

to Amazon S3 only succeed through an access point if the VPC endpoint policy grants access to both the access point and the underlying bucket.

Note

To make resources accessible only within a VPC, make sure to create a [private hosted zone](#) for your VPC endpoint. To use a private hosted zone, [modify your VPC settings](#) so that the [VPC network attributes](#) enableDnsHostnames and enableDnsSupport are set to true.

The following example policy statement configures a VPC endpoint to allow calls to GetObject for a bucket named awsexamplebucket1 and an access point named example-vpc-ap.

```
{  
    "Version": "2012-10-17",  
    "Statement": [  
        {  
            "Principal": "*",  
            "Action": [  
                "s3:GetObject"  
            ],  
            "Effect": "Allow",  
            "Resource": [  
                "arn:aws:s3:::awsexamplebucket1/*",  
                "arn:aws:s3:us-west-2:123456789012:accesspoint/example-vpc-ap/object/*"  
            ]  
        }  
    ]  
}
```

Note

The "Resource" declaration in this example uses an Amazon Resource Name (ARN) to specify the access point. For more information about access point ARNs, see [Using access points \(p. 310\)](#).

For more information about VPC endpoint policies, see [Using endpoint policies for Amazon S3](#) in the *VPC User Guide*.

Managing public access to access points

Amazon S3 access points support independent *block public access* settings for each access point. When you create an access point, you can specify block public access settings that apply to that access point. For any request made through an access point, Amazon S3 evaluates the block public access settings for that access point, the underlying bucket, and the bucket owner's account. If any of these settings indicate that the request should be blocked, Amazon S3 rejects the request.

For more information about the S3 Block Public Access feature, see [Blocking public access to your Amazon S3 storage \(p. 584\)](#).

Important

- All block public access settings are enabled by default for access points. You must explicitly disable any settings that you don't want to apply to an access point.
- Amazon S3 currently doesn't support changing an access point's block public access settings after the access point has been created.

Example

Example: Create an access point with Custom Block Public Access Settings

This example creates an access point named example-ap for bucket example-bucket in account 123456789012 with non-default Block Public Access settings. The example then retrieves the new access point's configuration to verify its Block Public Access settings.

AWS CLI

```
aws s3control create-access-point --name example-ap --account-id 123456789012 --bucket example-bucket --public-access-block-configuration BlockPublicAccls=false,IgnorePublicAccls=false,BlockPublicPolicy=true,RestrictPublicBuckets=true
```

```
aws s3control get-access-point --name example-ap --account-id 123456789012

{
    "Name": "example-ap",
    "Bucket": "example-bucket",
    "NetworkOrigin": "Internet",
    "PublicAccessBlockConfiguration": {
        "BlockPublicAccls": false,
        "IgnorePublicAccls": false,
        "BlockPublicPolicy": true,
        "RestrictPublicBuckets": true
    },
    "CreationDate": "2019-11-27T00:00:00Z"
}
```

Using access points

You can access the objects in an Amazon S3 bucket with an *access point* using the AWS Management Console, AWS CLI, AWS SDKs, or the S3 REST APIs.

Access points have Amazon Resource Names (ARNs). Access point ARNs are similar to bucket ARNs, but they are explicitly typed and encode the access point's Region and the AWS account ID of the access point's owner. For more information about ARNs, see [Amazon Resource Names \(ARNs\)](#) in the [AWS General Reference](#).

Access point ARNs use the format `arn:aws:s3:<region>:<account-id>:accesspoint/<resource>`. For example:

- `arn:aws:s3:us-west-2:123456789012:accesspoint/test` represents the access point named `test`, owned by account `123456789012` in Region `us-west-2`.
- `arn:aws:s3:us-west-2:123456789012:accesspoint/*` represents all access points under account `123456789012` in Region `us-west-2`.

ARNs for objects accessed through an access point use the format `arn:aws:s3:<region>:<account-id>:accesspoint/<access-point-name>/object/<resource>`. For example:

- `arn:aws:s3:us-west-2:123456789012:accesspoint/test/object/unit-01` represents the object `unit-01`, accessed through the access point named `test`, owned by account `123456789012` in Region `us-west-2`.
- `arn:aws:s3:us-west-2:123456789012:accesspoint/test/object/*` represents all objects for access point `test`, in account `123456789012` in Region `us-west-2`.
- `arn:aws:s3:us-west-2:123456789012:accesspoint/test/object/unit-01/finance/*` represents all objects under prefix `unit-01/finance/` for access point `test`, in account `123456789012` in Region `us-west-2`.

Topics

- [Monitoring and logging access points \(p. 311\)](#)

- [Using Amazon S3 access points with the Amazon S3 console \(p. 312\)](#)
- [Using a bucket-style alias for your access point \(p. 314\)](#)
- [Using access points with compatible Amazon S3 operations \(p. 315\)](#)

Monitoring and logging access points

Amazon S3 logs requests made through access points and requests made to the APIs that manage access points, such as `CreateAccessPoint` and `GetAccessPointPolicy`. To monitor and manage usage patterns, you can also configure Amazon CloudWatch Logs request metrics for access points.

Topics

- [CloudWatch request metrics \(p. 311\)](#)
- [Request logs \(p. 311\)](#)

CloudWatch request metrics

To understand and improve the performance of applications that are using access points, you can use CloudWatch for Amazon S3 request metrics. Request metrics help you monitor Amazon S3 requests to quickly identify and act on operational issues.

By default, request metrics are available at the bucket level. However, you can define a filter for request metrics using a shared prefix, object tags, or an access point. When you create an access point filter, the request metrics configuration includes requests to the access point that you specify. You can receive metrics, set alarms, and access dashboards to view real-time operations performed through this access point.

You must opt in to request metrics by configuring them in the console or by using the Amazon S3 API. Request metrics are available at 1-minute intervals after some latency for processing. Request metrics are billed at the same rate as CloudWatch custom metrics. For more information, see [Amazon CloudWatch pricing](#).

To create a request metrics configuration that filters by access point, see [Creating a metrics configuration that filters by prefix, object tag, or access point \(p. 1014\)](#).

Request logs

You can log requests made through access points and requests made to the APIs that manage access points, such as `CreateAccessPoint` and `GetAccessPointPolicy`, by using server access logging and AWS CloudTrail.

CloudTrail log entries for requests made through access points include the access point ARN in the `resources` section of the log.

For example, suppose you have the following configuration:

- A bucket named `DOC-EXAMPLE-BUCKET1` in Region `us-west-2` that contains an object named `my-image.jpg`
- An access point named `my-bucket-ap` that is associated with `DOC-EXAMPLE-BUCKET1`
- An AWS account ID of `123456789012`

The following example shows the `resources` section of a CloudTrail log entry for the preceding configuration:

```
"resources": [
    {"type": "AWS::S3::Object",
     "ARN": "arn:aws:s3:::DOC-EXAMPLE-BUCKET1/my-image.jpg"
    },
    {"accountId": "123456789012",
     "type": "AWS::S3::Bucket",
     "ARN": "arn:aws:s3:::DOC-EXAMPLE-BUCKET1"
    },
    {"accountId": "123456789012",
     "type": "AWS::S3::AccessPoint",
     "ARN": "arn:aws:s3:us-west-2:123456789012:accesspoint/my-bucket-ap"
    }
]
```

For more information about S3 Server Access Logs, see [Logging requests using server access logging \(p. 978\)](#). For more information about AWS CloudTrail, see [What is AWS CloudTrail?](#) in the [AWS CloudTrail User Guide](#).

Using Amazon S3 access points with the Amazon S3 console

This section explains how to manage and use your Amazon S3 access points using the AWS Management Console. Before you begin, navigate to the detail page for the access point you want to manage or use, as described in the following procedure.

Topics

- [Listing access points for your account \(p. 312\)](#)
- [Listing access points for a bucket \(p. 312\)](#)
- [Viewing configuration details for an access point \(p. 313\)](#)
- [Using an access point \(p. 313\)](#)
- [Viewing block public access settings for an access point \(p. 313\)](#)
- [Editing an access point policy \(p. 313\)](#)
- [Deleting an access point \(p. 314\)](#)

[Listing access points for your account](#)

1. Sign in to the AWS Management Console and open the Amazon S3 console at <https://console.aws.amazon.com/s3/>.
2. In the navigation pane on the left side of the console, choose **access points**.
3. On the **access points** page, under **access points**, select the AWS Region that contains the access points you want to list.
4. (Optional) Search for access points by name by entering a name into the text field next to the Region dropdown menu.
5. Choose the name of the access point you want to manage or use.

[Listing access points for a bucket](#)

To list all access points for a single bucket

1. Sign in to the AWS Management Console and open the Amazon S3 console at <https://console.aws.amazon.com/s3/>.

2. In the navigation pane on the left side of the console, choose **Buckets**.
3. On the **Buckets** page, select the name of the bucket whose access points you want to list.
4. On the bucket detail page, choose the **access points** tab.
5. Choose the name of the access point you want to manage or use.

Viewing configuration details for an access point

1. Navigate to the access point detail page for the access point whose details you want to view, as described in [Listing access points for your account \(p. 312\)](#).
2. Under **access point overview**, view configuration details and properties for the selected access point.

Using an access point

1. Navigate to the access point detail page for the access point you want to use, as described in [Listing access points for your account \(p. 312\)](#).
2. Under the **Objects** tab, choose the name of an object or objects that you want to access through the access point. On the object operation pages, the console displays a label above the name of your bucket that shows the access point that you're currently using. While you're using the access point, you can only perform the object operations that are allowed by the access point permissions.

Note

- The console view always shows all objects in the bucket. Using an access point as described in this procedure restricts the operations you can perform on those objects, but not whether you can see that they exist in the bucket.
- The S3 Management Console doesn't support using virtual private cloud (VPC) access points to access bucket resources. To access bucket resources from a VPC access point, use the AWS CLI, AWS SDKs, or Amazon S3 REST APIs.

Viewing block public access settings for an access point

1. Navigate to the access point detail page for the access point whose settings you want to view, as described in [Listing access points for your account \(p. 312\)](#).
2. Choose **Permissions**.
3. Under **access point policy**, review the access point's Block Public Access settings.

Note

You can't change the Block Public Access settings for an access point after the access point is created.

Editing an access point policy

1. Navigate to the access point detail page for the access point whose policy you want to edit, as described in [Listing access points for your account \(p. 312\)](#).
2. Choose **Permissions**.
3. Under **access point policy**, choose **Edit**.
4. Enter the access point policy in the text field. The console automatically displays the Amazon Resource Name (ARN) for the access point, which you can use in the policy.

Deleting an access point

1. Navigate to the list of access points for your account or for a specific bucket, as described in [Listing access points for your account \(p. 312\)](#).
2. Select the option button next to the name of the access point that you want to delete.
3. Choose **Delete**.
4. Confirm that you want to delete your access point by entering its name in the text field that appears, and choose **Delete**.

Using a bucket-style alias for your access point

When you create an access point, Amazon S3 automatically generates an alias that you can use instead of an Amazon S3 bucket name for data access. You can use this access point alias instead of an Amazon Resource Name (ARN) for any access point data plane operation. For a list of these operations, see [Access point compatibility with AWS services \(p. 315\)](#).

The following shows an example ARN and access point alias for an access point named *my-access-point*.

- **ARN** — `arn:aws:s3:region:account-id:accesspoint/my-access-point`
- **Access point alias** — `my-access-point-hrzrlukc5m36ft7okagg1f3gmwluquuse1b-s3alias`

For more information about ARNs, see [Amazon Resource Names \(ARNs\)](#) in the *AWS General Reference*.

Access point alias names

An access point alias name is created within the same namespace as an Amazon S3 bucket. This alias name is automatically generated and cannot be changed. An access point alias name meets all the requirements of a valid Amazon S3 bucket name and consists of the following parts:

`[Access point prefix]-[Metadata]-s3alias`

Note

The `-s3alias` suffix is reserved for access point alias names and can't be used for bucket or access point names. For more information about Amazon S3 bucket naming rules, see [Bucket naming rules \(p. 118\)](#).

Access point alias use cases and limitations

When adopting access points, you can use access point alias names without requiring extensive code changes.

When you create an access point, Amazon S3 automatically generates an access point alias name, as shown in the following example.

```
aws s3control create-access-point --bucket DOC-EXAMPLE-BUCKET1 --name my-access-point --account-id 111122223333
{
    "AccessPointArn": "arn:aws:s3:region:111122223333:accesspoint/my-access-point",
    "Alias": "my-access-point-aqfqprnstn7aefdfbarligizwgyfouse1a-s3alias"
}
```

You can use this access point alias name instead of an Amazon S3 bucket name in any data plane operation. For a list of these operations, see [Access point compatibility with AWS services \(p. 315\)](#).

```
aws s3api get-object --bucket my-access-point-aqfqprnstaefdfbarligizwgyfouse1a-s3alias --  
key dir/my_data.rtf my_data.rtf  
{  
    "AcceptRanges": "bytes",  
    "LastModified": "2020-01-08T22:16:28+00:00",  
    "ContentLength": 910,  
    "ETag": "\"00751974dc146b76404bb7290f8f51bb\"",  
    "VersionId": "null",  
    "ContentType": "text/rtf",  
    "Metadata": {}  
}
```

Limitations

- Aliases cannot be configured by customers.
- Aliases cannot be deleted or modified or disabled on an Access Point.
- You can use this access point alias name instead of an Amazon S3 bucket name in some data plane operation. For a list of these operations, see [Access point compatibility with S3 operations \(p. 315\)](#).
- You can't use an access point alias name for Amazon S3 control plane operations. For a list of Amazon S3 control plane operations, see [Amazon S3 Control](#) in the *Amazon Simple Storage Service API Reference*.
- Aliases cannot be used in IAM policies.
- Aliases cannot be used as a logging destination for S3 server access logs.
- Aliases cannot be used as a logging destination for AWS CloudTrail logs.
- Amazon SageMaker GroundTruth does not support access point alias.

Using access points with compatible Amazon S3 operations

The following examples demonstrate how to use access points with compatible operations in Amazon S3.

Topics

- [Access point compatibility with AWS services \(p. 315\)](#)
- [Access point compatibility with S3 operations \(p. 315\)](#)
- [Request an object through an access point \(p. 316\)](#)
- [Upload an object through an access point alias \(p. 317\)](#)
- [Delete an object through an access point \(p. 317\)](#)
- [List objects through an access point alias \(p. 317\)](#)
- [Add a tag set to an object through an access point \(p. 317\)](#)
- [Grant access permissions through an access point using an ACL \(p. 317\)](#)

Access point compatibility with AWS services

Amazon S3 access points aliases allow any application that requires an S3 bucket name to easily use an access point. You can use S3 access point aliases anywhere you use S3 bucket names to access data in S3.

Access point compatibility with S3 operations

You can use access points to access a bucket using the following subset of Amazon S3 APIs. All the operations listed below can accept either access point ARNs or access point aliases:

S3 operations

- [AbortMultipartUpload](#)
- [CompleteMultipartUpload](#)
- [CopyObject](#) (same-region copies only)
- [CreateMultipartUpload](#)
- [DeleteObject](#)
- [DeleteObjectTagging](#)
- [GetBucketAcl](#)
- [GetBucketCors](#)
- [GetBucketLocation](#)
- [GetBucketNotificationConfiguration](#)
- [GetBucketPolicy](#)
- [GetObject](#)
- [GetObjectAcl](#)
- [GetObjectLegalHold](#)
- [GetObjectRetention](#)
- [GetObjectTagging](#)
- [HeadBucket](#)
- [HeadObject](#)
- [ListMultipartUploads](#)
- [ListObjects](#)
- [ListObjectsV2](#)
- [ListParts](#)
- [Presign](#)
- [PutObject](#)
- [PutObjectLegalHold](#)
- [PutObjectRetention](#)
- [PutObjectAcl](#)
- [PutObjectTagging](#)
- [RestoreObject](#)
- [UploadPart](#)
- [UploadPartCopy](#) (same-region copies only)

Request an object through an access point

The following example requests the object `my-image.jpg` through the access point `prod` owned by account ID `123456789012` in Region `us-west-2`, and saves the downloaded file as `download.jpg`.

AWS CLI

```
aws s3api get-object --key my-image.jpg --bucket arn:aws:s3:us-west-2:123456789012:accesspoint/prod download.jpg
```

Upload an object through an access point alias

The following example uploads the object `my-image.jpg` through the access point alias `my-access-point-hrzrlukc5m36ft7okagglf3gmwluquse1b-s3alias` owned by account ID `123456789012` in Region `us-west-2`.

AWS CLI

```
aws s3api put-object --bucket my-access-point-hrzrlukc5m36ft7okagglf3gmwluquse1b-s3alias --key my-image.jpg --body my-image.jpg
```

Delete an object through an access point

The following example deletes the object `my-image.jpg` through the access point `prod` owned by account ID `123456789012` in Region `us-west-2`.

AWS CLI

```
aws s3api delete-object --bucket arn:aws:s3:us-west-2:123456789012:accesspoint/prod --key my-image.jpg
```

List objects through an access point alias

The following example lists objects through the access point alias `my-access-point-hrzrlukc5m36ft7okagglf3gmwluquse1b-s3alias` owned by account ID `123456789012` in Region `us-west-2`.

AWS CLI

```
aws s3api list-objects-v2 --bucket my-access-point-hrzrlukc5m36ft7okagglf3gmwluquse1b-s3alias
```

Add a tag set to an object through an access point

The following example adds a tag set to the existing object `my-image.jpg` through the access point `prod` owned by account ID `123456789012` in Region `us-west-2`.

AWS CLI

```
aws s3api put-object-tagging --bucket arn:aws:s3:us-west-2:123456789012:accesspoint/prod --key my-image.jpg --tagging TagSet=[{Key="finance",Value="true"}]
```

Grant access permissions through an access point using an ACL

The following example applies an ACL to an existing object `my-image.jpg` through the access point `prod` owned by account ID `123456789012` in Region `us-west-2`.

AWS CLI

```
aws s3api put-object-acl --bucket arn:aws:s3:us-west-2:123456789012:accesspoint/prod --key my-image.jpg --acl private
```

Access points restrictions and limitations

Amazon S3 access points have the following restrictions and limitations:

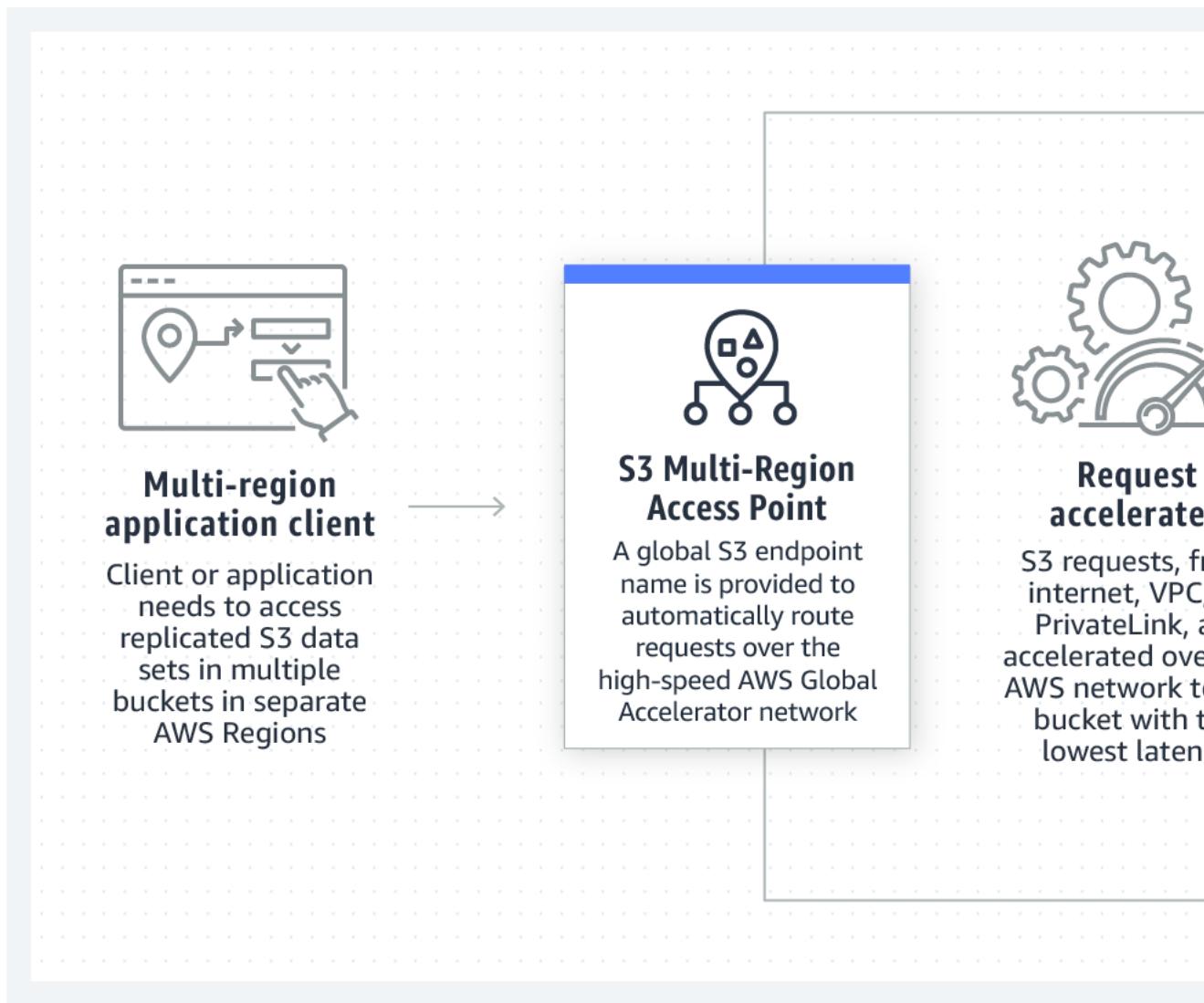
- You can only create access points for buckets that you own.
- Each access point is associated with exactly one bucket, which you must specify when you create the access point. After you create an access point, you can't associate it with a different bucket. However, you can delete an access point and then create another one with the same name associated with a different bucket.
- Access point names must meet certain conditions. For more information about naming access points, see [Rules for naming Amazon S3 access points \(p. 306\)](#).
- After you create an access point, you can't change its virtual private cloud (VPC) configuration.
- Access point policies are limited to 20 KB in size.
- You can create a maximum of 10,000 access points per AWS account per Region. If you need more than 10,000 access points for a single account in a single Region, you can request a service quota increase. For more information about service quotas and requesting an increase, see [AWS Service Quotas](#) in the [AWS General Reference](#).
- You can't use an access point as a destination for S3 Replication. For more information about replication, see [Replicating objects \(p. 753\)](#).
- You can only address access points using virtual-host-style URLs. For more information about virtual-host-style addressing, see [Methods for accessing a bucket \(p. 125\)](#).
- APIs that control access point functionality (for example, `PutAccessPoint` and `GetAccessPointPolicy`) don't support cross-account calls.
- You must use AWS Signature Version 4 when making requests to an access point using the REST APIs. For more information about authenticating requests, see [Authenticating Requests \(AWS Signature Version 4\)](#) in the [Amazon Simple Storage Service API Reference](#).
- Access points only support access over HTTPS.
- Access points don't support anonymous access.

Multi-Region Access Points in Amazon S3

Amazon S3 Multi-Region Access Points provide a global endpoint that applications can use to fulfill requests from S3 buckets located in multiple AWS Regions. You can use Multi-Region Access Points to build multi-Region applications with the same simple architecture used in a single Region, and then run those applications anywhere in the world. Instead of sending requests over the congested public internet, Multi-Region Access Points provide built-in network resilience with acceleration of internet-based requests to Amazon S3. Application requests made to a Multi-Region Access Point global endpoint use [AWS Global Accelerator](#) to automatically route over the AWS global network to the S3 bucket with the lowest network latency.

When you create a Multi-Region Access Point, you specify a set of Regions where you want to store data to be served through that Multi-Region Access Point. You can use [S3 Cross-Region Replication \(CRR\)](#) to synchronize data among buckets in those Regions. You can then request or write data through the Multi-Region Access Point global endpoint. Amazon S3 automatically serves the request to the replicated dataset from the available Region over the AWS global network with the lowest latency. Multi-Region Access Points are also compatible with applications running in Amazon virtual private clouds (VPCs), including those using [AWS PrivateLink for Amazon S3 \(p. 385\)](#).

The following is a graphical representation of a Multi-Region Access Point and how it routes requests to buckets.



Topics

- [Creating Multi-Region Access Points \(p. 320\)](#)
- [Making requests using a Multi-Region Access Point \(p. 326\)](#)
- [Managing Multi-Region Access Points \(p. 331\)](#)
- [Monitoring and logging requests made through a Multi-Region Access Point to underlying resources \(p. 332\)](#)
- [Multi-Region Access Point restrictions and limitations \(p. 334\)](#)

Creating Multi-Region Access Points

To create a Multi-Region Access Point in Amazon S3, you specify the name, choose one bucket in each AWS Region that you want to serve requests for the Multi-Region Access Point, and configure the Amazon S3 Block Public Access settings for the Multi-Region Access Point. You provide this information in a create request, which Amazon S3 processes asynchronously. Amazon S3 provides a token that you can use to monitor the status of the asynchronous creation request.

Make sure to resolve security warnings, errors, general warnings, and suggestions from AWS Identity and Access Management Access Analyzer before you save your policy. IAM Access Analyzer runs policy checks to validate your policy against IAM [policy grammar](#) and [best practices](#). These checks generate findings and provide actionable recommendations to help you author policies that are functional and conform to security best practices. To learn more about validating policies using IAM Access Analyzer, see [IAM Access Analyzer policy validation](#) in the *IAM User Guide*. To view a list of the warnings, errors, and suggestions that are returned by IAM Access Analyzer, see [IAM Access Analyzer policy check reference](#).

When you use the API, the request to create a Multi-Region Access Point is asynchronous. When you submit a request to create a Multi-Region Access Point, Amazon S3 synchronously authorizes the request. It then immediately returns a token that you can use to track the progress of the creation request. For more information about tracking asynchronous requests to create and manage Multi-Region Access Points, see [Managing Multi-Region Access Points \(p. 331\)](#).

After you create the Multi-Region Access Point, you can create an access control policy for it. Each Multi-Region Access Point can have an associated policy. A Multi-Region Access Point policy is a resource-based policy that allows you to limit the use of the Multi-Region Access Point by resource, user, or other conditions.

Note

For an application or user to be able to access an object through a Multi-Region Access Point, both the access policy for the Multi-Region Access Point and the access policy for the underlying buckets that contain the object must permit the request. When the two policies are different, the more restrictive policy takes precedence.

Using a bucket with a Multi-Region Access Point does not change the bucket's behavior when the bucket is accessed through the existing bucket name or an Amazon Resource Name (ARN). All existing operations against the bucket continue to work as before. Restrictions that you include in a Multi-Region Access Point policy apply only to requests that are made through the Multi-Region Access Point.

You can update the policy for a Multi-Region Access Point after creating it, but you can't delete the policy. The closest possible approximation to deleting a policy is to update the Multi-Region Access Point policy to deny all permissions.

Topics

- [Rules for naming Amazon S3 Multi-Region Access Points \(p. 321\)](#)
- [Rules for choosing buckets for Amazon S3 Multi-Region Access Points \(p. 322\)](#)
- [Blocking public access with Amazon S3 Multi-Region Access Points \(p. 323\)](#)
- [Creating Amazon S3 Multi-Region Access Points \(p. 323\)](#)
- [Configuring a Multi-Region Access Point for use with AWS PrivateLink \(p. 324\)](#)

Rules for naming Amazon S3 Multi-Region Access Points

When you create a Multi-Region Access Point, you give it a name, which is a string that you choose. You can't change the name of the Multi-Region Access Point after it is created. The name must be unique in your AWS account, and it must conform to the naming requirements listed in [Multi-Region Access Point restrictions and limitations \(p. 334\)](#). To help you identify the Multi-Region Access Point, use a name that is meaningful to you, to your organization, or that reflects the scenario.

You use this name when invoking Multi-Region Access Point management operations, such as `GetMultiRegionAccessPoint` and `PutMultiRegionAccessPointPolicy`. The name is not used to send requests to the Multi-Region Access Point, and it doesn't need to be exposed to clients who make requests using the Multi-Region Access Point.

When Amazon S3 creates a Multi-Region Access Point, it automatically assigns an alias to it. This alias is a unique alphanumeric string that ends in `.mrap`. The alias is used to construct the hostname and the Amazon Resource Name (ARN) for a Multi-Region Access Point. The fully qualified name is also based on the alias for the Multi-Region Access Point.

You can't determine the name of a Multi-Region Access Point from its alias, so you can disclose an alias without risk of exposing the name, purpose, or owner of the Multi-Region Access Point. Amazon S3 selects the alias for each new Multi-Region Access Point, and the alias can't be changed. For more information about addressing a Multi-Region Access Point, see [Making requests using a Multi-Region Access Point \(p. 326\)](#).

Multi-Region Access Point aliases are unique throughout time and aren't based on the name or configuration of a Multi-Region Access Point. If you create a Multi-Region Access Point, and then delete it and create another one with the same name and configuration, the second Multi-Region Access Point will have a different alias than the first. New Multi-Region Access Points can never have the same alias as a previous Multi-Region Access Point.

Rules for choosing buckets for Amazon S3 Multi-Region Access Points

Each Multi-Region Access Point is associated with the Regions where you want to fulfill requests. The Multi-Region Access Point must be associated with exactly one bucket in each of those Regions. You specify the name of each bucket in the request to create the Multi-Region Access Point. Each bucket that supports the Multi-Region Access Point must be owned by the same AWS account that owns the Multi-Region Access Point.

A single bucket can be used by multiple Multi-Region Access Points.

Important

- You can specify the buckets that are associated with a Multi-Region Access Point only at the time that you create it. After it is created, you can't add, modify, or remove buckets from the Multi-Region Access Point configuration. To change the buckets, you must delete the entire Multi-Region Access Point and create a new one.
- You can't delete a bucket that is part of a Multi-Region Access Point. If you want to delete a bucket attached to a Multi-Region Access Point, delete the Multi-Region Access Point first.
- The AWS account that owns the Multi-Region Access Point must also own the associated buckets. For more information about using permissions with Multi-Region Access Points, see [Multi-Region Access Point permissions \(p. 328\)](#).
- Not all Regions support Multi-Region Access Points. To see the list of supported Regions, see [Multi-Region Access Point restrictions and limitations \(p. 334\)](#).

You can create replication rules to synchronize data between buckets. These rules enable you to automatically copy data from source buckets to destination buckets. Having buckets connected to a Multi-Region Access Point does not affect how replication works. Configuring replication with Multi-Region Access Points is described in a later section.

It is important to realize that when you make a request to a Multi-Region Access Point, the Multi-Region Access Point does not make any considerations about which bucket can fulfill the request. This is why replication is recommended. Otherwise, one of the buckets in the Multi-Region Access Point might have the necessary data, but there's no way to guarantee it will receive the request. For more information, see [Configuring bucket replication for use with Multi-Region Access Points \(p. 330\)](#).

Blocking public access with Amazon S3 Multi-Region Access Points

Each Multi-Region Access Point has distinct settings for Amazon S3 Block Public Access. These settings operate in conjunction with the Block Public Access settings for the buckets that underly the Multi-Region Access Point and for the AWS account that owns both the Multi-Region Access Point and the underlying buckets.

When Amazon S3 authorizes a request, it applies the most restrictive combination of these settings. If the Block Public Access settings for any of these resources (the Multi-Region Access Point, the underlying bucket, or the owner account) block access for the requested action or resource, Amazon S3 rejects the request.

We recommend that you enable all Block Public Access settings unless you have a specific need to disable any of them. By default, all Block Public Access settings are enabled for a Multi-Region Access Point. Be aware that if Block Public Access is enabled, the Multi-Region Access Point will not be able to accept internet-based requests.

Important

Amazon S3 doesn't currently support changing the Block Public Access settings for a Multi-Region Access Point after it has been created.

For more information about Amazon S3 Block Public Access, see [Blocking public access to your Amazon S3 storage \(p. 584\)](#).

Creating Amazon S3 Multi-Region Access Points

The following example demonstrates how to create a Multi-Region Access Point using the AWS Management Console.

Using the S3 console

To create a Multi-Region Access Point

1. Sign in to the AWS Management Console and open the Amazon S3 console at <https://console.aws.amazon.com/s3/>.
2. In the navigation pane, choose **Multi-Region Access Points**.
3. In the **Multi-Region Access Point name** field, supply a name for the Multi-Region Access Point.
4. To select the buckets that will be associated with this Multi-Region Access Point, choose **Add buckets**.

To create a new bucket, choose **Create bucket**. After creating the bucket, choose **Add buckets** to add the bucket to the Multi-Region Access Point.

For more information about creating buckets, see [Creating a bucket \(p. 119\)](#).

5. Under **Block Public Access settings for this Multi-Region Access Point**, select the Block Public Access settings that you want to apply to the Multi-Region Access Point. By default, all Block Public Access settings are enabled for new Multi-Region Access Points. We recommend that you leave all settings enabled unless you know that you have a specific need to disable any of them.

Note

Amazon S3 currently doesn't support changing a Multi-Region Access Point's Block Public Access settings after the Multi-Region Access Point has been created.

6. Choose **Create Multi-Region Access Point**.

Using the AWS CLI

You can use the AWS CLI to create a Multi-Region Access Point. Remember that when you create the Multi-Region Access Point, you need to provide all the buckets it will support. There is no option to add buckets to the Multi-Region Access Point after it has been created.

The following example creates a Multi-Region Access Point with two buckets using the AWS CLI.

```
aws s3control create-multi-region-access-point --account-id 111122223333 --details '{  
    "Name": "simple-multiregionaccesspoint-with-two-regions",  
    "PublicAccessBlock": {  
        "BlockPublicAcls": true,  
        "IgnorePublicAcls": true,  
        "BlockPublicPolicy": true,  
        "RestrictPublicBuckets": true  
    },  
    "Regions": [  
        { "Bucket": "DOC-EXAMPLE-BUCKET1" },  
        { "Bucket": "DOC-EXAMPLE-BUCKET2" }  
    ]  
' --region us-west-2
```

Topics

- [Configuring a Multi-Region Access Point for use with AWS PrivateLink \(p. 324\)](#)

Configuring a Multi-Region Access Point for use with AWS PrivateLink

AWS PrivateLink provides you with private connectivity to Amazon S3 using private IP addresses in your virtual private cloud (VPC). You can provision one or more interface endpoints inside your VPC to connect to Amazon S3 Multi-Region Access Points.

You can create **com.amazonaws.s3-global.accesspoint** endpoints for Multi-Region Access Points through the AWS Management Console, AWS CLI, or AWS SDKs. To learn more about how to configure an interface endpoint for Multi-Region Access Point, see [Interface VPC endpoints](#) in the *VPC User Guide*.

To make requests to a Multi-Region Access Point via interface endpoints, follow these steps to configure the VPC and the Multi-Region Access Point.

To configure a Multi-Region Access Point to use with AWS PrivateLink

1. Create or have an appropriate VPC endpoint that can connect to Multi-Region Access Points. For more information about creating VPC endpoints, see [Interface VPC endpoints](#) in the *VPC User Guide*.

Important

Make sure to create a **com.amazonaws.s3-global.accesspoint** endpoint. Other endpoint types cannot access Multi-Region Access Points.

After this VPC endpoint is created, all Multi-Region Access Point requests in the VPC route through this endpoint if you have private DNS enabled for the endpoint. This is enabled by default.

2. If the Multi-Region Access Point policy does not support connections from VPC endpoints, you will need to update it.
3. Verify that the individual bucket policies will allow access to the users of the Multi-Region Access Point.

Remember that Multi-Region Access Points work by routing requests to buckets, not by fulfilling requests themselves. This is important to remember because the originator of the request must have permissions to the Multi-Region Access Point and be allowed to access the individual buckets in the Multi-Region Access Point. Otherwise, the request might be routed to a bucket where the originator doesn't have permissions to fulfill the request. A Multi-Region Access Point and the buckets must be owned by the same AWS account. However, VPCs from different accounts can use a Multi-Region Access Point if the permissions are configured correctly.

Because of this, the VPC endpoint policy must allow access both to the Multi-Region Access Point and to each underlying bucket that you want to be able to fulfill requests. For example, suppose that you have a Multi-Region Access Point with alias `mfzwi23gnjvgw.mrap`. It is backed by buckets `doc-examplebucket1` and `doc-examplebucket2`, all owned by AWS account `123456789012`. In this case, the following VPCE policy would allow `GetObject` requests from the VPC made to `mfzwi23gnjvgw.mrap` to be fulfilled by either backing bucket.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "Read-buckets-and-MRAP-VPCE-policy",
            "Principal": "*",
            "Action": [
                "s3:GetObject"
            ],
            "Effect": "Allow",
            "Resource": [
                "arn:aws:s3:::doc-examplebucket1/*",
                "arn:aws:s3:::doc-examplebucket2/*",
                "arn:aws:s3:::123456789012:accesspoint/mfzwi23gnjvgw.mrap/object/*"
            ]
        }
    ]
}
```

As mentioned previously, you also must make sure that the Multi-Region Access Point policy is configured to support access through a VPC endpoint. You don't need to specify the VPC endpoint that is requesting access. The following sample policy would grant access to any requestor trying to use the Multi-Region Access Point for the `GetObject` requests.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "Open-read-MRAP-policy",
            "Effect": "Allow",
            "Principal": "*",
            "Action": [
                "s3:GetObject"
            ],
            "Resource": "arn:aws:s3:::123456789012:accesspoint/mfzwi23gnjvgw.mrap/object/*"
        }
    ]
}
```

And of course, the individual buckets would each need a policy to support access from requests submitted through VPC endpoint. The following example policy grants read access to any anonymous users, which would include requests made through the VPC endpoint.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {

```

```
        "Sid": "Public-read",
        "Effect": "Allow",
        "Principal": "*",
        "Action": "s3:GetObject",
        "Resource": [
            "arn:aws:s3:::doc-examplebucket1",
            "arn:aws:s3:::doc-examplebucket2/*"
        ]
    }
}
```

For more information about editing a VPCE policy, see [Control access to services with VPC endpoints](#) in the *VPC User Guide*.

Removing access to a Multi-Region Access Point from a VPC endpoint

If you own a Multi-Region Access Point and want to remove access to it from an interface endpoint, you must supply a new access policy to the Multi-Region Access Point that prevents access for requests coming through VPC endpoints. Keep in mind that if the buckets in your Multi-Region Access Point support requests through VPC endpoints, they will continue to support these requests. If you want to prevent that support, you must also update the policies for the buckets. Supplying a new access policy to the Multi-Region Access Point only prevents access to the Multi-Region Access Point.

Note

You can't delete an access policy for a Multi-Region Access Point. To remove access to a Multi-Region Access Point, you must provide a new access policy with the modified access that you want.

As an alternative, you could update the bucket policies to prevent requests through VPC endpoints. In this case, the user could still access the Multi-Region Access Point through the VPC endpoint. But if the request is routed to a bucket where the bucket policy prevents access, it would generate an error message.

Making requests using a Multi-Region Access Point

Multi-Region Access Points in Amazon S3 have Amazon Resource Names (ARNs), which you can use to direct requests to them using the AWS SDKs and to identify a Multi-Region Access Point in access control policies. A Multi-Region Access Point ARN doesn't include or disclose its name. For more information about ARNs, see [Amazon Resource Names \(ARNs\)](#) in the *AWS General Reference*.

Multi-Region Access Point ARNs use the format `arn:aws:s3:::<account-id>:accesspoint/<MRAP_alias>`. The following are a few examples.

- `arn:aws:s3:::123456789012:accesspoint/mfzwi23gnjvgw.mrap` represents the Multi-Region Access Point with the alias `mfzwi23gnjvgw.mrap`, owned by AWS account 123456789012.
- `arn:aws:s3:::123456789012:accesspoint/*` represents all Multi-Region Access Points under account 123456789012. This ARN matches all Multi-Region Access Points for account 123456789012, but doesn't match any Regional access points because the ARN doesn't include an AWS Region. In contrast, the ARN `arn:aws:s3:us-west-2:123456789012:accesspoint/*` matches all Regional access points in Region `us-west-2` for account 123456789012, but doesn't match any Multi-Region Access Points.

ARNs for objects that are accessed through a Multi-Region Access Point use the format `arn:aws:s3:::<account_id>:accesspoint/<MRAP_alias>/object/<key>`. As with Multi-Region

Access Point ARNs, the ARNs for objects that are accessed through Multi-Region Access Points don't include an AWS Region. Here are some examples.

- `arn:aws:s3:::123456789012:accesspoint/mfzwi23gnjvgw.mrap/object/unit-01` represents the object unit-01, accessed through the Multi-Region Access Point with the alias `mfzwi23gnjvgw.mrap`, owned by account 123456789012.
- `arn:aws:s3:::123456789012:accesspoint/mfzwi23gnjvgw.mrap/object/*` represents all objects that can be accessed through the Multi-Region Access Point with alias `mfzwi23gnjvgw.mrap`, in account 123456789012.
- `arn:aws:s3:::123456789012:accesspoint/mfzwi23gnjvgw.mrap/object/unit-01/finance/*` represents all objects that can be accessed under prefix `unit-01/finance/` for the Multi-Region Access Point with alias `mfzwi23gnjvgw.mrap`, in account 123456789012.

Multi-Region Access Point hostnames

You can access data in Amazon S3 through a Multi-Region Access Point using the hostname of the Multi-Region Access Point. Requests can be directed to this hostname from the public internet or from a virtual private cloud (VPC) if you have configured one or more internet gateways for the Multi-Region Access Point. For more information about creating VPC interface endpoints to use with Multi-Region Access Points, see [Configuring a Multi-Region Access Point for use with AWS PrivateLink \(p. 324\)](#).

You can also make requests through a Multi-Region Access Point from a VPC using AWS PrivateLink if you have configured a VPC endpoint. Be aware that with requests to a Multi-Region Access Point using AWS PrivateLink, you cannot directly use an endpoint-specific Regional DNS ending with `<Region>.vpce.amazonaws.com`. This hostname will not have a cert associated with it so it cannot be used directly. You can still use the public DNS name of the VPC endpoint as a CNAME or ALIAS target. Alternatively, you can enable private DNS on the endpoint and use the standard Multi-Region Access Point `<MRAP_alias>.accesspoint.s3-global.amazonaws.com` DNS names as described below.

When you use the REST APIs for Amazon S3 data operations (for example, `GetObject`) through a Multi-Region Access Point, the hostname for the request is `<MRAP_alias>.accesspoint.s3-global.amazonaws.com`. For example, to make a `GetObject` request through the Multi-Region Access Point with alias `mfzwi23gnjvgw.mrap`, make a request to the hostname `mfzwi23gnjvgw.mrap.accesspoint.s3-global.amazonaws.com`. Note the `s3-global` portion of the hostname that indicates this hostname is not for a specific Region.

Making requests through a Multi-Region Access Point is similar to making requests through a single-Region access point. It is important to be aware of the following differences:

- Multi-Region Access Point ARNs don't include an AWS Region. They follow the format `arn:aws:s3:::<account-id>:accesspoint/<MRAP_alias>`.
- For requests made through the REST APIs (this does not require the use of an ARN), Multi-Region Access Points use a different endpoint scheme. The scheme is `<MRAP_alias>.accesspoint.s3-global.amazonaws.com`—for example, `mfzwi23gnjvgw.mrap.accesspoint.s3-global.amazonaws.com`. Note the differences compared to a single-Region access point:
 - Multi-Region Access Point hostnames use their alias, not the Multi-Region Access Point name.
 - Multi-Region Access Point hostnames don't include the owner's AWS account ID.
 - Multi-Region Access Point hostnames don't include an AWS Region.
 - Multi-Region Access Point hostnames include `s3-global.amazonaws.com` instead of `s3.amazonaws.com`.
- Requests must be signed using Signature Version 4A (SigV4A). When you use the AWS SDK, the SDK automatically converts a SigV4 signature to SigV4A. For more information about SigV4A, see [Signing AWS API requests](#) in the [AWS General Reference](#).

Multi-Region Access Points and Amazon S3 Transfer Acceleration

Amazon S3 Transfer Acceleration is a feature that enables fast transfer of data to buckets. It is configured on the individual bucket level and you can use it to transfer objects faster to buckets. To read more about Transfer Acceleration, see [Configuring fast, secure file transfers using Amazon S3 Transfer Acceleration \(p. 136\)](#).

When dealing with Multi-Region Access Points, it is important to know that Multi-Region Access Points use a similar accelerated transfer mechanism as Transfer Acceleration for sending large objects over the AWS network. Because of this, you don't need to perform any special configuration or handling to gain the benefits of the faster transfer rates when sending requests through a Multi-Region Access Point. This increased performance is automatically incorporated into the Multi-Region Access Point.

Topics

- [Multi-Region Access Point permissions \(p. 328\)](#)
- [Multi-Region Access Point request routing \(p. 329\)](#)
- [Configuring bucket replication for use with Multi-Region Access Points \(p. 330\)](#)
- [Multi-Region Access Point supported operations \(p. 331\)](#)

Multi-Region Access Point permissions

When you make a request through a Multi-Region Access Point, Amazon S3 authorizes the request against the Multi-Region Access Point and against the underlying bucket that the request is routed to. Thus, for a request to succeed, both the Multi-Region Access Point and at least one underlying bucket must permit the operation.

For example, suppose that you make a `GetObject` request through a Multi-Region Access Point using a user called `AppDataReader` in your AWS account. To help ensure that the request won't be denied, user `AppDataReader` must be granted the `s3:GetObject` permission by the Multi-Region Access Point and by each bucket underlying the Multi-Region Access Point. `AppDataReader` won't be able to retrieve data from any bucket that doesn't grant this permission.

In general, underlying buckets still have individual S3 Block Public Access settings, policies, and access control lists (ACLs, including object ACLs) that remain in effect in all cases.

Managing public access to a Multi-Region Access Point

Multi-Region Access Points support independent Block Public Access settings for each Multi-Region Access Point. When you create a Multi-Region Access Point, you can specify the Block Public Access settings that apply to that Multi-Region Access Point.

For any request that is made through a Multi-Region Access Point, Amazon S3 evaluates the Block Public Access settings for that Multi-Region Access Point, the underlying buckets, and the account that owns both the Multi-Region Access Point and underlying buckets. If any of these settings indicate that the request should be blocked, Amazon S3 rejects the request. For more information about the Amazon S3 Block Public Access feature, see [Blocking public access to your Amazon S3 storage \(p. 584\)](#).

Important

All Block Public Access settings are enabled by default for Multi-Region Access Points. You must explicitly turn off any settings that you don't want to apply to a Multi-Region Access Point. Amazon S3 doesn't currently support changing the Block Public Access settings for a Multi-Region Access Point after it has been created.

Delegating access control to Multi-Region Access Point policies

You can delegate access control for a bucket to the Multi-Region Access Point access policy. The following example bucket policy allows full access to all access points owned by the bucket owner's account. This means that all access to this bucket is controlled by the policies that are attached to its access points. We recommend configuring your buckets this way for all use cases that don't require direct access to the bucket.

```
{  
    "Version": "2012-10-17",  
    "Statement" : [  
        {  
            "Effect": "Allow",  
            "Principal" : { "AWS": "*" },  
            "Action" : "*",  
            "Resource" : [ "Bucket ARN", "Bucket ARN/*"],  
            "Condition": {  
                "StringEquals" : { "s3:DataAccessPointAccount" : "Bucket owner's account ID" }  
            }  
        }]  
}
```

The following example bucket policy delegates access control to any of the bucket's Multi-Region Access Points. If you want to delegate access to specific Multi-Region Access Points, you can use the `s3:DataAccessPointArn` condition key instead.

```
{  
    "Version": "2012-10-17",  
    "Statement" : [  
        {  
            "Effect": "Allow",  
            "Principal" : { "AWS": "*" },  
            "Action" : "*",  
            "Resource" : [ "Bucket ARN", "Bucket ARN/*"],  
            "Condition": {  
                "StringEquals" : { "s3:DataAccessPointArn" : "MRAP ARN" }  
            }  
        }]  
}
```

Multi-Region Access Point request routing

When you make a request through a Multi-Region Access Point, Amazon S3 determines which of the buckets that are associated with the Multi-Region Access Point can respond to its request with the lowest latency. Amazon S3 then directs the request to that bucket, regardless of the AWS Region it is located in.

After the Multi-Region Access Point routes the request to the lowest-latency bucket, Amazon S3 processes the request as if you made it directly to that bucket. Multi-Region Access Points are not aware of the data contents of an Amazon S3 bucket. If you make `GET` requests to a Multi-Region Access Point, you can configure S3 Cross-Region Replication to create consistent datasets in your Amazon S3 buckets behind a Multi-Region Access Point. Then any bucket can fulfill the `GET` request successfully.

Amazon S3 directs Multi-Region Access Point requests according to the following rules:

- Amazon S3 optimizes requests to be fulfilled with the lowest possible latency. It looks at the buckets supported by the Multi-Region Access Point and relays the request to the bucket that has the lowest latency.
- If the request specifies an existing resource (for example, `GetObject`), Amazon S3 does *not* consider the name of the object when fulfilling the request. This means that an object might exist in one bucket

in the Multi-Region Access Point, but your request will be routed to a bucket that does not contain the object. This will result in a 404 error message returned to the client. To ensure that your requests are fulfilled using the specific objects that you want, we recommend that you turn on bucket versioning and include version IDs in your requests. This helps ensure that you have the correct version of the object that you are looking for.

We also recommend that you configure replication for your buckets. This helps resolve the potential issue when the object that you want is in a bucket in the Multi-Region Access Point, but it's not located in the specific bucket that your request was routed to. For more information about configuring replication, see [Configuring bucket replication for use with Multi-Region Access Points \(p. 330\)](#).

- If the request is to create a resource (for example, `PutObject` or `CreateMultipartUpload`), Amazon S3 fulfills the request using the lowest-latency bucket. For example, consider a video company that wants to support video uploads from anywhere in the world to the bucket with the lowest latency. When a user makes a `PUT` request to the Multi-Region Access Point, the object is put into the bucket with the lowest latency. This demonstrates one of the reasons why bi-directional replication can be important. For more information about replication with Multi-Region Access Points, see [Configuring bucket replication for use with Multi-Region Access Points \(p. 330\)](#)

Configuring bucket replication for use with Multi-Region Access Points

When you make a request to a Multi-Region Access Point endpoint, Amazon S3 automatically routes the request to the bucket that responds to the request with the lowest latency. It does not consider the contents of the request when making this decision. If you make a request to `GET` an object, your request might be routed to a bucket that does not have a copy of this object. If that happens, you will receive a 404 error. If you want the Multi-Region Access Point to be able to recover the object regardless of which bucket receives the request, you must configure Amazon S3 Cross-Region Replication.

Consider a Multi-Region Access Point with three buckets:

- A bucket named `my-bucket-usw2` in Region `us-west-2` that contains object `my-image.jpg`.
- A bucket named `my-bucket-aps1` in Region `ap-south-1` that contains object `my-image.jpg`.
- A bucket named `my-bucket-euc1` in Region `eu-central-1` that does not contain an object `my-image.jpg`.

In this situation, if you make a `GetObject` request for the object `my-image.jpg`, the success of that request depends upon which bucket receives your request. Because Amazon S3 does not consider the contents of the request, it might route your `GetObject` request to the `my-bucket-euc1` bucket if that bucket responds with the lowest latency. Even though your object is in a bucket in the Multi-Region Access Point, you will get a 404 error because the individual bucket that received your request did not have the object.

Enabling replication helps mitigate this result. With appropriate replication rules, the `my-image.jpg` image is copied over to the `my-bucket-euc1` bucket, meaning that you retrieve the object if Amazon S3 routes your request to that bucket.

Replication works as normal with buckets that are assigned to a Multi-Region Access Point. Amazon S3 does not perform any special handling with buckets that are in Multi-Region Access Points. Amazon S3 provides 1:N and N:N replication options for flexible synchronization among buckets. For more information about configuring replication in your buckets, see [Setting up replication \(p. 758\)](#).

We do have a few recommendations for you if you want to have the greatest replication performance when working with Multi-Region Access Points. First, we recommend configuring S3 Replication Time Control (S3 RTC), but be aware that comes with an additional cost. For more information about S3

Replication Time Control, see [Meeting compliance requirements using S3 Replication Time Control \(S3 RTC\) \(p. 807\)](#). We also recommend enabling bi-directional replication to support keeping buckets synchronized when a bucket is updated through the Multi-Region Access Point. Finally, you should enable Amazon CloudWatch metrics to monitor the replication events.

Warning

If you use the AWS Management Console to create replication rules from the Multi-Region Access Point console, any pre-existing replication configurations on the specified buckets will be replaced. If you want to add to or modify existing replication configurations instead of replacing them, you can modify the rules using each bucket's replication configuration page in the console, or by using the AWS CLI, SDKs, or REST API. For more information about modifying replication configurations, see [Replication configuration \(p. 759\)](#).

Multi-Region Access Point supported operations

You can use Multi-Region Access Point to access buckets using the following subset of Amazon S3 APIs:

- [AbortMultipartUpload](#)
- [CompleteMultipartUpload](#)
- [CreateMultipartUpload](#)
- [DeleteObject](#)
- [DeleteObjectTagging](#)
- [GetObject](#)
- [GetObjectAcl](#)
- [GetObjectLegalHold](#)
- [GetObjectRetention](#)
- [GetObjectTagging](#)
- [HeadObject](#)
- [ListMultipartUploads](#)
- [ListObjectsV2](#)
- [ListParts](#)
- [PutObject](#)
- [PutObjectAcl](#)
- [PutObjectLegalHold](#)
- [PutObjectRetention](#)
- [PutObjectTagging](#)
- [RestoreObject](#)
- [UploadPart](#)

Note

Multi-Region Access Points do not support the [CopyObject](#) API operation. Instead you will need to perform [CopyObject](#) actions directly between buckets.

Managing Multi-Region Access Points

Amazon S3 provides a set of operations to manage Multi-Region Access Points. Amazon S3 processes some of these operations synchronously and some asynchronously. When you invoke an asynchronous operation, Amazon S3 first synchronously authorizes the requested operation. If authorization is

successful, Amazon S3 returns a token that you can use to track the progress and results of the requested operation.

Note

Requests that are made through the AWS Management Console are always synchronous. The console waits until the request is completed before enabling you to submit another request.

You can view the current status and results of the asynchronous operations using the console, or you can use `DescribeMultiRegionAccessPointOperation` in the AWS CLI, AWS SDKs, or REST API. Amazon S3 provides a tracking token in the response to an asynchronous operation. You include that tracking token as an argument to `DescribeMultiRegionAccessPointOperation`. Amazon S3 then returns the current status and results of the specified operation, including any errors or relevant resource information. Amazon S3 performs `DescribeMultiRegionAccessPointOperation` operations synchronously.

All requests to create or maintain Multi-Region Access Points are routed to the US West (Oregon) Region. This is true regardless of which Region you are in when making the request, or what Regions the Multi-Region Access Point supports. In addition, you must grant the `s3>ListAllMyBuckets` permission to the user, role, or other IAM entity that makes a request to manage a Multi-Region Access Point.

Monitoring and logging requests made through a Multi-Region Access Point to underlying resources

Amazon S3 logs requests made through Multi-Region Access Point and requests made to the APIs that manage them, such as `CreateMultiRegionAccessPoint` and `GetMultiRegionAccessPointPolicy`. Requests made to Amazon S3 through a Multi-Region Access Point appear in your Amazon S3 server access logs and AWS CloudTrail logs with the Multi-Region Access Point hostname. An access point's hostname takes the form <MRAP_alias>.accesspoint.s3-global.amazonaws.com. For example, suppose that you have the following bucket and Multi-Region Access Point configuration:

- A bucket named `my-bucket-usw2` in Region `us-west-2` that contains object `my-image.jpg`.
- A bucket named `my-bucket-aps1` in Region `ap-south-1` that contains object `my-image.jpg`.
- A bucket named `my-bucket-euc1` in Region `eu-central-1` that doesn't contain an object named `my-image.jpg`.
- A Multi-Region Access Point named `my-mrap` with the alias `mfzwi23gnjvgw.mrap` that is configured to fulfill requests from all three buckets.
- Your AWS account ID is `123456789012`.

A request made to retrieve `my-image.jpg` directly through any of the buckets appears in your logs with a hostname of <bucket_name>.s3.<Region>.amazonaws.com.

If you make the request through the Multi-Region Access Point instead, Amazon S3 first determines which of the buckets in the different Regions will fulfill the request with the lowest latency. After Amazon S3 determines which bucket to use to fulfill the request, it sends the request to that bucket and logs the operation using the Multi-Region Access Point hostname. In this example, if Amazon S3 relayed the request to `my-bucket-aps1`, your logs would reflect a successful GET request for `my-image.jpg` from `my-bucket-aps1`, using a hostname of `mfzwi23gnjvgw.mrap.accesspoint.s3-global.amazonaws.com`.

It is important to be aware that Amazon S3 does not perform any consideration about which bucket might be able to fulfill the request. If Amazon S3 determined that the `my-bucket-euc1` bucket would have the lowest latency, your logs would reflect a failed GET request for `my-image.jpg` from `my-bucket-euc1`, using a hostname of `mfzwi23gnjvgw.mrap.accesspoint.s3-`

`global.amazonaws.com`. If the request had been routed to `my-bucket-usw2` instead, your logs would indicate a successful GET request.

For more information about Amazon S3 server access logs, see [Logging requests using server access logging \(p. 978\)](#). For more information about AWS CloudTrail, see [What Is AWS CloudTrail?](#) in the [AWS CloudTrail User Guide](#).

Monitoring and logging requests made to Multi-Region Access Point management APIs

Amazon S3 provides several operations to manage Multi-Region Access Points, such as `CreateMultiRegionAccessPoint` and `GetMultiRegionAccessPointPolicy`. When you make these requests using the AWS CLI, SDKs, or REST API, Amazon S3 processes these requests asynchronously. Provided that you have the appropriate permissions for the request, Amazon S3 returns a token for these requests. You can use this token with `DescribeAsyncOperation` to help you to view the status of ongoing asynchronous operations. S3 processes `DescribeAsyncOperation` requests synchronously. You can use the AWS Management Console, AWS CLI, SDKs, or REST API to view the status of asynchronous requests.

Note

The console only displays the status of asynchronous requests made within the previous 14 days. To view the status of older requests, use the AWS CLI, SDKs, or REST API.

Asynchronous management operations can be in one of several states:

NEW

Amazon S3 has received the request and is preparing to perform the operation.

IN_PROGRESS

Amazon S3 is currently performing the operation.

SUCCESS

The operation succeeded. The response includes relevant information, such as the Multi-Region Access Point alias for a `CreateMultiRegionAccessPoint` request.

FAILED

The operation failed. The response includes an error message indicating the reason for the request failure.

Topics

- [AWS CloudTrail with Multi-Region Access Points \(p. 333\)](#)

AWS CloudTrail with Multi-Region Access Points

You can use AWS CloudTrail to view, search, download, archive, analyze, and respond to account activity across your AWS infrastructure. With Multi-Region Access Points and CloudTrail logging, you can identify who or what took which action, what resources were acted upon, when the event occurred, and other details to help you analyze and respond to activity through your Multi-Region Access Point.

How to set up AWS CloudTrail for Multi-Region Access Points

To enable CloudTrail logging for any operations around creating or maintaining Multi-Region Access Points, you must configure CloudTrail logging to record the events in the US West (Oregon) Region.

This is true regardless of which Region you are in when making the request, or what Regions the Multi-Region Access Point supports. All requests to create or maintain a Multi-Region Access Point are routed through the US West (Oregon) Region. You should either add this Region to an existing trail or create a new trail containing this Region and all the Regions associated with the Multi-Region Access Point.

Amazon S3 logs requests made through a Multi-Region Access Point and requests made to the API operations that manage access points, such as `CreateMultiRegionAccessPoint` and `GetMultiRegionAccessPointPolicy`. When you log these requests through a Multi-Region Access Point, they appear in your AWS CloudTrail logs with the hostname of the Multi-Region Access Point. For example, if you make requests to a bucket through a Multi-Region Access Point with the alias `mfzwi23gnjvgw.mrap`, entries in the CloudTrail log would have a hostname of `mfzwi23gnjvgw.mrap.accesspoint.s3-global.amazonaws.com`.

Remember that Multi-Region Access Points serve to route requests to the bucket that responds with the lowest latency. Because of this, when you are looking at the CloudTrail logs for a Multi-Region Access Point, you will see requests being made of the underlying buckets. Some of those requests might be direct requests to the bucket and not routed through the Multi-Region Access Point. This is important to keep in mind when reviewing traffic. When a bucket is in a Multi-Region Access Point, requests can still be made to that bucket directly without going through the Multi-Region Access Point.

There are asynchronous events involved with creating and managing Multi-Region Access Points. Asynchronous requests don't have completion events in the CloudTrail log. For more information about asynchronous requests, see [Monitoring and logging requests made to Multi-Region Access Point management APIs \(p. 333\)](#).

For more information about AWS CloudTrail, see [What Is AWS CloudTrail?](#) in the *AWS CloudTrail User Guide*.

Multi-Region Access Point restrictions and limitations

Multi-Region Access Points in Amazon S3 have the following restrictions and limitations.

- Multi-Region Access Point names:
 - Must be unique within a single AWS account.
 - Must begin with a number or lowercase letter.
 - Must be between 3 and 50 characters long.
 - Can't begin or end with a dash.
 - Can't contain underscores, uppercase letters, or periods.
 - Can't be edited after they are created.
- Multi-Region Access Point aliases are generated by Amazon S3 and can't be edited or reused.
- You cannot access data through a Multi-Region Access Point using gateway endpoints or interface endpoints. In order to use AWS PrivateLink, you must create Multi-Region Access Point endpoints. For more information, see [Configuring a Multi-Region Access Point for use with AWS PrivateLink \(p. 324\)](#).
- You cannot use a Multi-Region Access Point as the distribution origin for Amazon CloudFront.
- Multi-Region Access Point minimum requirements:
 - Transport Layer Security (TLS) v1.2
 - Signature Version 4 (SigV4A)

Multi-Region Access Points support Signature Version 4A. This version of SigV4 allows requests to be signed for multiple AWS Regions. This is useful in API operations that might result in data access from one of several Regions. When using the AWS SDK, you supply your credentials and the requests

to Multi-Region Access Points will use Signature Version 4A without additional configuration. For more information about SigV4A, see [Signing AWS API requests](#) in the *AWS General Reference*.

- Multi-Region Access Point limitations:
 - IPv6 is not supported.
 - Amazon S3 on Outposts buckets are not supported.
 - No CopyObject support, either as the source or destination.
 - No support for S3 Batch Operations.
- Service quota limits:
 - There is a maximum of 100 Multi-Region Access Points per account.
 - There is a limit of 20 Regions for a single Multi-Region Access Point.
- Only the following AWS Regions are supported:
 - US East (N. Virginia)
 - US East (Ohio)
 - US West (N. California)
 - US West (Oregon)
 - Asia Pacific (Mumbai)
 - Asia Pacific (Osaka)
 - Asia Pacific (Seoul)
 - Asia Pacific (Singapore)
 - Asia Pacific (Sydney)
 - Asia Pacific (Tokyo)
 - Canada (Central)
 - Europe (Frankfurt)
 - Europe (Ireland)
 - Europe (London)
 - Europe (Paris)
 - Europe (Stockholm)
 - South America (São Paulo)

Amazon S3 security

Cloud security at AWS is the highest priority. As an AWS customer, you benefit from a data center and network architecture that are built to meet the requirements of the most security-sensitive organizations.

Security is a shared responsibility between AWS and you. The [shared responsibility model](#) describes this as security of the cloud and security *in* the cloud:

Security of the cloud

AWS is responsible for protecting the infrastructure that runs AWS services in the AWS Cloud. AWS also provides you with services that you can use securely. The effectiveness of our security is regularly tested and verified by third-party auditors as part of the [AWS compliance programs](#). To learn about the compliance programs that apply to Amazon S3, see [AWS Services in Scope by Compliance Program](#).

Security in the cloud

Your responsibility is determined by the AWS service that you use. You are also responsible for other factors including the sensitivity of your data, your organization's requirements, and applicable laws and regulations. For Amazon S3, your responsibility includes the following areas:

- Managing your data, including [object ownership](#) and [encryption](#).
- Classifying your assets.
- [Managing access](#) to your data using [IAM roles](#) and other service configurations to apply the appropriate permissions.
- Enabling detective controls such as [AWS CloudTrail](#) or [Amazon GuardDuty](#) for Amazon S3.

This documentation will help you understand how to apply the shared responsibility model when using Amazon S3. The following topics show you how to configure Amazon S3 to meet your security and compliance objectives. You'll also learn how to use other AWS services that can help you monitor and secure your Amazon S3 resources.

Topics

- [Data protection in Amazon S3 \(p. 336\)](#)
- [Protecting data using encryption \(p. 337\)](#)
- [Internetwork traffic privacy \(p. 385\)](#)
- [AWS PrivateLink for Amazon S3 \(p. 385\)](#)
- [Identity and access management in Amazon S3 \(p. 394\)](#)
- [Controlling ownership of objects and disabling ACLs for your bucket \(p. 601\)](#)
- [Logging and monitoring in Amazon S3 \(p. 626\)](#)
- [Compliance Validation for Amazon S3 \(p. 627\)](#)
- [Resilience in Amazon S3 \(p. 628\)](#)
- [Infrastructure security in Amazon S3 \(p. 631\)](#)
- [Configuration and vulnerability analysis in Amazon S3 \(p. 632\)](#)
- [Security Best Practices for Amazon S3 \(p. 633\)](#)

Data protection in Amazon S3

Amazon S3 provides a highly durable storage infrastructure designed for mission-critical and primary data storage. S3 Standard, S3 Intelligent-Tiering, S3 Standard-IA, S3 Glacier Instant Retrieval, S3 Glacier

Flexible Retrieval, and S3 Glacier Deep Archive redundantly store objects on multiple devices across a minimum of three Availability Zones in an AWS Region. An Availability Zone is one or more discrete data centers with redundant power, networking, and connectivity in an AWS Region. Availability Zones are physically separated by a meaningful distance, many kilometers, from any other Availability Zone, although all are within 100 km (60 miles) of each other. The S3 One Zone-IA storage class stores data redundantly across multiple devices within a single Availability Zone. These services are designed to handle concurrent device failures by quickly detecting and repairing any lost redundancy, and they also regularly verify the integrity of your data using checksums.

Amazon S3 standard storage offers the following features:

- Backed with the [Amazon S3 Service Level Agreement](#).
- Designed to provide 99.999999999% durability and 99.99% availability of objects over a given year.
- S3 Standard, S3 Intelligent-Tiering, S3 Standard-IA, S3 Glacier Instant Retrieval, S3 Glacier Flexible Retrieval, and S3 Glacier Deep Archive are all designed to sustain data in the event of the loss of an entire Amazon S3 Availability Zone.

Amazon S3 further protects your data using versioning. You can use versioning to preserve, retrieve, and restore every version of every object that is stored in your Amazon S3 bucket. With versioning, you can easily recover from both unintended user actions and application failures. By default, requests retrieve the most recently written version. You can retrieve older versions of an object by specifying a version of the object in a request.

For data protection purposes, we recommend that you protect AWS account credentials and set up individual user accounts with AWS Identity and Access Management, so that each user is given only the permissions necessary to fulfill their job duties.

If you require FIPS 140-2 validated cryptographic modules when accessing AWS through a command line interface or an API, use a FIPS endpoint. For more information about the available FIPS endpoints, see [Federal Information Processing Standard \(FIPS\) 140-2](#).

The following security best practices also address data protection in Amazon S3:

- [Implement server-side encryption](#)
- [Enforce encryption of data in transit](#)
- [Consider using Macie with Amazon S3](#)
- [Identify and audit all your Amazon S3 buckets](#)
- [Monitor Amazon Web Services security advisories](#)

Protecting data using encryption

Data protection refers to protecting data while in-transit (as it travels to and from Amazon S3) and at rest (while it is stored on disks in Amazon S3 data centers). You can protect data in transit using Secure Socket Layer/Transport Layer Security (SSL/TLS) or client-side encryption. You have the following options for protecting data at rest in Amazon S3:

- **Server-Side Encryption** – Request Amazon S3 to encrypt your object before saving it on disks in its data centers and then decrypt it when you download the objects.

To configure server-side encryption, see [Specifying server-side encryption with AWS KMS \(SSE-KMS\) \(p. 342\)](#) or [Specifying Amazon S3 encryption \(p. 357\)](#).

- **Client-Side Encryption** – Encrypt data client-side and upload the encrypted data to Amazon S3. In this case, you manage the encryption process, the encryption keys, and related tools.

To configure client-side encryption, see [Protecting data using client-side encryption \(p. 381\)](#).

For more information about server-side encryption and client-side encryption, review the topics listed below.

Topics

- [Protecting data using server-side encryption \(p. 338\)](#)
- [Protecting data using client-side encryption \(p. 381\)](#)

Protecting data using server-side encryption

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 its data centers and decrypts it for you when you access it. As long as you authenticate your request and you have access permissions, there is no difference in the way you access encrypted or unencrypted objects. For example, if you share your objects using a presigned URL, that URL works the same way for both encrypted and unencrypted objects. Additionally, when you list objects in your bucket, the list API returns a list of all objects, regardless of whether they are encrypted.

Note

You can't apply different types of server-side encryption to the same object simultaneously.

You have three mutually exclusive options, depending on how you choose to manage the encryption keys.

Server-Side Encryption with Amazon S3-Managed Keys (SSE-S3)

When you use Server-Side Encryption with Amazon S3-Managed Keys (SSE-S3), each object is encrypted with a unique key. As an additional safeguard, it encrypts the key itself with a root key that it regularly rotates. Amazon S3 server-side encryption uses one of the strongest block ciphers available, 256-bit Advanced Encryption Standard (AES-256) GCM, to encrypt your data. For objects encrypted prior to AES-GCM, AES-CBC is still supported to decrypt those objects. For more information, see [Protecting data using server-side encryption with Amazon S3-managed encryption keys \(SSE-S3\) \(p. 355\)](#).

Server-Side Encryption with KMS keys Stored in AWS Key Management Service (SSE-KMS)

Server-Side Encryption with AWS KMS keys (SSE-KMS) is similar to SSE-S3, but with some additional benefits and charges for using this service. There are separate permissions for the use of a KMS key that provides added protection against unauthorized access of your objects in Amazon S3. SSE-KMS also provides you with an audit trail that shows when your KMS key was used and by whom. Additionally, you can create and manage customer managed keys or use AWS managed keys that are unique to you, your service, and your Region. For more information, see [Protecting data using server-side encryption with AWS Key Management Service \(SSE-KMS\) \(p. 338\)](#).

Server-Side Encryption with Customer-Provided Keys (SSE-C)

With Server-Side Encryption with Customer-Provided Keys (SSE-C), you manage the encryption keys and Amazon S3 manages the encryption, as it writes to disks, and decryption, when you access your objects. For more information, see [Protecting data using server-side encryption with customer-provided encryption keys \(SSE-C\) \(p. 366\)](#).

Protecting data using server-side encryption with AWS Key Management Service (SSE-KMS)

Server-side encryption is the encryption of data at its destination by the application or service that receives it. AWS Key Management Service (AWS KMS) is a service that combines secure, highly available hardware and software to provide a key management system scaled for the cloud. Amazon S3 uses AWS KMS keys to encrypt your Amazon S3 objects. AWS KMS encrypts only the object data. The checksum, along with the specified algorithm, are stored as part of the object's metadata. If server-side encryption is requested for the object, then the checksum is stored in encrypted form.

If you use KMS keys, you can use AWS KMS through the [AWS Management Console](#) or the [AWS KMS APIs](#) to do the following:

- Centrally create KMS keys
- Define the policies that control how KMS keys can be used
- Audit their usage to prove that they are being used correctly

The security controls in AWS KMS can help you meet encryption-related compliance requirements. You can use these KMS keys to protect your data in Amazon S3 buckets. When you use SSE-KMS encryption with an S3 bucket, the AWS KMS keys must be in the same Region as the bucket.

There are additional charges for using AWS KMS keys. For more information, see [AWS KMS key concepts](#) in the [AWS Key Management Service Developer Guide](#) and [AWS KMS pricing](#).

Permissions

To upload an object encrypted with an AWS KMS key to Amazon S3, you need `kms:GenerateDataKey` permissions on the key. To download an object encrypted with an AWS KMS key, you need `kms:Decrypt` permissions. For information about AWS KMS permissions required for multipart upload, see [Multipart upload API and permissions \(p. 170\)](#).

Topics

- [AWS KMS keys \(p. 339\)](#)
- [Amazon S3 Bucket Keys \(p. 340\)](#)
- [Requiring server-side encryption \(p. 340\)](#)
- [Encryption context \(p. 341\)](#)
- [AWS Signature Version 4 \(p. 342\)](#)
- [Specifying server-side encryption with AWS KMS \(SSE-KMS\) \(p. 342\)](#)
- [Reducing the cost of SSE-KMS with Amazon S3 Bucket Keys \(p. 347\)](#)

AWS KMS keys

When you use server-side encryption with AWS KMS (SSE-KMS), you can use the default [AWS managed key](#), or you can specify a [customer managed key](#) that you have already created. AWS KMS uses *envelope encryption* to further protect your data. Envelope encryption is the practice of encrypting your plaintext data with a data key, and then encrypting that data key with a root key.

If you don't specify a customer managed key, Amazon S3 automatically creates an AWS KMS key in your AWS account the first time that you add an object encrypted with SSE-KMS to a bucket. By default, Amazon S3 uses this KMS key for SSE-KMS.

If you want to use a customer managed key for SSE-KMS, create the customer managed key before you configure SSE-KMS. Then, when you configure SSE-KMS for your bucket, specify the existing customer managed key.

Creating a customer managed key gives you more flexibility and control. For example, you can create, rotate, and disable customer managed keys. You can also define access controls and audit the customer managed key that you use to protect your data. For more information about customer managed and AWS managed keys, see [AWS KMS concepts](#) in the [AWS Key Management Service Developer Guide](#).

If you choose to encrypt your data using a KMS key or customer managed key, AWS KMS and Amazon S3 perform the following actions:

- Amazon S3 requests a plaintext [data key](#) and a copy of the key encrypted under the specified KMS key.

- AWS KMS generates a data key, encrypts it under the KMS key, and sends both the plaintext data key and the encrypted data key to Amazon S3.
- Amazon S3 encrypts the data using the data key and removes the plaintext key from memory as soon as possible after use.
- Amazon S3 stores the encrypted data key as metadata with the encrypted data.

When you request that your data be decrypted, Amazon S3 and AWS KMS perform the following actions:

- Amazon S3 sends the encrypted data key to AWS KMS.
- AWS KMS decrypts the key by using the same KMS key and returns the plaintext data key to Amazon S3.
- Amazon S3 decrypts the ciphertext and removes the plaintext data key from memory as soon as possible.

Important

When you use an AWS KMS key for server-side encryption in Amazon S3, you must choose a symmetric encryption KMS key. Amazon S3 supports only symmetric encryption KMS keys and not asymmetric keys. For more information, see [Using Symmetric and Asymmetric Keys](#) in the [AWS Key Management Service Developer Guide](#).

Amazon S3 Bucket Keys

When you configure server-side encryption using AWS KMS (SSE-KMS), you can configure your bucket to use S3 Bucket Keys for SSE-KMS. Using a bucket-level key for SSE-KMS can reduce your AWS KMS request costs by up to 99 percent by decreasing the request traffic from Amazon S3 to AWS KMS.

When you configure your bucket to use S3 Bucket Keys for SSE-KMS on new objects, AWS KMS generates a bucket-level key that is used to create unique [data keys](#) for objects in the bucket. This bucket key is used for a time-limited period within Amazon S3, further reducing the need for Amazon S3 to make requests to AWS KMS to complete encryption operations. For more information about using S3 Bucket Keys, see [Reducing the cost of SSE-KMS with Amazon S3 Bucket Keys \(p. 347\)](#).

Requiring server-side encryption

To require server-side encryption of all objects in a particular Amazon S3 bucket, you can use a policy. For example, the following bucket policy denies the upload object (`s3:PutObject`) permission to everyone if the request does not include the `x-amz-server-side-encryption` header requesting server-side encryption with SSE-KMS.

```
{  
    "Version": "2012-10-17",  
    "Id": "PutObjectPolicy",  
    "Statement": [  
        {  
            "Sid": "DenyUnEncryptedObjectUploads",  
            "Effect": "Deny",  
            "Principal": "*",  
            "Action": "s3:PutObject",  
            "Resource": "arn:aws:s3:::DOC-EXAMPLE-BUCKET1/*",  
            "Condition": {  
                "StringNotEquals": {  
                    "s3:x-amz-server-side-encryption": "aws:kms"  
                }  
            }  
        }  
    ]  
}
```

To require that a particular AWS KMS key be used to encrypt the objects in a bucket, you can use the `s3:x-amz-server-side-encryption-aws-kms-key-id` condition key. To specify the KMS key, you must use a key Amazon Resource Name (ARN) that is in the "`arn:aws:kms:region:acct-id:key/key-id`" format.

Note

When you upload an object, you can specify the KMS key using the `x-amz-server-side-encryption-aws-kms-key-id` header. If the header is not present in the request, Amazon S3 assumes that you want to use the AWS managed key. Regardless, the AWS KMS key ID that Amazon S3 uses for object encryption must match the AWS KMS key ID in the policy, otherwise Amazon S3 denies the request.

For a complete list of Amazon S3-specific condition keys, see [Condition keys for Amazon S3](#).

Encryption context

An encryption context is a set of key-value pairs that contains additional contextual information about the data. The encryption context is not encrypted. When an encryption context is specified for an encryption operation, Amazon S3 must specify the same encryption context for the decryption operation. Otherwise, the decryption fails. AWS KMS uses the encryption context as [additional authenticated data \(AAD\)](#) to support [authenticated encryption](#). For more information about the encryption context, see [Encryption context in the AWS Key Management Service Developer Guide](#).

Amazon S3 automatically uses the object or bucket Amazon Resource Name (ARN) as the encryption context pair:

- **If you use SSE-KMS without enabling an S3 Bucket Key**, the object ARN is used as the encryption context.

```
arn:aws:s3:::object_ARN
```

- **If you use SSE-KMS and enable an S3 Bucket Key**, the bucket ARN is used as the encryption context. For more information about S3 Bucket Keys, see [Reducing the cost of SSE-KMS with Amazon S3 Bucket Keys \(p. 347\)](#).

```
arn:aws:s3:::bucket_ARN
```

You can optionally provide an additional encryption context pair using the `x-amz-server-side-encryption-context` header in an [S3:PutObject](#) request. However, because the encryption context is not encrypted, make sure it does not include sensitive information. Amazon S3 stores this additional key pair alongside the default encryption context. When it processes your PUT request, Amazon S3 appends the default encryption context of `aws:s3:arn` to the one that you provide.

You can use the encryption context to identify and categorize your cryptographic operations. You can also use the default encryption context ARN value to track relevant requests in AWS CloudTrail by viewing which Amazon S3 ARN was used with which encryption key.

In the `requestParameters` field of a CloudTrail log file, the encryption context looks similar to the following one.

```
"encryptionContext": {  
    "aws:s3:arn": "arn:aws:s3:::DOC-EXAMPLE-BUCKET1/file_name"  
}
```

When you use SSE-KMS with the optional S3 Bucket Keys feature, the encryption context value is the ARN of the bucket.

```
"encryptionContext": {
```

```
    "aws:s3:arn": "arn:aws:s3:::DOC-EXAMPLE-BUCKET1"  
}
```

AWS Signature Version 4

Signature Version 4 is the process of adding authentication information to AWS requests sent by HTTP. For security, most requests to AWS must be signed with an access key, which consists of an access key ID and secret access key. These two keys are commonly referred to as your security credentials. For more information, see [Authenticating Requests \(AWS Signature Version 4\)](#) and [Signature Version 4 signing process](#).

Important

- All GET and PUT requests for AWS KMS encrypted objects must be made using Secure Sockets Layer (SSL) or Transport Layer Security (TLS). Requests must also be signed using valid credentials, such as AWS Signature Version 4 (or AWS Signature Version 2).
- If your object uses SSE-KMS, don't send encryption request headers for GET requests and HEAD requests, or you'll get an HTTP 400 BadRequest error.

Topics

- [Specifying server-side encryption with AWS KMS \(SSE-KMS\) \(p. 342\)](#)
- [Reducing the cost of SSE-KMS with Amazon S3 Bucket Keys \(p. 347\)](#)

Specifying server-side encryption with AWS KMS (SSE-KMS)

When you create an object, you can specify the use of server-side encryption with AWS Key Management Service (AWS KMS) keys to encrypt your data. This encryption is known as SSE-KMS. You can apply encryption when you are either uploading a new object or copying an existing object.

You can specify SSE-KMS using the Amazon S3 console, REST API operations, AWS SDKs, and AWS Command Line Interface (AWS CLI). For more information, see the following topics.

Note

You can use multi-Region AWS KMS keys in Amazon S3. However, Amazon S3 currently treats multi-Region keys as though they were single-Region keys, and does not use the multi-Region features of the key. For more information, see [Using multi-Region keys in AWS Key Management Service Developer Guide](#).

Using the S3 console

This topic describes how to set or change the type of encryption an object using the Amazon S3 console.

Note

If you change an object's encryption, a new object is created to replace the old one. If S3 Versioning is enabled, a new version of the object is created, and the existing object becomes an older version. The role that changes the property also becomes the owner of the new object or (object version).

To add or change encryption for 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 **Buckets** list, choose the name of the bucket that contains the object.
3. In the **Objects** list, choose the name of the object that you want to add or change encryption for.
The **Object overview** opens, displaying the properties for your object.
4. Under **Server-side encryption settings**, choose **Edit**.

The **Edit server-side encryption** page opens

5. To enable server-side encryption for your object, under **Server-side encryption**, choose **Enable**.
6. Under **Encryption key type**, choose **AWS Key Management Service key (SSE-KMS)**.

Important

If you use the AWS KMS option for your default encryption configuration, you are subject to the RPS (requests per second) limits of AWS KMS. For more information about AWS KMS limits and how to request a limit increase, see [AWS KMS limits](#).

7. Under **AWS KMS key**, choose one of the following:

- **AWS managed key (aws/s3)**
- **Choose from your AWS KMS keys**, and choose your **KMS key**.
- **Enter KMS master key ARN**, and enter your AWS KMS key Amazon Resource Name (ARN).

Important

You can use only AWS KMS keys that are enabled in the same AWS Region as the bucket. When you choose **Choose from your AWS KMS keys**, the S3 console lists only 100 KMS keys per Region. If you have more than 100 KMS keys in the same Region, you can see only the first 100 KMS keys in the S3 console. To use a KMS key that is not listed in the console, choose **Custom KMS ARN**, and enter the KMS key ARN.

When you use an AWS KMS key for server-side encryption in Amazon S3, you must choose a KMS key that is enabled in the same Region as your bucket. Additionally, Amazon S3 supports only symmetric encryption KMS keys, and not asymmetric KMS keys. For more information, see [Using Symmetric and Asymmetric Keys](#) in the *AWS Key Management Service Developer Guide*.

For more information about creating an AWS KMS key, see [Creating Keys](#) in the *AWS Key Management Service Developer Guide*. For more information about using AWS KMS with Amazon S3, see [Protecting data using server-side encryption with AWS Key Management Service \(SSE-KMS\)](#) (p. 338).

8. Choose **Save changes**.

Note

This action applies encryption to all specified objects. When encrypting folders, wait for the save operation to finish before adding new objects to the folder.

Using the REST API

When you create an object—that is, when you upload a new object or copy an existing object—you can specify the use of server-side encryption with AWS KMS keys to encrypt your data. To do this, add the `x-amz-server-side-encryption` header to the request. Set the value of the header to the encryption algorithm `aws:kms`. Amazon S3 confirms that your object is stored using SSE-KMS by returning the response header `x-amz-server-side-encryption`.

If you specify the `x-amz-server-side-encryption` header with a value of `aws:kms`, you can also use the following request headers:

- `x-amz-server-side-encryption-aws-kms-key-id`
- `x-amz-server-side-encryption-context`
- `x-amz-server-side-encryption-bucket-key-enabled`

Topics

- [Amazon S3 REST API operations that support SSE-KMS \(p. 344\)](#)

- [Encryption context \(x-amz-server-side-encryption-context\) \(p. 344\)](#)
- [AWS KMS key ID \(x-amz-server-side-encryption-aws-kms-key-id\) \(p. 345\)](#)
- [S3 Bucket Keys \(x-amz-server-side-encryption-aws-bucket-key-enabled\) \(p. 345\)](#)

Amazon S3 REST API operations that support SSE-KMS

The following REST API operations accept the `x-amz-server-side-encryption`, `x-amz-server-side-encryption-aws-kms-key-id`, and `x-amz-server-side-encryption-context` request headers.

- [PUT object](#)— When you upload data using the PUT API operation, you can specify these request headers.
- [COPY object](#)— When you copy an object, you have both a source object and a target object. When you pass SSE-KMS headers with the COPY operation, they are applied only to the target object. When copying an existing object, regardless of whether the source object is encrypted or not, the destination object is not encrypted unless you explicitly request server-side encryption.
- [POST Object](#)— When you use a POST operation to upload an object, instead of the request headers, you provide the same information in the form fields.
- [Create Multipart Upload](#)— When you upload large objects using the multipart upload API, you can specify these headers. You specify these headers in the initiate request.

The response headers of the following REST API operations return the `x-amz-server-side-encryption` header when an object is stored using server-side encryption.

- [PUT object](#)
- [COPY object](#)
- [POST Object](#)
- [Create Multipart Upload](#)
- [Upload Part](#)
- [Upload Part Copy](#)
- [Complete Multipart Upload](#)
- [Get Object](#)
- [Head Object](#)

Important

- All GET and PUT requests for an object protected by AWS KMS fail if you don't make them using Secure Sockets Layer (SSL), Transport Layer Security (TLS), or Signature Version 4.
- If your object uses SSE-KMS, don't send encryption request headers for GET requests and HEAD requests, or you'll get an HTTP 400 BadRequest error.

Encryption context (x-amz-server-side-encryption-context)

If you specify `x-amz-server-side-encryption:aws:kms`, the Amazon S3 API supports an encryption context with the `x-amz-server-side-encryption-context` header. An encryption context is a set of key-value pairs that contain additional contextual information about the data.

Amazon S3 automatically uses the object or bucket Amazon Resource Name (ARN) as the encryption context pair. If you use SSE-KMS without enabling an S3 Bucket Key, you use the object ARN as your encryption context, for example, `arn:aws:s3:::object_ARN`. However, if you use SSE-KMS and enable an S3 Bucket Key, you use the bucket ARN for your encryption context, for example, `arn:aws:s3:::bucket_ARN`.

You can optionally provide an additional encryption context pair using the `x-amz-server-side-encryption-context` header. However, because the encryption context is not encrypted, make sure it does not include sensitive information. Amazon S3 stores this additional key pair alongside the default encryption context.

For information about the encryption context in Amazon S3, see [Encryption context \(p. 341\)](#). For general information about the encryption context, see [AWS Key Management Service Concepts - Encryption context](#) in the *AWS Key Management Service Developer Guide*.

[AWS KMS key ID \(x-amz-server-side-encryption-aws-kms-key-id\)](#)

You can use the `x-amz-server-side-encryption-aws-kms-key-id` header to specify the ID of the customer managed key used to protect the data. If you specify `x-amz-server-side-encryption:aws:kms`, but don't provide `x-amz-server-side-encryption-aws-kms-key-id`, Amazon S3 uses the AWS managed key to protect the data. If you want to use a customer managed key, you must provide the `x-amz-server-side-encryption-aws-kms-key-id` header of the customer managed key.

Important

When you use an AWS KMS key for server-side encryption in Amazon S3, you must choose a symmetric encryption KMS key. Amazon S3 supports only symmetric encryption KMS keys and not asymmetric keys. For more information, see [Using Symmetric and Asymmetric Keys](#) in the *AWS Key Management Service Developer Guide*.

[S3 Bucket Keys \(x-amz-server-side-encryption-aws-bucket-key-enabled\)](#)

You can use the `x-amz-server-side-encryption-aws-bucket-key-enabled` request header to enable or disable an S3 Bucket Key at the object-level. S3 Bucket Keys can reduce your AWS KMS request costs by decreasing the request traffic from Amazon S3 to AWS KMS. For more information, see [Reducing the cost of SSE-KMS with Amazon S3 Bucket Keys \(p. 347\)](#).

If you specify `x-amz-server-side-encryption:aws:kms`, but don't provide `x-amz-server-side-encryption-aws-bucket-key-enabled`, your object uses the S3 Bucket Key settings for the destination bucket to encrypt your object. For more information, see [Configuring an S3 Bucket Key at the object level using Batch Operations, REST API, AWS SDKs, or AWS CLI \(p. 352\)](#).

[Using the AWS SDKs](#)

When using AWS SDKs, you can request Amazon S3 to use AWS KMS keys. This section provides examples of using the AWS SDKs for Java and .NET. For information about other SDKs, go to [Sample Code and Libraries](#).

Important

When you use an AWS KMS key for server-side encryption in Amazon S3, you must choose a symmetric encryption KMS key. Amazon S3 supports only symmetric encryption KMS keys and not asymmetric keys. For more information, see [Using Symmetric and Asymmetric Keys](#) in the *AWS Key Management Service Developer Guide*.

[Copy operation](#)

When copying objects, you add the same request properties (`ServerSideEncryptionMethod` and `ServerSideEncryptionKeyManagementServiceKeyId`) to request Amazon S3 to use an AWS KMS key. For more information about copying objects, see [Copying objects \(p. 201\)](#).

[Put operation](#)

[Java](#)

When uploading an object using the AWS SDK for Java, you can request Amazon S3 to use an AWS KMS key by adding the `SSEAwsKeyManagementParams` property as shown in the following request.

```
PutObjectRequest putRequest = new PutObjectRequest(bucketName,  
    keyName, file).withSSEAwsKeyManagementParams(new SSEAwsKeyManagementParams());
```

In this case, Amazon S3 uses the AWS managed key (see [Protecting data using server-side encryption with AWS Key Management Service \(SSE-KMS\) \(p. 338\)](#)). You can optionally create a symmetric encryption KMS key and specify that in the request.

```
PutObjectRequest putRequest = new PutObjectRequest(bucketName,  
    keyName, file).withSSEAwsKeyManagementParams(new SSEAwsKeyManagementParams(keyID));
```

For more information about creating customer managed keys, see [Programming the AWS KMS API in the AWS Key Management Service Developer Guide](#).

For working code examples of uploading an object, see the following topics. You will need to update those code examples and provide encryption information as shown in the preceding code fragment.

- For uploading an object in a single operation, see [Uploading objects \(p. 158\)](#).
- For a multipart upload, see the following topics:
 - Using the high-level multipart upload API, see [Uploading an object using multipart upload \(p. 174\)](#).
 - Using the low-level multipart upload API, see [Using the AWS SDKs \(low-level-level API\) \(p. 180\)](#).

.NET

When uploading an object using the AWS SDK for .NET, you can request Amazon S3 to use an AWS KMS key by adding the `ServerSideEncryptionMethod` property as shown in the following request.

```
PutObjectRequest putRequest = new PutObjectRequest  
{  
    BucketName = bucketName,  
    Key = keyName,  
    // other properties.  
    ServerSideEncryptionMethod = ServerSideEncryptionMethod.AWSKMS  
};
```

In this case, Amazon S3 uses the AWS managed key. For more information, see [Protecting data using server-side encryption with AWS Key Management Service \(SSE-KMS\) \(p. 338\)](#). You can optionally create your own symmetric encryption customer managed key and specify that in the request.

```
PutObjectRequest putRequest1 = new PutObjectRequest  
{  
    BucketName = bucketName,  
    Key = keyName,  
    // other properties.  
    ServerSideEncryptionMethod = ServerSideEncryptionMethod.AWSKMS,  
    ServerSideEncryptionKeyManagementServiceKeyId = keyId  
};
```

For more information about creating customer managed keys, see [Programming the AWS KMS API in the AWS Key Management Service Developer Guide](#).

For working code examples of uploading an object, see the following topics. You will need to update these code examples and provide encryption information as shown in the preceding code fragment.

- For uploading an object in a single operation, see [Uploading objects \(p. 158\)](#).

- For multipart upload, see the following topics:
 - Using the high-level multipart upload API, see [Uploading an object using multipart upload \(p. 174\)](#).
 - Using the low-level multipart upload API, see [Uploading an object using multipart upload \(p. 174\)](#).

Presigned URLs

Java

When creating a presigned URL for an object encrypted with an AWS KMS key, you must explicitly specify Signature Version 4.

```
ClientConfiguration clientConfiguration = new ClientConfiguration();
clientConfiguration.setSignerOverride("AWSS3V4SignerType");
AmazonS3Client s3client = new AmazonS3Client(
    new ProfileCredentialsProvider(), clientConfiguration);
...
```

For a code example, see [Sharing objects using presigned URLs \(p. 258\)](#).

.NET

When creating a presigned URL for an object encrypted with an AWS KMS key, you must explicitly specify Signature Version 4.

```
AWSConfigs.S3Config.UseSignatureVersion4 = true;
```

For a code example, see [Sharing objects using presigned URLs \(p. 258\)](#).

Reducing the cost of SSE-KMS with Amazon S3 Bucket Keys

Amazon S3 Bucket Keys reduce the cost of Amazon S3 server-side encryption using AWS Key Management Service (SSE-KMS). This new bucket-level key for SSE can reduce AWS KMS request costs by up to 99 percent by decreasing the request traffic from Amazon S3 to AWS KMS. With a few clicks in the AWS Management Console, and without any changes to your client applications, you can configure your bucket to use an S3 Bucket Key for AWS KMS-based encryption on new objects.

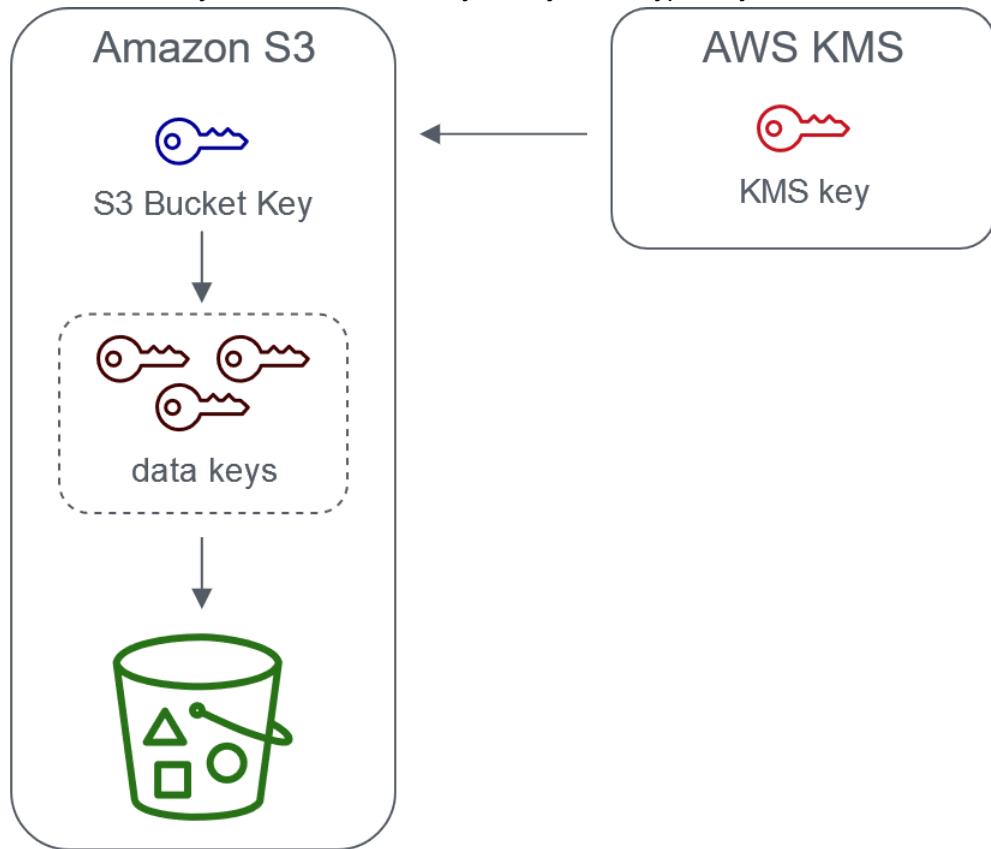
S3 Bucket Keys for SSE-KMS

Workloads that access millions or billions of objects encrypted with SSE-KMS can generate large volumes of requests to AWS KMS. When you use SSE-KMS to protect your data without an S3 Bucket Key, Amazon S3 uses an individual AWS KMS [data key](#) for every object. It makes a call to AWS KMS every time a request is made against a KMS-encrypted object. For information about how SSE-KMS works, see [Protecting data using server-side encryption with AWS Key Management Service \(SSE-KMS\) \(p. 338\)](#).

When you configure your bucket to use an S3 Bucket Key for SSE-KMS, AWS KMS generates a bucket-level key that is used to create unique data keys for *new* objects that you add to the bucket. This S3 Bucket Key is used for a time-limited period within Amazon S3, reducing the need for Amazon S3 to make requests to AWS KMS to complete encryption operations. This reduces traffic from S3 to AWS KMS, allowing you to access AWS KMS-encrypted objects in S3 at a fraction of the previous cost.

When you configure an S3 Bucket Key, objects that are already in the bucket do not use the S3 Bucket Key. To configure an S3 Bucket Key for existing objects, you can use a COPY operation. For more information, see [Configuring an S3 Bucket Key at the object level using Batch Operations, REST API, AWS SDKs, or AWS CLI \(p. 352\)](#).

Amazon S3 will only share an S3 Bucket Key for objects encrypted by the same AWS KMS key.



Server-side encryption with AWS Key Management service using an S3 Bucket Key

Configuring S3 Bucket Keys

You can configure your bucket to use an S3 Bucket Key for SSE-KMS on new objects through the Amazon S3 console, AWS SDKs, AWS CLI, or REST API. You can also override the S3 Bucket Key configuration for specific objects in a bucket with an individual per-object KMS key using the REST API, AWS SDK, or AWS CLI. You can also view S3 Bucket Key settings.

Before you configure your bucket to use an S3 Bucket Key, review [Changes to note before enabling an S3 Bucket Key \(p. 349\)](#).

Configuring an S3 Bucket Key using the Amazon S3 console

When you create a new bucket, you can configure your bucket to use an S3 Bucket Key for SSE-KMS on new objects. You can also configure an existing bucket to use an S3 Bucket Key for SSE-KMS on new objects by updating your bucket properties.

For more information, see [Configuring your bucket to use an S3 Bucket Key with SSE-KMS for new objects \(p. 350\)](#).

REST API, AWS CLI, and AWS SDK support for S3 Bucket Keys

You can use the REST API, AWS CLI, or AWS SDK to configure your bucket to use an S3 Bucket Key for SSE-KMS on new objects. You can also enable an S3 Bucket Key at the object level.

For more information, see the following:

- Configuring an S3 Bucket Key at the object level using Batch Operations, REST API, AWS SDKs, or AWS CLI (p. 352)
- Configuring your bucket to use an S3 Bucket Key with SSE-KMS for new objects (p. 350)

The following APIs support S3 Bucket Keys for SSE-KMS:

- [PutBucketEncryption](#)
 - `ServerSideEncryptionRule` accepts the `BucketKeyEnabled` parameter for enabling and disabling an S3 Bucket Key.
- [GetBucketEncryption](#)
 - `ServerSideEncryptionRule` returns the settings for `BucketKeyEnabled`.
- [PutObject](#), [CopyObject](#), [CreateMultipartUpload](#), and [PostObject](#)
 - `x-amz-server-side-encryption-bucket-key-enabled` request header enables or disables an S3 Bucket Key at the object level.
- [HeadObject](#), [GetObject](#), [UploadPartCopy](#), [UploadPart](#), and [CompleteMultipartUpload](#)
 - `x-amz-server-side-encryption-bucket-key-enabled` response header indicates if an S3 Bucket Key is enabled or disabled for an object.

Working with AWS CloudFormation

In AWS CloudFormation, the `AWS::S3::Bucket` resource includes an encryption property called `BucketKeyEnabled` that you can use to enable or disable an S3 Bucket Key.

For more information, see [Using AWS CloudFormation \(p. 352\)](#).

Changes to note before enabling an S3 Bucket Key

Before you enable an S3 Bucket Key, please note the following related changes:

IAM or KMS key policies

If your existing IAM policies or AWS KMS key policies use your object Amazon Resource Name (ARN) as the encryption context to refine or limit access to your KMS key, these policies won't work with an S3 Bucket Key. S3 Bucket Keys use the bucket ARN as encryption context. Before you enable an S3 Bucket Key, update your IAM policies or AWS KMS key policies to use your bucket ARN as encryption context.

For more information about encryption context and S3 Bucket Keys, see [Encryption context \(p. 341\)](#).

AWS KMS CloudTrail events

After you enable an S3 Bucket Key, your AWS KMS CloudTrail events log your bucket ARN instead of your object ARN. Additionally, you see fewer KMS CloudTrail events for SSE-KMS objects in your logs. Because key material is time-limited in Amazon S3, fewer requests are made to AWS KMS.

Using an S3 Bucket Key with replication

You can use S3 Bucket Keys with Same-Region Replication (SRR) and Cross-Region Replication (CRR).

When Amazon S3 replicates an encrypted object, it generally preserves the encryption settings of the replica object in the destination bucket. However, if the source object is not encrypted and your destination bucket uses default encryption or an S3 Bucket Key, Amazon S3 encrypts the object with the destination bucket's configuration.

The following examples illustrate how an S3 Bucket Key works with replication. For more information, see [Replicating objects created with server-side encryption \(SSE\) using KMS keys \(p. 814\)](#).

Example Example 1 – Source object uses S3 Bucket Keys, destination bucket uses default encryption

If your source object uses an S3 Bucket Key but your destination bucket uses default encryption with SSE-KMS, the replica object maintains its S3 Bucket Key encryption settings in the destination bucket. The destination bucket still uses default encryption with SSE-KMS.

Example Example 2 – Source object is not encrypted, destination bucket uses an S3 Bucket Key with SSE-KMS

If your source object is not encrypted and the destination bucket uses an S3 Bucket Key with SSE-KMS, the source object is encrypted with an S3 Bucket Key using SSE-KMS in the destination bucket. This results in the ETag of the source object being different from the ETag of the replica object. You must update applications that use the ETag to accommodate for this difference.

Working with S3 Bucket Keys

For more information about enabling and working with S3 Bucket Keys, see the following sections:

- [Configuring your bucket to use an S3 Bucket Key with SSE-KMS for new objects \(p. 350\)](#)
- [Configuring an S3 Bucket Key at the object level using Batch Operations, REST API, AWS SDKs, or AWS CLI \(p. 352\)](#)
- [Viewing settings for an S3 Bucket Key \(p. 354\)](#)

Configuring your bucket to use an S3 Bucket Key with SSE-KMS for new objects

When you configure server-side encryption using SSE-KMS, you can configure your bucket to use an S3 Bucket Key for SSE-KMS on new objects. S3 Bucket Keys decrease the request traffic from Amazon S3 to AWS Key Management Service (AWS KMS) and reduce the cost of SSE-KMS. For more information, see [Reducing the cost of SSE-KMS with Amazon S3 Bucket Keys \(p. 347\)](#).

You can configure your bucket to use an S3 Bucket Key for SSE-KMS on new objects by using the Amazon S3 console, REST API, AWS SDK, AWS CLI, or AWS CloudFormation. If you want to enable or disable an S3 Bucket Key for existing objects, you can use a COPY operation. For more information, see [Configuring an S3 Bucket Key at the object level using Batch Operations, REST API, AWS SDKs, or AWS CLI \(p. 352\)](#) and [Using S3 Batch Operations to encrypt objects with S3 Bucket Keys \(p. 901\)](#).

When an S3 Bucket Key is enabled for the source or destination bucket, the encryption context will be the bucket Amazon Resource Name (ARN) and not the object ARN, for example, `arn:aws:s3:::bucket_ARN`. You need to update your IAM policies to use the bucket ARN for the encryption context. For more information, see [Granting additional permissions for the IAM role \(p. 816\)](#).

The following examples illustrate how an S3 Bucket Key works with replication. For more information, see [Replicating objects created with server-side encryption \(SSE\) using KMS keys \(p. 814\)](#).

Prerequisite:

Before you configure your bucket to use an S3 Bucket Key, review [Changes to note before enabling an S3 Bucket Key \(p. 349\)](#).

Using the S3 console

In the S3 console, you can enable or disable an S3 Bucket Key for a new or existing bucket. Objects in the S3 console inherit their S3 Bucket Key setting from the bucket configuration. When you enable an S3 Bucket Key for your bucket, new objects that you upload to the bucket use an S3 Bucket Key for server-side encryption using AWS KMS.

Uploading, copying, or modifying objects in buckets that have an S3 Bucket Key enabled

If you upload, modify, or copy an object in a bucket that has an S3 Bucket Key enabled, the S3 Bucket Key settings for that object might be updated to align with bucket configuration.

If an object already has an S3 Bucket Key enabled, the S3 Bucket Key settings for that object don't change when you copy or modify the object. However, if you modify or copy an object that doesn't have an S3 Bucket Key enabled, and the destination bucket has an S3 Bucket Key configuration, the object inherits the destination bucket's S3 Bucket Key settings. For example, if your source object doesn't have an S3 Bucket Key enabled but the destination bucket has S3 Bucket Key enabled, an S3 Bucket Key will be enabled for the object.

To enable an S3 Bucket Key when you create a new bucket

1. Sign in to the AWS Management Console and open the Amazon S3 console at <https://console.aws.amazon.com/s3/>.
2. Choose **Create bucket**.
3. Enter your bucket name, and choose your AWS Region.
4. Under **Default encryption**, choose **Enable**.
5. Under **Encryption type**, choose **AWS Key Management Service key (SSE-KMS)**.
6. Choose an AWS KMS key:
 - Choose **AWS managed key (aws/s3)**.
 - Choose **Customer managed key**, and choose a symmetric encryption customer managed key in the same Region as your bucket.
7. Under **Bucket Key**, choose **Enable**.
8. Choose **Create bucket**.

Amazon S3 creates your bucket with an S3 Bucket Key enabled. New objects that you upload to the bucket will use an S3 Bucket Key. To disable an S3 Bucket Key, follow the previous steps, and choose **disable**.

To enable an S3 Bucket Key for an existing bucket

1. Open the Amazon S3 console at <https://console.aws.amazon.com/s3/>.
2. In the **Buckets** list, choose the bucket that you want to enable an S3 Bucket Key for.
3. Choose **Properties**.
4. Under **Default encryption**, choose **Edit**.
5. Under **Default encryption**, choose **Enable**.
6. Under **Encryption type**, choose **AWS Key Management Service key (SSE-KMS)**.
7. Choose an AWS KMS key:
 - Choose **AWS managed key (aws/s3)**.
 - Choose **Customer managed key**, and choose a symmetric encryption customer managed key in the same Region as your bucket.
8. Under **Bucket Key**, choose **Enable**.
9. Choose **Save changes**.

Amazon S3 enables an S3 Bucket Key for new objects added to your bucket. Existing objects don't use the S3 Bucket Key. To disable an S3 Bucket Key, follow the previous steps, and choose **Disable**.

Using the REST API

You can use [PutBucketEncryption](#) to enable or disable an S3 Bucket Key for your bucket. To configure an S3 Bucket Key with [PutBucketEncryption](#), specify the [ServerSideEncryptionRule](#), which includes

default encryption with server-side encryption using AWS KMS key. You can also optionally use a customer managed key by specifying the KMS key ID for the customer managed key.

For more information and example syntax, see [PutBucketEncryption](#).

Using the AWS SDK for Java

The following example enables default bucket encryption with SSE-KMS and an S3 Bucket Key using the AWS SDK for Java.

Java

```
AmazonS3 s3client = AmazonS3ClientBuilder.standard()
    .withRegion(Regions.DEFAULT_REGION)
    .build();

ServerSideEncryptionByDefault serverSideEncryptionByDefault = new
    ServerSideEncryptionByDefault()
        .withSSEAlgorithm(SSEAlgorithm.KMS);
ServerSideEncryptionRule rule = new ServerSideEncryptionRule()
    .withApplyServerSideEncryptionByDefault(serverSideEncryptionByDefault)
    .withBucketKeyEnabled(true);
ServerSideEncryptionConfiguration serverSideEncryptionConfiguration =
    new ServerSideEncryptionConfiguration().withRules(Collections.singleton(rule));

SetBucketEncryptionRequest setBucketEncryptionRequest = new
    SetBucketEncryptionRequest()
        .withServerSideEncryptionConfiguration(serverSideEncryptionConfiguration)
        .withBucketName(bucketName);

s3client.setBucketEncryption(setBucketEncryptionRequest);
```

Using the AWS CLI

The following example enables default bucket encryption with SSE-KMS and an S3 Bucket Key using the AWS CLI.

```
aws s3api put-bucket-encryption --bucket <bucket-name> --server-side-encryption-
configuration '{
    "Rules": [
        {
            "ApplyServerSideEncryptionByDefault": {
                "SSEAlgorithm": "aws:kms",
                "KMSMasterKeyID": "<KMS-Key-ARN>"
            },
            "BucketKeyEnabled": true
        }
    ]
}'
```

Using AWS CloudFormation

For more information about configuring an S3 Bucket Key using AWS CloudFormation, see [AWS::S3::Bucket ServerSideEncryptionRule](#) in the *AWS CloudFormation User Guide*.

[Configuring an S3 Bucket Key at the object level using Batch Operations, REST API, AWS SDKs, or AWS CLI](#)

When you perform a PUT or COPY operation using the REST API, AWS SDKs, or AWS CLI, you can enable or disable an S3 Bucket Key at the object level. S3 Bucket Keys reduce the cost of server-side encryption using AWS Key Management Service (AWS KMS) (SSE-KMS) by decreasing request traffic from Amazon

S3 to AWS KMS. For more information, see [Reducing the cost of SSE-KMS with Amazon S3 Bucket Keys \(p. 347\)](#).

When you configure an S3 Bucket Key for an object using a PUT or COPY operation, Amazon S3 only updates the settings for that object. The S3 Bucket Key settings for the destination bucket do not change. If you don't specify an S3 Bucket Key for your object, Amazon S3 applies the S3 Bucket Key settings for the destination bucket to the object.

Prerequisite:

Before you configure your object to use an S3 Bucket Key, review [Changes to note before enabling an S3 Bucket Key \(p. 349\)](#).

Topics

- [Amazon S3 Batch Operations \(p. 353\)](#)
- [Using the REST API \(p. 353\)](#)
- [Using the AWS SDK for Java \(PutObject\) \(p. 353\)](#)
- [Using the AWS CLI \(PutObject\) \(p. 354\)](#)

Amazon S3 Batch Operations

To encrypt your existing Amazon S3 objects, you can use Amazon S3 Batch Operations. You provide S3 Batch Operations with a list of objects to operate on, and Batch Operations calls the respective API to perform the specified operation. You can use the [S3 Batch Operations Copy operation](#) to copy existing unencrypted objects and write them back to the same bucket as encrypted objects. A single Batch Operations job can perform the specified operation on billions of objects. For more information, see [Performing large-scale batch operations on Amazon S3 objects \(p. 881\)](#) and [Encrypting objects with Amazon S3 Batch Operations](#).

Using the REST API

When you use SSE-KMS, you can enable an S3 Bucket Key for an object using the following APIs:

- [PutObject](#) – When you upload an object, you can specify the `x-amz-server-side-encryption-bucket-key-enabled` request header to enable or disable an S3 Bucket Key at the object level.
- [CopyObject](#) – When you copy an object and configure SSE-KMS, you can specify the `x-amz-server-side-encryption-bucket-key-enabled` request header to enable or disable an S3 Bucket Key for your object.
- [PostObject](#) – When you use a POST operation to upload an object and configure SSE-KMS, you can use the `x-amz-server-side-encryption-bucket-key-enabled` form field to enable or disable an S3 Bucket Key for your object.
- [CreateMultipartUpload](#) – When you upload large objects using the multipart upload API and configure SSE-KMS, you can use the `x-amz-server-side-encryption-bucket-key-enabled` request header to enable or disable an S3 Bucket Key for your object.

To enable an S3 Bucket Key at the object level, include the `x-amz-server-side-encryption-bucket-key-enabled` request header. For more information about SSE-KMS and the REST API, see [Using the REST API \(p. 343\)](#).

Using the AWS SDK for Java (PutObject)

You can use the following example to configure an S3 Bucket Key at the object level using the AWS SDK for Java.

Java

```
AmazonS3 s3client = AmazonS3ClientBuilder.standard()
    .withRegion(Regions.DEFAULT_REGION)
    .build();

String bucketName = "bucket name";
String keyName = "key name for object";
String contents = "file contents";

PutObjectRequest putObjectRequest = new PutObjectRequest(bucketName, keyName, contents)
    .withBucketKeyEnabled(true);

s3client.putObject(putObjectRequest);
```

Using the AWS CLI (PutObject)

You can use the following AWS CLI example to configure an S3 Bucket Key at the object level as part of a PutObject request.

```
aws s3api put-object --bucket <bucket name> --key <object key name> --server-side-
encryption aws:kms --bucket-key-enabled --body <filepath>
```

Viewing settings for an S3 Bucket Key

You can view settings for an S3 Bucket Key at the bucket or object level using the Amazon S3 console, REST API, AWS CLI, or AWS SDKs.

S3 Bucket Keys decrease request traffic from Amazon S3 to AWS KMS and reduce the cost of server-side encryption using AWS Key Management Service (SSE-KMS). For more information, see [Reducing the cost of SSE-KMS with Amazon S3 Bucket Keys \(p. 347\)](#).

To view S3 Bucket Key settings for a bucket or an object that has inherited S3 Bucket Key settings from the bucket configuration, you need permission to perform the `s3:GetEncryptionConfiguration` action. For more information, see [GetBucketEncryption](#) in the *Amazon Simple Storage Service API Reference*.

Using the S3 console

In the S3 console, you can view the S3 Bucket Key settings for your bucket or object. S3 Bucket Key settings are inherited from the bucket configuration unless the source objects already has an S3 Bucket Key configured.

Objects and folders in the same bucket can have different S3 Bucket Key settings. For example, if you upload an object using the REST API and enable an S3 Bucket Key for the object, the object retains its S3 Bucket Key setting in the destination bucket, even if S3 Bucket Key is disabled in the destination bucket. As another example, if you enable an S3 Bucket Key for an existing bucket, objects that are already in the bucket do not use an S3 Bucket Key. However, new objects have an S3 Bucket Key enabled.

To view bucket-level an S3 Bucket Key setting

1. Sign in to the AWS Management Console and open the Amazon S3 console at <https://console.aws.amazon.com/s3/>.
2. In the **Buckets** list, choose the bucket that you want to enable an S3 Bucket Key for.
3. Choose **Properties**.
4. In the **Default encryption** section, under **Bucket Key**, you see the S3 Bucket Key setting for your bucket.

If you can't see the S3 Bucket Key setting, you might not have permission to perform the `s3:GetEncryptionConfiguration` action. For more information, see [GetBucketEncryption](#) in the *Amazon Simple Storage Service API Reference*.

To view the S3 Bucket Key setting for your 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 **Buckets** list, choose the bucket that you want to enable an S3 Bucket Key for.
3. In the **Objects** list, choose your object name.
4. On the **Details** tab, under **Server-side encryption settings**, choose **Edit**.

Under **Bucket Key**, you see the S3 Bucket Key setting for your object but you cannot edit it.

Using the REST API

To return bucket-level S3 Bucket Key settings

To return encryption information for a bucket, including settings for an S3 Bucket Key, use the `GetBucketEncryption` operation. S3 Bucket Key settings are returned in the response body in the `ServerSideEncryptionConfiguration` with the `BucketKeyEnabled` setting. For more information, see [GetBucketEncryption](#) in the *Amazon S3 API Reference*.

To return object-level settings for an S3 Bucket Key

To return the S3 Bucket Key status for an object, use the `HeadObject` operation. `HeadObject` returns the `x-amz-server-side-encryption-bucket-key-enabled` response header to show whether an S3 Bucket Key is enabled or disabled for the object. For more information, see [HeadObject](#) in the *Amazon S3 API Reference*.

The following API operations also return the `x-amz-server-side-encryption-bucket-key-enabled` response header if an S3 Bucket Key is configured for an object:

- [PutObject](#)
- [PostObject](#)
- [CopyObject](#)
- [CreateMultipartUpload](#)
- [UploadPartCopy](#)
- [UploadPart](#)
- [CompleteMultipartUpload](#)
- [GetObject](#)

Protecting data using server-side encryption with Amazon S3-managed encryption keys (SSE-S3)

Server-side encryption protects data at rest. Amazon S3 encrypts each object with a unique key. As an additional safeguard, it encrypts the key itself with a key that it rotates regularly. Amazon S3 server-side encryption uses one of the strongest block ciphers available to encrypt your data, 256-bit Advanced Encryption Standard (AES-256).

There are no additional fees for using server-side encryption with Amazon S3-managed keys (SSE-S3). However, requests to configure the default encryption feature incur standard Amazon S3 request charges. For information about pricing, see [Amazon S3 pricing](#).

If you need server-side encryption for all of the objects that are stored in a bucket, use a bucket policy. For example, the following bucket policy denies permissions to upload an object unless the request includes the `x-amz-server-side-encryption` header to request server-side encryption:

```
{  
    "Version": "2012-10-17",  
    "Id": "PutObjectPolicy",  
    "Statement": [  
        {  
            "Sid": "DenyIncorrectEncryptionHeader",  
            "Effect": "Deny",  
            "Principal": "*",  
            "Action": "s3:PutObject",  
            "Resource": "arn:aws:s3:::awsexamplebucket1/*",  
            "Condition": {  
                "StringNotEquals": {  
                    "s3:x-amz-server-side-encryption": "AES256"  
                }  
            }  
        },  
        {  
            "Sid": "DenyUnencryptedObjectUploads",  
            "Effect": "Deny",  
            "Principal": "*",  
            "Action": "s3:PutObject",  
            "Resource": "arn:aws:s3:::awsexamplebucket1/*",  
            "Condition": {  
                "Null": {  
                    "s3:x-amz-server-side-encryption": "true"  
                }  
            }  
        }  
    ]  
}
```

Note

- Server-side encryption encrypts only the object data, not object metadata.

API Support for Server-Side Encryption

To request server-side encryption using the object creation REST APIs, provide the `x-amz-server-side-encryption` request header. For information about the REST APIs, see [Using the REST API \(p. 357\)](#).

The following Amazon S3 APIs support this header:

- PUT operations—Specify the request header when uploading data using the PUT API. For more information, see [PUT Object](#).
- Initiate Multipart Upload—Specify the header in the initiate request when uploading large objects using the multipart upload API . For more information, see [Initiate Multipart Upload](#).
- COPY operations—When you copy an object, you have both a source object and a target object. For more information, see [PUT Object - Copy](#).

Note

When using a POST operation to upload an object, instead of providing the request header, you provide the same information in the form fields. For more information, see [POST Object](#).

The AWS SDKs also provide wrapper APIs that you can use to request server-side encryption. You can also use the AWS Management Console to upload objects and request server-side encryption.

Topics

- [Specifying Amazon S3 encryption \(p. 357\)](#)

Specifying Amazon S3 encryption

When you create an object, you can specify the use of server-side encryption with Amazon S3-managed encryption keys to encrypt your data. This is true when you are either uploading a new object or copying an existing object. This encryption is known as SSE-S3.

You can specify SSE-S3 using the S3 console, REST APIs, AWS SDKs, and AWS CLI. For more information, see the topics below.

For a sample of how to copy an object without encryption, see [Copying objects \(p. 201\)](#).

Using the S3 console

This topic describes how to set or change the type of encryption an object using the AWS Management Console. When you copy an object using the console, it copies the object as is. That means if the source is encrypted, the target object is also encrypted. The console also allows you to add or change encryption for an object.

Note

If you change an object's encryption, a new object is created to replace the old one. If S3 Versioning is enabled, a new version of the object is created, and the existing object becomes an older version. The role that changes the property also becomes the owner of the new object or (object version).

To add or change encryption for 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 **Buckets** list, choose the name of the bucket that contains the object.
3. In the **Objects** list, choose the name of the object that you want to add or change encryption for.

The **Object overview** opens, displaying the properties for your object.

4. Under **Server-side encryption settings**, choose **Edit**.

The **Edit server-side encryption** page opens.

5. To enable server-side encryption for your object, under **Server-side encryption**, choose **Enable**.
6. To enable server-side encryption using an Amazon S3-managed key, under **Encryption key type**, choose **Amazon S3 key (SSE-S3)**.

For more information about using Amazon S3 server-side encryption to encrypt your data, see [Protecting data using server-side encryption with Amazon S3-managed encryption keys \(SSE-S3\) \(p. 355\)](#).

7. Choose **Save changes**.

Note

This action applies encryption to all specified objects. When encrypting folders, wait for the save operation to finish before adding new objects to the folder.

Using the REST API

At the time of object creation—that is, when you are uploading a new object or making a copy of an existing object—you can specify if you want Amazon S3 to encrypt your data by adding the `x-amz-server-side-encryption` header to the request. Set the value of the header to the encryption algorithm `AES256` that Amazon S3 supports. Amazon S3 confirms that your object is stored using server-side encryption by returning the response header `x-amz-server-side-encryption`.

The following REST upload APIs accept the `x-amz-server-side-encryption` request header.

- [PUT Object](#)
- [PUT Object - Copy](#)
- [POST Object](#)
- [Initiate Multipart Upload](#)

When uploading large objects using the multipart upload API, you can specify server-side encryption by adding the `x-amz-server-side-encryption` header to the Initiate Multipart Upload request. When you are copying an existing object, regardless of whether the source object is encrypted or not, the destination object is not encrypted unless you explicitly request server-side encryption.

The response headers of the following REST APIs return the `x-amz-server-side-encryption` header when an object is stored using server-side encryption.

- [PUT Object](#)
- [PUT Object - Copy](#)
- [POST Object](#)
- [Initiate Multipart Upload](#)
- [Upload Part](#)
- [Upload Part - Copy](#)
- [Complete Multipart Upload](#)
- [Get Object](#)
- [Head Object](#)

Note

Encryption request headers should not be sent for `GET` requests and `HEAD` requests if your object uses SSE-S3 or you'll get an HTTP 400 BadRequest error.

Using the AWS SDKs

When using AWS SDKs, you can request Amazon S3 to use Amazon S3-managed encryption keys. This section provides examples of using the AWS SDKs in multiple languages. For information about other SDKs, go to [Sample Code and Libraries](#).

Java

When you use the AWS SDK for Java to upload an object, you can use server-side encryption to encrypt it. To request server-side encryption, use the `ObjectMetadata` property of the `PutObjectRequest` to set the `x-amz-server-side-encryption` request header. When you call the `putObject()` method of the `AmazonS3Client`, Amazon S3 encrypts and saves the data.

You can also request server-side encryption when uploading objects with the multipart upload API:

- When using the high-level multipart upload API, you use the `TransferManager` methods to apply server-side encryption to objects as you upload them. You can use any of the upload methods that take `ObjectMetadata` as a parameter. For more information, see [Uploading an object using multipart upload \(p. 174\)](#).
- When using the low-level multipart upload API, you specify server-side encryption when you initiate the multipart upload. You add the `ObjectMetadata` property by calling the `InitiateMultipartUploadRequest.setObjectMetadata()` method. For more information, see [Using the AWS SDKs \(low-level-level API\) \(p. 180\)](#).

You can't directly change the encryption state of an object (encrypting an unencrypted object or decrypting an encrypted object). To change an object's encryption state, you make a copy of the

object, specifying the desired encryption state for the copy, and then delete the original object. Amazon S3 encrypts the copied object only if you explicitly request server-side encryption. To request encryption of the copied object through the Java API, use the `ObjectMetadata` property to specify server-side encryption in the `CopyObjectRequest`.

Example Example

The following example shows how to set server-side encryption using the AWS SDK for Java. It shows how to perform the following tasks:

- Upload a new object using server-side encryption.
- Change an object's encryption state (in this example, encrypting a previously unencrypted object) by making a copy of the object.
- Check the encryption state of the object.

For more information about server-side encryption, see [Using the REST API \(p. 357\)](#). For instructions on creating and testing a working sample, see [Testing the Amazon S3 Java Code Examples \(p. 1192\)](#).

```
import com.amazonaws.AmazonServiceException;
import com.amazonaws.SdkClientException;
import com.amazonaws.auth.profile.ProfileCredentialsProvider;
import com.amazonaws.regions.Regions;
import com.amazonaws.services.s3.AmazonS3;
import com.amazonaws.services.s3.AmazonS3ClientBuilder;
import com.amazonaws.services.s3.internal.SSEResultBase;
import com.amazonaws.services.s3.model.*;

import java.io.ByteArrayInputStream;

public class SpecifyServerSideEncryption {

    public static void main(String[] args) {
        Regions clientRegion = Regions.DEFAULT_REGION;
        String bucketName = "*** Bucket name ***";
        String keyNameToEncrypt = "*** Key name for an object to upload and encrypt ***";
        String keyNameToCopyAndEncrypt = "*** Key name for an unencrypted object to be encrypted by copying ***";
        String copiedObjectKeyName = "*** Key name for the encrypted copy of the unencrypted object ***";

        try {
            AmazonS3 s3Client = AmazonS3ClientBuilder.standard()
                .withRegion(clientRegion)
                .withCredentials(new ProfileCredentialsProvider())
                .build();

            // Upload an object and encrypt it with SSE.
            uploadObjectWithSSEEncryption(s3Client, bucketName, keyNameToEncrypt);

            // Upload a new unencrypted object, then change its encryption state
            // to encrypted by making a copy.
            changeSSEEncryptionStatusByCopying(s3Client,
                bucketName,
                keyNameToCopyAndEncrypt,
                copiedObjectKeyName);
        } catch (AmazonServiceException e) {
            // The call was transmitted successfully, but Amazon S3 couldn't process
            // it, so it returned an error response.
            e.printStackTrace();
        }
    }
}
```

```

        } catch (SdkClientException e) {
            // Amazon S3 couldn't be contacted for a response, or the client
            // couldn't parse the response from Amazon S3.
            e.printStackTrace();
        }
    }

    private static void uploadObjectWithSSEEncryption(AmazonS3 s3Client, String
bucketName, String keyName) {
    String objectContent = "Test object encrypted with SSE";
    byte[] objectBytes = objectContent.getBytes();

    // Specify server-side encryption.
    ObjectMetadata objectMetadata = new ObjectMetadata();
    objectMetadata.setContentLength(objectBytes.length);
    objectMetadata.setSSEAlgorithm(ObjectMetadata.AES_256_SERVER_SIDE_ENCRYPTION);
    PutObjectRequest putRequest = new PutObjectRequest(bucketName,
        keyName,
        new ByteArrayInputStream(objectBytes),
        objectMetadata);

    // Upload the object and check its encryption status.
    PutObjectResult putResult = s3Client.putObject(putRequest);
    System.out.println("Object \"" + keyName + "\" uploaded with SSE.");
    printEncryptionStatus(putResult);
}

private static void changeSSEEncryptionStatusByCopying(AmazonS3 s3Client,
    String bucketName,
    String sourceKey,
    String destKey) {
    // Upload a new, unencrypted object.
    PutObjectResult putResult = s3Client.putObject(bucketName, sourceKey, "Object
example to encrypt by copying");
    System.out.println("Unencrypted object \"" + sourceKey + "\" uploaded.");
    printEncryptionStatus(putResult);

    // Make a copy of the object and use server-side encryption when storing the
    // copy.
    CopyObjectRequest request = new CopyObjectRequest(bucketName,
        sourceKey,
        bucketName,
        destKey);
    ObjectMetadata objectMetadata = new ObjectMetadata();
    objectMetadata.setSSEAlgorithm(ObjectMetadata.AES_256_SERVER_SIDE_ENCRYPTION);
    request.setNewObjectMetadata(objectMetadata);

    // Perform the copy operation and display the copy's encryption status.
    CopyObjectResult response = s3Client.copyObject(request);
    System.out.println("Object \"" + destKey + "\" uploaded with SSE.");
    printEncryptionStatus(response);

    // Delete the original, unencrypted object, leaving only the encrypted copy in
    // Amazon S3.
    s3Client.deleteObject(bucketName, sourceKey);
    System.out.println("Unencrypted object \"" + sourceKey + "\" deleted.");
}

private static void printEncryptionStatus(SSEResultBase response) {
    String encryptionStatus = response.getSSEAlgorithm();
    if (encryptionStatus == null) {
        encryptionStatus = "Not encrypted with SSE";
    }
    System.out.println("Object encryption status is: " + encryptionStatus);
}
}

```

.NET

When you upload an object, you can direct Amazon S3 to encrypt it. To change the encryption state of an existing object, you make a copy of the object and delete the source object. By default, the copy operation encrypts the target only if you explicitly request server-side encryption of the target object. To specify server-side encryption in the `CopyObjectRequest`, add the following:

```
ServerSideEncryptionMethod = ServerSideEncryptionMethod.AES256
```

For a working sample of how to copy an object, see [Using the AWS SDKs \(p. 204\)](#).

The following example uploads an object. In the request, the example directs Amazon S3 to encrypt the object. The example then retrieves object metadata and verifies the encryption method that was used. For information about creating and testing a working sample, see [Running the Amazon S3 .NET Code Examples \(p. 1193\)](#).

```
using Amazon;
using Amazon.S3;
using Amazon.S3.Model;
using System;
using System.Threading.Tasks;

namespace Amazon.DocSamples.S3
{
    class SpecifyServerSideEncryptionTest
    {
        private const string bucketName = "*** bucket name ***";
        private const string keyName = "*** key name for object created ***";
        // Specify your bucket region (an example region is shown).
        private static readonly RegionEndpoint bucketRegion = RegionEndpoint.USWest2;
        private static IAmazonS3 client;

        public static void Main()
        {
            client = new AmazonS3Client(bucketRegion);
            WritingAnObjectAsync().Wait();
        }

        static async Task WritingAnObjectAsync()
        {
            try
            {
                var putRequest = new PutObjectRequest
                {
                    BucketName = bucketName,
                    Key = keyName,
                    ContentBody = "sample text",
                    ServerSideEncryptionMethod = ServerSideEncryptionMethod.AES256
                };

                var putResponse = await client.PutObjectAsync(putRequest);

                // Determine the encryption state of an object.
                GetObjectMetadataRequest metadataRequest = new GetObjectMetadataRequest
                {
                    BucketName = bucketName,
                    Key = keyName
                };
                GetObjectMetadataResponse response = await
client.GetObjectMetadataAsync(metadataRequest);
```

```
    ServerSideEncryptionMethod objectEncryption =
    response.ServerSideEncryptionMethod;

    Console.WriteLine("Encryption method used: {0}",
    objectEncryption.ToString());
    }
    catch (AmazonS3Exception e)
    {
        Console.WriteLine("Error encountered ***. Message:'{0}' when writing an
object", e.Message);
    }
    catch (Exception e)
    {
        Console.WriteLine("Unknown encountered on server. Message:'{0}' when
writing an object", e.Message);
    }
}
}
```

PHP

This topic shows how to use classes from version 3 of the AWS SDK for PHP to add server-side encryption to objects that you upload to Amazon Simple Storage Service (Amazon S3). It assumes that you are already following the instructions for [Using the AWS SDK for PHP and Running PHP Examples \(p. 1193\)](#) and have the AWS SDK for PHP properly installed.

To upload an object to Amazon S3, use the [Aws\S3\S3Client::putObject\(\)](#) method. To add the `x-amz-server-side-encryption` request header to your upload request, specify the `ServerSideEncryption` parameter with the value `AES256`, as shown in the following code example. For information about server-side encryption requests, see [Using the REST API \(p. 357\)](#).

```
require 'vendor/autoload.php';

use Aws\S3\S3Client;

$bucket = '*** Your Bucket Name ***';
$keyname = '*** Your Object Key ***';

// $filepath should be an absolute path to a file on disk.
$filepath = '*** Your File Path ***';

$s3 = new S3Client([
    'version' => 'latest',
    'region'  => 'us-east-1'
]);

// Upload a file with server-side encryption.
$result = $s3->putObject([
    'Bucket'           => $bucket,
    'Key'              => $keyname,
    'SourceFile'       => $filepath,
    'ServerSideEncryption' => 'AES256',
]);
```

In response, Amazon S3 returns the `x-amz-server-side-encryption` header with the value of the encryption algorithm that was used to encrypt your object's data.

When you upload large objects using the multipart upload API, you can specify server-side encryption for the objects that you are uploading, as follows:

- When using the low-level multipart upload API, specify server-side encryption when you call the [Aws\S3\S3Client::createMultipartUpload\(\)](#) method. To add the `x-amz-server-`

side-encryption request header to your request, specify the `array` parameter's `ServerSideEncryption` key with the value `AES256`. For more information about the low-level multipart upload API, see [Using the AWS SDKs \(low-level API\) \(p. 180\)](#).

- When using the high-level multipart upload API, specify server-side encryption using the `ServerSideEncryption` parameter of the [CreateMultipartUpload](#) method. For an example of using the `setOption()` method with the high-level multipart upload API, see [Uploading an object using multipart upload \(p. 174\)](#).

To determine the encryption state of an existing object, retrieve the object metadata by calling the [Aws\S3\S3Client::headObject\(\)](#) method as shown in the following PHP code example.

```
require 'vendor/autoload.php';

use Aws\S3\S3Client;

$bucket = '*** Your Bucket Name ***';
$keyname = '*** Your Object Key ***';

$s3 = new S3Client([
    'version' => 'latest',
    'region'  => 'us-east-1'
]);

// Check which server-side encryption algorithm is used.
$result = $s3->headObject([
    'Bucket' => $bucket,
    'Key'     => $keyname,
]);
echo $result['ServerSideEncryption'];
```

To change the encryption state of an existing object, make a copy of the object using the [Aws\S3\S3Client::copyObject\(\)](#) method and delete the source object. By default, `copyObject()` does not encrypt the target unless you explicitly request server-side encryption of the destination object using the `ServerSideEncryption` parameter with the value `AES256`. The following PHP code example makes a copy of an object and adds server-side encryption to the copied object.

```
require 'vendor/autoload.php';

use Aws\S3\S3Client;

$sourceBucket = '*** Your Source Bucket Name ***';
$sourceKeyname = '*** Your Source Object Key ***';

$targetBucket = '*** Your Target Bucket Name ***';
$targetKeyname = '*** Your Target Object Key ***';

$s3 = new S3Client([
    'version' => 'latest',
    'region'  => 'us-east-1'
]);

// Copy an object and add server-side encryption.
$s3->copyObject([
    'Bucket'           => $targetBucket,
    'Key'              => $targetKeyname,
    'CopySource'       => "{$sourceBucket}/{$sourceKeyname}",
    'ServerSideEncryption' => 'AES256',
]);
```

- [AWS SDK for PHP for Amazon S3 Aws\S3\S3Client Class](#)

- [AWS SDK for PHP Documentation](#)

Ruby

When using the AWS SDK for Ruby to upload an object, you can specify that the object be stored encrypted at rest with server-side encryption (SSE). When you read the object back, it is automatically decrypted.

The following AWS SDK for Ruby – Version 3 example demonstrates how to specify that a file uploaded to Amazon S3 be encrypted at rest.

```
require "aws-sdk-s3"

# Wraps Amazon S3 object actions.
class ObjectPutSseWrapper
    attr_reader :object

    # @param object [Aws::S3::Object] An existing Amazon S3 object.
    def initialize(object)
        @object = object
    end

    def put_object_encrypted(object_content, encryption)
        @object.put(body: object_content, server_side_encryption: encryption)
        true
    rescue Aws::Errors::ServiceError => e
        puts "Couldn't put your content to #{object.key}. Here's why: #{e.message}"
        false
    end
end

def run_demo
    bucket_name = "doc-example-bucket"
    object_key = "my-encrypted-content"
    object_content = "This is my super-secret content."
    encryption = "AES256"

    wrapper = ObjectPutSseWrapper.new(Aws::S3::Object.new(bucket_name, object_content))
    return unless wrapper.put_object_encrypted(object_content, encryption)

    puts "Put your content into #{bucket_name}:#{object_key} and encrypted it with
#{encryption}."
end

run_demo if $PROGRAM_NAME == __FILE__
```

For an example that shows how to upload an object without SSE, see [Uploading objects \(p. 158\)](#).

The following code example demonstrates how to determine the encryption state of an existing object.

```
require "aws-sdk-s3"

# Wraps Amazon S3 object actions.
class ObjectGetEncryptionWrapper
    attr_reader :object

    # @param object [Aws::S3::Object] An existing Amazon S3 object.
    def initialize(object)
        @object = object
    end

    # Gets the object into memory.
```

```

#
# @return [Aws::S3::Types::GetObjectOutput, nil] The retrieved object data if
# successful; otherwise nil.
def get_object
  @object.get
rescue Aws::Errors::ServiceError => e
  puts "Couldn't get object #{@object.key}. Here's why: #{e.message}"
end
end

# Replace bucket name and object key with an existing bucket and object that you own.
def run_demo
  bucket_name = "doc-example-bucket"
  object_key = "my-object.txt"

  wrapper = ObjectGetEncryptionWrapper.new(Aws::S3::Object.new(bucket_name,
object_key))
  obj_data = wrapper.get_object
  return unless obj_data

  encryption = obj_data.server_side_encryption.nil? ? "no" :
obj_data.server_side_encryption
  puts "Object #{object_key} uses #{encryption} encryption."
end

run_demo if $PROGRAM_NAME == __FILE__

```

If server-side encryption is not used for the object that is stored in Amazon S3, the method returns null.

To change the encryption state of an existing object, make a copy of the object and delete the source object. By default, the copy methods do not encrypt the target unless you explicitly request server-side encryption. You can request the encryption of the target object by specifying the `server_side_encryption` value in the options hash argument as shown in the following Ruby code example. The code example demonstrates how to copy an object and encrypt the copy.

```

require "aws-sdk-s3"

# Wraps Amazon S3 object actions.
class ObjectCopyEncryptWrapper
  attr_reader :source_object

  # @param source_object [Aws::S3::Object] An existing Amazon S3 object. This is used
  # as the source object for
  #                                         copy actions.
  def initialize(source_object)
    @source_object = source_object
  end

  # Copy the source object to the specified target bucket, rename it with the target
  # key, and encrypt it.
  #
  # @param target_bucket [Aws::S3::Bucket] An existing Amazon S3 bucket where the
  # object is copied.
  # @param target_object_key [String] The key to give the copy of the object.
  # @return [Aws::S3::Object, nil] The copied object when successful; otherwise, nil.
  def copy_object(target_bucket, target_object_key, encryption)
    @source_object.copy_to(bucket: target_bucket.name, key: target_object_key,
server_side_encryption: encryption)
    target_bucket.object(target_object_key)
    rescue Aws::Errors::ServiceError => e
      puts "Couldn't copy #{@source_object.key} to #{target_object_key}. Here's why:
#{e.message}"
  end

```

```
end

# Replace the source and target bucket names with existing buckets you own and replace
# the source object key
# with an existing object in the source bucket.
def run_demo
  source_bucket_name = "doc-example-bucket1"
  source_key = "my-source-file.txt"
  target_bucket_name = "doc-example-bucket2"
  target_key = "my-target-file.txt"
  target_encryption = "AES256"

  source_bucket = Aws::S3::Bucket.new(source_bucket_name)
  wrapper = ObjectCopyEncryptWrapper.new(source_bucket.object(source_key))
  target_bucket = Aws::S3::Bucket.new(target_bucket_name)
  target_object = wrapper.copy_object(target_bucket, target_key, target_encryption)
  return unless target_object

  puts "Copied #{source_key} from #{source_bucket_name} to
#{target_object.bucket_name}:#{target_object.key} and \"\
      \"encrypted the target with #{target_object.server_side_encryption} encryption."
end

run_demo if $PROGRAM_NAME == __FILE__
```

Using the AWS CLI

To specify SSE-S3 when you upload an object using the AWS CLI, use the following example.

```
aws s3api put-object --bucket DOC-EXAMPLE-BUCKET1 --key object-key-name --server-side-
encryption AES256 --body file path
```

For more information, see [put-object](#) in the *AWS CLI reference*. To specify SSE-S3 when you copy an object using the AWS CLI, see [copy-object](#).

Using AWS CloudFormation

For examples of setting up encryption using AWS CloudFormation, see [Create a bucket with default encryption](#) and [Create a bucket using AWS KMS server-side encryption with an S3 Bucket Key](#) in the *AWS CloudFormation User Guide*.

Protecting data using server-side encryption with customer-provided encryption keys (SSE-C)

Server-side encryption is about protecting data at rest. Server-side encryption encrypts only the object data, not object metadata. Using server-side encryption with customer-provided encryption keys (SSE-C) allows you to set your own encryption keys. With the encryption key you provide as part of your request, Amazon S3 manages the encryption as it writes to disks and decryption when you access your objects. Therefore, you don't need to maintain any code to perform data encryption and decryption. The only thing you do is manage the encryption keys you provide.

When you upload an object, Amazon S3 uses the encryption key you provide to apply AES-256 encryption to your data and removes the encryption key from memory. When you retrieve an object, you must provide the same encryption key as part of your request. Amazon S3 first verifies that the encryption key you provided matches and then decrypts the object before returning the object data to you.

There are no new charges for using server-side encryption with customer-provided encryption keys (SSE-C). However, requests to configure and use SSE-C incur standard Amazon S3 request charges. For information about pricing, see [Amazon S3 pricing](#).

Important

Amazon S3 does not store the encryption key you provide. Instead, it stores a randomly salted HMAC value of the encryption key to validate future requests. The salted HMAC value cannot be used to derive the value of the encryption key or to decrypt the contents of the encrypted object. That means if you lose the encryption key, you lose the object.

For more information about SSE-C, see the following topics.

Topics

- [SSE-C overview \(p. 367\)](#)
- [Requiring and restricting SSE-C \(p. 367\)](#)
- [Presigned URLs and SSE-C \(p. 368\)](#)
- [Specifying server-side encryption with customer-provided keys \(SSE-C\) \(p. 368\)](#)

SSE-C overview

This section provides an overview of SSE-C:

- You must use HTTPS.

Important

Amazon S3 rejects any requests made over HTTP when using SSE-C. For security considerations, we recommend that you consider any key you erroneously send using HTTP to be compromised. You should discard the key and rotate as appropriate.

- The ETag in the response is not the MD5 of the object data.
- You manage a mapping of which encryption key was used to encrypt which object. Amazon S3 does not store encryption keys. You are responsible for tracking which encryption key you provided for which object.
 - If your bucket is versioning-enabled, each object version you upload using this feature can have its own encryption key. You are responsible for tracking which encryption key was used for which object version.
 - Because you manage encryption keys on the client side, you manage any additional safeguards, such as key rotation, on the client side.

Warning

If you lose the encryption key, any GET request for an object without its encryption key fails, and you lose the object.

Requiring and restricting SSE-C

To require server-side encryption with customer provided keys (SSE-C) for all objects in a particular Amazon S3 bucket, you can use a bucket policy.

For example, the following bucket policy denies upload object (`s3:PutObject`) permissions for all requests that don't include the `x-amz-server-side-encryption-customer-algorithm` header requesting SSE-C.

```
{  
    "Version": "2012-10-17",  
    "Id": "PutObjectPolicy",  
    "Statement": [  
        {  
            "Sid": "RequireSSECOBJECTUploads",  
            "Effect": "Deny",  
            "Principal": "*",  
            "Action": "s3:PutObject",  
            "Resource": "arn:aws:s3:::mybucket/*",  
            "Condition": {"StringNotEquals": {"x-amz-server-side-encryption-customer-algorithm": "AES256"}, "StringNotEquals": {"x-amz-server-side-encryption-customer-key": ""}}  
        }  
    ]  
}
```

```
        "Action": "s3:PutObject",
        "Resource": "arn:aws:s3:::DOC-EXAMPLE-BUCKET/*",
        "Condition": {
            "Null": {
                "s3:x-amz-server-side-encryption-customer-algorithm": "true"
            }
        }
    ]
}
```

To restrict server-side encryption of all objects in a particular Amazon S3 bucket, you can also use a policy.

For example, the following bucket policy denies the upload object (`s3:PutObject`) permission to everyone if the request includes the `x-amz-server-side-encryption-customer-algorithm` header requesting server-side encryption with customer provided keys (SSE-C).

```
{
    "Version": "2012-10-17",
    "Id": "PutObjectPolicy",
    "Statement": [
        {
            "Sid": "RestrictSSECOBJECTUploads",
            "Effect": "Deny",
            "Principal": "*",
            "Action": "s3:PutObject",
            "Resource": "arn:aws:s3:::DOC-EXAMPLE-BUCKET/*",
            "Condition": {
                "Null": {
                    "s3:x-amz-server-side-encryption-customer-algorithm": "false"
                }
            }
        }
    ]
}
```

Presigned URLs and SSE-C

You can generate a presigned URL that can be used for operations such as upload a new object, retrieve an existing object, or object metadata. Presigned URLs support the SSE-C as follows:

- When creating a presigned URL, you must specify the algorithm using the `x-amz-server-side-encryption-customer-algorithm` in the signature calculation.
- When using the presigned URL to upload a new object, retrieve an existing object, or retrieve only object metadata, you must provide all the encryption headers in your client application.

Note

For non-SSE-C objects, you can generate a presigned URL and directly paste that into a browser, for example to access the data.

However, this is not true for SSE-C objects because in addition to the presigned URL, you also need to include HTTP headers that are specific to SSE-C objects. Therefore, you can use the presigned URL for SSE-C objects only programmatically.

Specifying server-side encryption with customer-provided keys (SSE-C)

At the time of object creation with the REST API, you can specify server-side encryption with customer-provided encryption keys (SSE-C). When you use SSE-C, you must provide encryption key information using the following request headers.

Name	Description
<code>x-amz-server-side-encryption-customer-algorithm</code>	Use this header to specify the encryption algorithm. The header value must be "AES256".
<code>x-amz-server-side-encryption-customer-key</code>	Use this header to provide the 256-bit, base64-encoded encryption key for Amazon S3 to use to encrypt or decrypt your data.
<code>x-amz-server-side-encryption-customer-key-MD5</code>	Use this header to provide the base64-encoded 128-bit MD5 digest of the encryption key according to RFC 1321 . Amazon S3 uses this header for a message integrity check to ensure that the encryption key was transmitted without error.

You can use AWS SDK wrapper libraries to add these headers to your request. If you need to, you can make the Amazon S3 REST API calls directly in your application.

Note

You cannot use the Amazon S3 console to upload an object and request SSE-C. You also cannot use the console to update (for example, change the storage class or add metadata) an existing object stored using SSE-C.

Using the REST API

Amazon S3 rest APIs that support SSE-C

The following Amazon S3 APIs support server-side encryption with customer-provided encryption keys (SSE-C).

- **GET operation** — When retrieving objects using the GET API (see [GET Object](#)), you can specify the request headers.
- **HEAD operation** — To retrieve object metadata using the HEAD API (see [HEAD Object](#)), you can specify these request headers.
- **PUT operation** — When uploading data using the PUT Object API (see [PUT Object](#)), you can specify these request headers.
- **Multipart Upload** — When uploading large objects using the multipart upload API, you can specify these headers. You specify these headers in the initiate request (see [Initiate Multipart Upload](#)) and each subsequent part upload request (see [Upload Part](#) or [Upload Part - Copy](#))

[Upload Part - Copy](#)

-). For each part upload request, the encryption information must be the same as what you provided in the initiate multipart upload request.
- **POST operation** — When using a POST operation to upload an object (see [POST Object](#)), instead of the request headers, you provide the same information in the form fields.
- **Copy operation** — When you copy an object (see [PUT Object - Copy](#)), you have both a source object and a target object:
 - If you want the target object encrypted using server-side encryption with AWS managed keys, you must provide the `x-amz-server-side-encryption` request header.
 - If you want the target object encrypted using SSE-C, you must provide encryption information using the three headers described in the preceding table.
 - If the source object is encrypted using SSE-C, you must provide encryption key information using the following headers so that Amazon S3 can decrypt the object for copying.

Name	Description
x-amz-copy-source-server-side-encryption-customer-algorithm	Include this header to specify the algorithm Amazon S3 should use to decrypt the source object. This value must be <code>AES256</code> .
x-amz-copy-source-server-side-encryption-customer-key	Include this header to provide the base64-encoded encryption key for Amazon S3 to use to decrypt the source object. This encryption key must be the one that you provided Amazon S3 when you created the source object. Otherwise, Amazon S3 cannot decrypt the object.
x-amz-copy-source-server-side-encryption-customer-key-MD5	Include this header to provide the base64-encoded 128-bit MD5 digest of the encryption key according to RFC 1321 .

Using the AWS SDKs to specify SSE-C for PUT, GET, Head, and Copy operations

The following examples show how to request server-side encryption with customer-provided keys (SSE-C) for objects. The examples perform the following operations. Each operation shows how to specify SSE-C-related headers in the request:

- **Put object**—Uploads an object and requests server-side encryption using a customer-provided encryption key.
- **Get object**—Downloads the object uploaded in the previous step. In the request, you provide the same encryption information you provided when you uploaded the object. Amazon S3 needs this information to decrypt the object so that it can return it to you.
- **Get object metadata**—Retrieves the object's metadata. You provide the same encryption information used when the object was created.
- **Copy object**—Makes a copy of the previously-uploaded object. Because the source object is stored using SSE-C, you must provide its encryption information in your copy request. By default, Amazon S3 encrypts the copy of the object only if you explicitly request it. This example directs Amazon S3 to store an encrypted copy of the object.

Java

Note

This example shows how to upload an object in a single operation. When using the Multipart Upload API to upload large objects, you provide encryption information in the same way shown in this example. For examples of multipart uploads that use the AWS SDK for Java, see [Uploading an object using multipart upload \(p. 174\)](#).

To add the required encryption information, you include an `SSECustomerKey` in your request. For more information about the `SSECustomerKey` class, see the REST API section.

For information about SSE-C, see [Protecting data using server-side encryption with customer-provided encryption keys \(SSE-C\) \(p. 366\)](#). For instructions on creating and testing a working sample, see [Testing the Amazon S3 Java Code Examples \(p. 1192\)](#).

Example

```
import com.amazonaws.AmazonServiceException;
import com.amazonaws.SdkClientException;
import com.amazonaws.auth.profile.ProfileCredentialsProvider;
```

```
import com.amazonaws.regions.Regions;
import com.amazonaws.services.s3.AmazonS3;
import com.amazonaws.services.s3.AmazonS3ClientBuilder;
import com.amazonaws.services.s3.model.*;

import javax.crypto.KeyGenerator;
import java.io.BufferedReader;
import java.io.File;
import java.io.IOException;
import java.io.InputStreamReader;
import java.security.NoSuchAlgorithmException;
import java.security.SecureRandom;

public class ServerSideEncryptionUsingClientSideEncryptionKey {
    private static SSECustomerKey SSE_KEY;
    private static AmazonS3 S3_CLIENT;
    private static KeyGenerator KEY_GENERATOR;

    public static void main(String[] args) throws IOException, NoSuchAlgorithmException
    {
        Regions clientRegion = Regions.DEFAULT_REGION;
        String bucketName = "**** Bucket name ****";
        String keyName = "**** Key name ****";
        String uploadFileName = "**** File path ****";
        String targetKeyName = "**** Target key name ****";

        // Create an encryption key.
        KEY_GENERATOR = KeyGenerator.getInstance("AES");
        KEY_GENERATOR.init(256, new SecureRandom());
        SSE_KEY = new SSECustomerKey(KEY_GENERATOR.generateKey());

        try {
            S3_CLIENT = AmazonS3ClientBuilder.standard()
                .withCredentials(new ProfileCredentialsProvider())
                .withRegion(clientRegion)
                .build();

            // Upload an object.
            uploadObject(bucketName, keyName, new File(uploadFileName));

            // Download the object.
            downloadObject(bucketName, keyName);

            // Verify that the object is properly encrypted by attempting to retrieve
            it
            // using the encryption key.
            retrieveObjectMetadata(bucketName, keyName);

            // Copy the object into a new object that also uses SSE-C.
            copyObject(bucketName, keyName, targetKeyName);
        } catch (AmazonServiceException e) {
            // The call was transmitted successfully, but Amazon S3 couldn't process
            // it, so it returned an error response.
            e.printStackTrace();
        } catch (SdkClientException e) {
            // Amazon S3 couldn't be contacted for a response, or the client
            // couldn't parse the response from Amazon S3.
            e.printStackTrace();
        }
    }

    private static void uploadObject(String bucketName, String keyName, File file) {
        PutObjectRequest putRequest = new PutObjectRequest(bucketName, keyName,
file).withSSECustomerKey(SSE_KEY);
        S3_CLIENT.putObject(putRequest);
        System.out.println("Object uploaded");
    }
}
```

```

        }

        private static void downloadObject(String bucketName, String keyName) throws
IOException {
    GetObjectRequest getObjectRequest = new GetObjectRequest(bucketName,
keyName).withSSECustomerKey(SSE_KEY);
    S3Object object = S3_CLIENT.getObject(getObjectRequest);

    System.out.println("Object content: ");
    displayTextInputStream(object.getObjectContent());
}

private static void retrieveObjectMetadata(String bucketName, String keyName) {
    GetObjectMetadataRequest getMetadataRequest = new
GetObjectMetadataRequest(bucketName, keyName)
        .withSSECustomerKey(SSE_KEY);
    ObjectMetadata objectMetadata =
S3_CLIENT.getObjectMetadata(getMetadataRequest);
    System.out.println("Metadata retrieved. Object size: " +
objectMetadata.getContentLength());
}

private static void copyObject(String bucketName, String keyName, String
targetKeyName)
    throws NoSuchAlgorithmException {
    // Create a new encryption key for target so that the target is saved using
SSE-C.
    SSECustomerKey newSSEKey = new SSECustomerKey(KEY_GENERATOR.generateKey());

    CopyObjectRequest copyRequest = new CopyObjectRequest(bucketName, keyName,
bucketName, targetKeyName)
        .withSourceSSECustomerKey(SSE_KEY)
        .withDestinationSSECustomerKey(newSSEKey);

    S3_CLIENT.copyObject(copyRequest);
    System.out.println("Object copied");
}

private static void displayTextInputStream(S3ObjectInputStream input) throws
IOException {
    // Read one line at a time from the input stream and display each line.
    BufferedReader reader = new BufferedReader(new InputStreamReader(input));
    String line;
    while ((line = reader.readLine()) != null) {
        System.out.println(line);
    }
    System.out.println();
}
}
}

```

.NET

Note

For examples of uploading large objects using the multipart upload API, see [Uploading an object using multipart upload \(p. 174\)](#) and [Using the AWS SDKs \(low-level-level API\) \(p. 180\)](#).

For information about SSE-C, see [Protecting data using server-side encryption with customer-provided encryption keys \(SSE-C\) \(p. 366\)](#). For information about creating and testing a working sample, see [Running the Amazon S3 .NET Code Examples \(p. 1193\)](#).

Example

```
using Amazon;
```

```
using Amazon.S3;
using Amazon.S3.Model;
using System;
using System.IO;
using System.Security.Cryptography;
using System.Threading.Tasks;

namespace Amazon.DocSamples.S3
{
    class SSEClientEncryptionKeyObjectOperationsTest
    {
        private const string bucketName = "*** bucket name ***";
        private const string keyName = "*** key name for new object created ***";
        private const string copyTargetKeyName = "*** key name for object copy ***";
        // Specify your bucket region (an example region is shown).
        private static readonly RegionEndpoint bucketRegion = RegionEndpoint.USWest2;
        private static IAmazonS3 client;

        public static void Main()
        {
            client = new AmazonS3Client(bucketRegion);
            ObjectOpsUsingClientEncryptionKeyAsync().Wait();
        }
        private static async Task ObjectOpsUsingClientEncryptionKeyAsync()
        {
            try
            {
                // Create an encryption key.
                Aes aesEncryption = Aes.Create();
                aesEncryption.KeySize = 256;
                aesEncryption.GenerateKey();
                string base64Key = Convert.ToBase64String(aesEncryption.Key);

                // 1. Upload the object.
                PutObjectRequest putObjectRequest = await UploadObjectAsync(base64Key);
                // 2. Download the object and verify that its contents matches what you
                uploaded.
                await DownloadObjectAsync(base64Key, putObjectRequest);
                // 3. Get object metadata and verify that the object uses AES-256
                encryption.
                await GetObjectMetadataAsync(base64Key);
                // 4. Copy both the source and target objects using server-side
                encryption with
                //      a customer-provided encryption key.
                await CopyObjectAsync(aesEncryption, base64Key);
            }
            catch (AmazonS3Exception e)
            {
                Console.WriteLine("Error encountered ***. Message:'{0}' when writing an
object", e.Message);
            }
            catch (Exception e)
            {
                Console.WriteLine("Unknown encountered on server. Message:'{0}' when
writing an object", e.Message);
            }
        }

        private static async Task<PutObjectRequest> UploadObjectAsync(string base64Key)
        {
            PutObjectRequest putObjectRequest = new PutObjectRequest
            {
                BucketName = bucketName,
                Key = keyName,
                ContentBody = "sample text",
            }
        }
    }
}
```

```

        ServerSideEncryptionCustomerMethod =
ServerSideEncryptionCustomerMethod.AES256,
        ServerSideEncryptionCustomerProvidedKey = base64Key
    };
    PutObjectResponse putObjectResponse = await
client.PutObjectAsync(putObjectRequest);
    return putObjectRequest;
}
private static async Task DownloadObjectAsync(string base64Key,
PutObjectRequest putObjectRequest)
{
    GetObjectRequest getObjectRequest = new GetObjectRequest
    {
        BucketName = bucketName,
        Key = keyName,
        // Provide encryption information for the object stored in Amazon S3.
        ServerSideEncryptionCustomerMethod =
ServerSideEncryptionCustomerMethod.AES256,
        ServerSideEncryptionCustomerProvidedKey = base64Key
    };

    using (GetObjectResponse getResponse = await
client.GetObjectAsync(getObjectRequest))
    using (StreamReader reader = new StreamReader(getResponse.ResponseStream))
    {
        string content = reader.ReadToEnd();
        if (String.Compare(putObjectRequest.ContentBody, content) == 0)
            Console.WriteLine("Object content is same as we uploaded");
        else
            Console.WriteLine("Error...Object content is not same.");

        if (getResponse.ServerSideEncryptionCustomerMethod ==
ServerSideEncryptionCustomerMethod.AES256)
            Console.WriteLine("Object encryption method is AES256, same as we
set");
        else
            Console.WriteLine("Error...Object encryption method is not the same
as AES256 we set");

        // Assert.AreEqual(putObjectRequest.ContentBody, content);
        // Assert.AreEqual(ServerSideEncryptionCustomerMethod.AES256,
getResponse.ServerSideEncryptionCustomerMethod);
    }
}
private static async Task GetObjectMetadataAsync(string base64Key)
{
    GetObjectMetadataRequest getObjectMetadataRequest = new
GetObjectMetadataRequest
    {
        BucketName = bucketName,
        Key = keyName,

        // The object stored in Amazon S3 is encrypted, so provide the
necessary encryption information.
        ServerSideEncryptionCustomerMethod =
ServerSideEncryptionCustomerMethod.AES256,
        ServerSideEncryptionCustomerProvidedKey = base64Key
    };

    GetObjectMetadataResponse getObjectMetadataResponse = await
client.GetObjectMetadataAsync(getObjectMetadataRequest);
    Console.WriteLine("The object metadata show encryption method used is:
{0}", getObjectMetadataResponse.ServerSideEncryptionCustomerMethod);
    // Assert.AreEqual(ServerSideEncryptionCustomerMethod.AES256,
getObjectMetadataResponse.ServerSideEncryptionCustomerMethod);
}

```

```
private static async Task CopyObjectAsync(Aes aesEncryption, string base64Key)
{
    aesEncryption.GenerateKey();
    string copyBase64Key = Convert.ToBase64String(aesEncryption.Key);

    CopyObjectRequest copyRequest = new CopyObjectRequest
    {
        SourceBucket = bucketName,
        SourceKey = keyName,
        DestinationBucket = bucketName,
        DestinationKey = copyTargetKeyName,
        // Information about the source object's encryption.
        CopySourceServerSideEncryptionCustomerMethod =
            ServerSideEncryptionCustomerMethod.AES256,
        CopySourceServerSideEncryptionCustomerProvidedKey = base64Key,
        // Information about the target object's encryption.
        ServerSideEncryptionCustomerMethod =
            ServerSideEncryptionCustomerMethod.AES256,
        ServerSideEncryptionCustomerProvidedKey = copyBase64Key
    };
    await client.CopyObjectAsync(copyRequest);
}
}
```

Using the AWS SDKs to specify SSE-C for multipart uploads

The example in the preceding section shows how to request server-side encryption with customer-provided key (SSE-C) in the PUT, GET, Head, and Copy operations. This section describes other Amazon S3 APIs that support SSE-C.

Java

To upload large objects, you can use multipart upload API (see [Uploading and copying objects using multipart upload \(p. 167\)](#)). You can use either high-level or low-level APIs to upload large objects. These APIs support encryption-related headers in the request.

- When using the high-level `TransferManager` API, you provide the encryption-specific headers in the `PutObjectRequest` (see [Uploading an object using multipart upload \(p. 174\)](#)).
- When using the low-level API, you provide encryption-related information in the `InitiateMultipartUploadRequest`, followed by identical encryption information in each `UploadPartRequest`. You do not need to provide any encryption-specific headers in your `CompleteMultipartUploadRequest`. For examples, see [Using the AWS SDKs \(low-level-level API\) \(p. 180\)](#).

The following example uses `TransferManager` to create objects and shows how to provide SSE-C related information. The example does the following:

- Creates an object using the `TransferManager.upload()` method. In the `PutObjectRequest` instance, you provide encryption key information to request. Amazon S3 encrypts the object using the customer-provided encryption key.
- Makes a copy of the object by calling the `TransferManager.copy()` method. The example directs Amazon S3 to encrypt the object copy using a new `SSECUSTOMER`. Because the source object is encrypted using SSE-C, the `CopyObjectRequest` also provides the encryption key of the source object so that Amazon S3 can decrypt the object before copying it.

Example

```
import com.amazonaws.AmazonServiceException;
import com.amazonaws.SdkClientException;
import com.amazonaws.auth.profile.ProfileCredentialsProvider;
import com.amazonaws.regions.Regions;
import com.amazonaws.services.s3.AmazonS3;
import com.amazonaws.services.s3.AmazonS3ClientBuilder;
import com.amazonaws.services.s3.model.CopyObjectRequest;
import com.amazonaws.services.s3.model.PutObjectRequest;
import com.amazonaws.services.s3.model.SSECustomerKey;
import com.amazonaws.services.s3.transfer.Copy;
import com.amazonaws.services.s3.transfer.TransferManager;
import com.amazonaws.services.s3.transfer.TransferManagerBuilder;
import com.amazonaws.services.s3.transfer.Upload;

import javax.crypto.KeyGenerator;
import java.io.File;
import java.security.SecureRandom;

public class ServerSideEncryptionCopyObjectUsingHLwithSSEC {

    public static void main(String[] args) throws Exception {
        Regions clientRegion = Regions.DEFAULT_REGION;
        String bucketName = "*** Bucket name ***";
        String fileToUpload = "*** File path ***";
        String keyName = "*** New object key name ***";
        String targetKeyName = "*** Key name for object copy ***";

        try {
            AmazonS3 s3Client = AmazonS3ClientBuilder.standard()
                .withRegion(clientRegion)
                .withCredentials(new ProfileCredentialsProvider())
                .build();
            TransferManager tm = TransferManagerBuilder.standard()
                .withS3Client(s3Client)
                .build();

            // Create an object from a file.
            PutObjectRequest putObjectRequest = new PutObjectRequest(bucketName,
keyName, new File(fileToUpload));

            // Create an encryption key.
            KeyGenerator keyGenerator = KeyGenerator.getInstance("AES");
            keyGenerator.init(256, new SecureRandom());
            SSECustomerKey sseCustomerEncryptionKey = new
SSECustomerKey(keyGenerator.generateKey());

            // Upload the object. TransferManager uploads asynchronously, so this call
returns immediately.
            putObjectRequest.setSSECustomerKey(sseCustomerEncryptionKey);
            Upload upload = tm.upload(putObjectRequest);

            // Optionally, wait for the upload to finish before continuing.
            upload.waitForCompletion();
            System.out.println("Object created.");

            // Copy the object and store the copy using SSE-C with a new key.
            CopyObjectRequest copyObjectRequest = new CopyObjectRequest(bucketName,
keyName, bucketName, targetKeyName);
            SSECustomerKey sseTargetObjectEncryptionKey = new
SSECustomerKey(keyGenerator.generateKey());
            copyObjectRequest.setSourceSSECustomerKey(sseCustomerEncryptionKey);

        }
    }
}
```

```
copyObjectRequest.setDestinationSSECustomerKey(sseTargetObjectEncryptionKey);

        // Copy the object. TransferManager copies asynchronously, so this call
returns immediately.
        Copy copy = tm.copy(copyObjectRequest);

        // Optionally, wait for the upload to finish before continuing.
        copy.waitForCompletion();
        System.out.println("Copy complete.");
    } catch (AmazonServiceException e) {
        // The call was transmitted successfully, but Amazon S3 couldn't process
        // it, so it returned an error response.
        e.printStackTrace();
    } catch (SdkClientException e) {
        // Amazon S3 couldn't be contacted for a response, or the client
        // couldn't parse the response from Amazon S3.
        e.printStackTrace();
    }
}
```

.NET

To upload large objects, you can use multipart upload API (see [Uploading and copying objects using multipart upload \(p. 167\)](#)). AWS SDK for .NET provides both high-level or low-level APIs to upload large objects. These APIs support encryption-related headers in the request.

- When using high-level Transfer-Utility API, you provide the encryption-specific headers in the TransferUtilityUploadRequest as shown. For code examples, see [Uploading an object using multipart upload \(p. 174\)](#).

```
TransferUtilityUploadRequest request = new TransferUtilityUploadRequest()
{
    FilePath = filePath,
    BucketName = existingBucketName,
    Key = keyName,
    // Provide encryption information.
    ServerSideEncryptionCustomerMethod = ServerSideEncryptionCustomerMethod.AES256,
    ServerSideEncryptionCustomerProvidedKey = base64Key,
};
```

- When using the low-level API, you provide encryption-related information in the initiate multipart upload request, followed by identical encryption information in the subsequent upload part requests. You do not need to provide any encryption-specific headers in your complete multipart upload request. For examples, see [Using the AWS SDKs \(low-level-level API\) \(p. 180\)](#).

The following is a low-level multipart upload example that makes a copy of an existing large object. In the example, the object to be copied is stored in Amazon S3 using SSE-C, and you want to save the target object also using SSE-C. In the example, you do the following:

- Initiate a multipart upload request by providing an encryption key and related information.
- Provide source and target object encryption keys and related information in the CopyPartRequest.
- Obtain the size of the source object to be copied by retrieving the object metadata.
- Upload the objects in 5 MB parts.

Example

```
using Amazon;
```

```

using Amazon.S3;
using Amazon.S3.Model;
using System;
using System.Collections.Generic;
using System.IO;
using System.Security.Cryptography;
using System.Threading.Tasks;

namespace Amazon.DocSamples.S3
{
    class SSECLowLevelMPUcopyObjectTest
    {
        private const string existingBucketName = "*** bucket name ***";
        private const string sourceKeyName      = "*** source object key name ***";
        private const string targetKeyName     = "*** key name for the target object
***";
        private const string filePath          = @ "*** file path ***";
        // Specify your bucket region (an example region is shown).
        private static readonly RegionEndpoint bucketRegion = RegionEndpoint.USWest2;
        private static IAmazonS3 s3Client;
        static void Main()
        {
            s3Client = new AmazonS3Client(bucketRegion);
            CopyObjClientEncryptionKeyAsync().Wait();
        }

        private static async Task CopyObjClientEncryptionKeyAsync()
        {
            Aes aesEncryption = Aes.Create();
            aesEncryption.KeySize = 256;
            aesEncryption.GenerateKey();
            string base64Key = Convert.ToBase64String(aesEncryption.Key);

            await CreateSampleObjUsingClientEncryptionKeyAsync(base64Key, s3Client);

            await CopyObjectAsync(s3Client, base64Key);
        }
        private static async Task CopyObjectAsync(IAmazonS3 s3Client, string
base64Key)
        {
            List<CopyPartResponse> uploadResponses = new List<CopyPartResponse>();

            // 1. Initialize.
            InitiateMultipartUploadRequest initiateRequest = new
InitiateMultipartUploadRequest
            {
                BucketName = existingBucketName,
                Key = targetKeyName,
                ServerSideEncryptionCustomerMethod =
ServerSideEncryptionCustomerMethod.AES256,
                ServerSideEncryptionCustomerProvidedKey = base64Key,
            };

            InitiateMultipartUploadResponse initResponse =
await s3Client.InitiateMultipartUploadAsync(initiateRequest);

            // 2. Upload Parts.
            long partSize = 5 * (long)Math.Pow(2, 20); // 5 MB
            long firstByte = 0;
            long lastByte = partSize;

            try
            {
                // First find source object size. Because object is stored encrypted
with

```

```

    // customer provided key you need to provide encryption information
    // in your request.
    GetObjectMetadataRequest getObjectMetadataRequest = new
GetObjectMetadataRequest()
{
    BucketName = existingBucketName,
    Key = sourceKeyName,
    ServerSideEncryptionCustomerMethod =
ServerSideEncryptionCustomerMethod.AES256,
    ServerSideEncryptionCustomerProvidedKey = base64Key // " *
**source object encryption key ***"
};

    GetObjectMetadataResponse getObjectMetadataResponse = await
s3Client.GetObjectMetadataAsync(getObjectMetadataRequest);

    long filePosition = 0;
    for (int i = 1; filePosition <
getObjectMetadataResponse.ContentLength; i++)
{
    CopyPartRequest copyPartRequest = new CopyPartRequest
{
    UploadId = initResponse.UploadId,
    // Source.
    SourceBucket = existingBucketName,
    SourceKey = sourceKeyName,
    // Source object is stored using SSE-C. Provide encryption
information.
    CopySourceServerSideEncryptionCustomerMethod =
ServerSideEncryptionCustomerMethod.AES256,
    CopySourceServerSideEncryptionCustomerProvidedKey =
base64Key, //****source object encryption key ***",
    FirstByte = firstByte,
    // If the last part is smaller then our normal part size then
use the remaining size.
    LastByte = lastByte >
    getObjectMetadataResponse.ContentLength ?
        getObjectMetadataResponse.ContentLength - 1 : lastByte,
        // Target.
        DestinationBucket = existingBucketName,
        DestinationKey = targetKeyName,
        PartNumber = i,
        // Encryption information for the target object.
        ServerSideEncryptionCustomerMethod =
ServerSideEncryptionCustomerMethod.AES256,
        ServerSideEncryptionCustomerProvidedKey = base64Key
    };
    uploadResponses.Add(await
s3Client.CopyPartAsync(copyPartRequest));
    filePosition += partSize;
    firstByte += partSize;
    lastByte += partSize;
}

    // Step 3: complete.
    CompleteMultipartUploadRequest completeRequest = new
CompleteMultipartUploadRequest
{
    BucketName = existingBucketName,
    Key = targetKeyName,
    UploadId = initResponse.UploadId,
};
completeRequest.AddPartETags(uploadResponses);

    CompleteMultipartUploadResponse completeUploadResponse =

```

```
        await s3Client.CompleteMultipartUploadAsync(completeRequest);
    }
    catch (Exception exception)
    {
        Console.WriteLine("Exception occurred: {0}", exception.Message);
        AbortMultipartUploadRequest abortMPURequest = new
AbortMultipartUploadRequest
        {
            BucketName = existingBucketName,
            Key = targetKeyName,
            UploadId = initResponse.UploadId
        };
        s3Client.AbortMultipartUpload(abortMPURequest);
    }
}
private static async Task CreateSampleObjUsingClientEncryptionKeyAsync(string
base64Key, IAmazonS3 s3Client)
{
    // List to store upload part responses.
    List<UploadPartResponse> uploadResponses = new
List<UploadPartResponse>();

    // 1. Initialize.
    InitiateMultipartUploadRequest initiateRequest = new
InitiateMultipartUploadRequest
    {
        BucketName = existingBucketName,
        Key = sourceKeyName,
        ServerSideEncryptionCustomerMethod =
ServerSideEncryptionCustomerMethod.AES256,
        ServerSideEncryptionCustomerProvidedKey = base64Key
    };

    InitiateMultipartUploadResponse initResponse =
        await s3Client.InitiateMultipartUploadAsync(initiateRequest);

    // 2. Upload Parts.
    long contentLength = new FileInfo(filePath).Length;
    long partSize = 5 * (long)Math.Pow(2, 20); // 5 MB

    try
    {
        long filePosition = 0;
        for (int i = 1; filePosition < contentLength; i++)
        {
            UploadPartRequest uploadRequest = new UploadPartRequest
            {
                BucketName = existingBucketName,
                Key = sourceKeyName,
                UploadId = initResponse.UploadId,
                PartNumber = i,
                PartSize = partSize,
                FilePosition = filePosition,
                FilePath = filePath,
                ServerSideEncryptionCustomerMethod =
ServerSideEncryptionCustomerMethod.AES256,
                ServerSideEncryptionCustomerProvidedKey = base64Key
            };

            // Upload part and add response to our list.
            uploadResponses.Add(await
s3Client.UploadPartAsync(uploadRequest));

            filePosition += partSize;
        }
    }
}
```

```
// Step 3: complete.
CompleteMultipartUploadRequest completeRequest = new
CompleteMultipartUploadRequest
{
    BucketName = existingBucketName,
    Key = sourceKeyName,
    UploadId = initResponse.UploadId,
    //PartETags = new List<PartETag>(uploadResponses)

};

completeRequest.AddPartETags(uploadResponses);

CompleteMultipartUploadResponse completeUploadResponse =
    await s3Client.CompleteMultipartUploadAsync(completeRequest);

}

catch (Exception exception)
{
    Console.WriteLine("Exception occurred: {0}", exception.Message);
    AbortMultipartUploadRequest abortMPUREquest = new
AbortMultipartUploadRequest
{
    BucketName = existingBucketName,
    Key = sourceKeyName,
    UploadId = initResponse.UploadId
};

await s3Client.AbortMultipartUploadAsync(abortMPUREquest);
}
}
}
}
```

Protecting data using client-side encryption

Client-side encryption is the act of encrypting your data locally to ensure its security as it passes to the Amazon S3 service. The Amazon S3 service receives your encrypted data; it does not play a role in encrypting or decrypting it.

To enable client-side encryption, you have the following options:

- Use a key stored in AWS Key Management Service (AWS KMS).
- Use a key that you store within your application.

Note

Amazon S3 supports only symmetric encryption KMS keys, and not asymmetric KMS keys. For more information, see [Using symmetric and asymmetric keys](#) in the *AWS Key Management Service Developer Guide*.

AWS Encryption SDK

The [AWS Encryption SDK](#) is a client-side encryption library that is separate from the language-specific SDKs. You can use this encryption library to more easily implement encryption best practices in Amazon S3. Unlike the Amazon S3 encryption clients in the language-specific AWS SDKs, the AWS Encryption SDK is not tied to Amazon S3 and can be used to encrypt or decrypt data to be stored anywhere.

The AWS Encryption SDK and the Amazon S3 encryption clients are not compatible because they produce ciphertexts with different data formats. For more information about the AWS Encryption SDK, see the [AWS Encryption SDK Developer Guide](#).

AWS SDK support for Amazon S3 client-side encryption

The following AWS SDKs support client-side encryption:

- [AWS SDK for .NET](#)
- [AWS SDK for Go](#)
- [AWS SDK for Java](#)
- [AWS SDK for PHP](#)
- [AWS SDK for Ruby](#)
- [AWS SDK for C++](#)

For information and examples, see [AWS SDK support for client-side encryption](#) in [AWS General Reference](#).

Option 1: Using a KMS key stored in AWS KMS

With this option, you use an AWS KMS key for client-side encryption when uploading or downloading data in Amazon S3.

- **When uploading an object** — Using the KMS key ID, the client first sends a request to AWS KMS for a new symmetric encryption key that it can use to encrypt their object data. AWS KMS returns two versions of a randomly generated data key:
 - A plaintext version of the data key that the client uses to encrypt the object data.
 - A cipher blob of the same data key that the client uploads to Amazon S3 as object metadata.

Note

The client obtains a unique data key for each object that it uploads.

- **When downloading an object** — The client downloads the encrypted object from Amazon S3 along with the cipher blob version of the data key stored as object metadata. The client then sends the cipher blob to AWS KMS to get the plaintext version of the data key so that it can decrypt the object data.

For more information about AWS KMS, see [What is AWS Key Management Service?](#) in the [AWS Key Management Service Developer Guide](#).

Example

The following code example demonstrates how to upload an object to Amazon S3 using AWS KMS with the AWS SDK for Java. The example uses an AWS managed key to encrypt data on the client side before uploading it to Amazon S3. If you already have a KMS key, you can use that by specifying the value of the `keyId` variable in the example code. If you don't have a KMS key, or you need another one, you can generate one through the Java API. The example code automatically generates a KMS key to use.

For instructions on creating and testing a working example, see [Testing the Amazon S3 Java Code Examples \(p. 1192\)](#).

```
AWSKMS kmsClient = AWSKMSClientBuilder.standard()
    .withRegion(Regions.DEFAULT_REGION)
    .build();

// create KMS key for for testing this example
CreateKeyRequest createKeyRequest = new CreateKeyRequest();
CreateKeyResult createKeyResult = kmsClient.createKey(createKeyRequest);

// --
// specify an AWS KMS key ID
String keyId = createKeyResult.getKeyMetadata().getKeyId();

String s3ObjectKey = "EncryptedContent1.txt";
String s3ObjectContent = "This is the 1st content to encrypt";
```

```
// --  
  
AmazonS3EncryptionV2 s3Encryption = AmazonS3EncryptionClientV2Builder.standard()  
    .withRegion(Regions.US_WEST_2)  
    .withCryptoConfiguration(new  
CryptoConfigurationV2().withCryptoMode(CryptoMode.StrictAuthenticatedEncryption))  
    .withEncryptionMaterialsProvider(new KMSEncryptionMaterialsProvider(keyId))  
    .build();  
  
s3Encryption.putObject(bucket_name, s3ObjectKey, s3ObjectContent);  
System.out.println(s3Encryption.getObjectAsString(bucket_name, s3ObjectKey));  
  
// schedule deletion of KMS key generated for testing  
ScheduleKeyDeletionRequest scheduleKeyDeletionRequest =  
    new  
ScheduleKeyDeletionRequest().withKeyId(keyId).withPendingWindowInDays(7);  
kmsClient.scheduleKeyDeletion(scheduleKeyDeletionRequest);  
  
s3Encryption.shutdown();  
kmsClient.shutdown();
```

Option 2: Using a key stored within your application

With this option, you use a root key that is stored within your application for client-side data encryption.

Important

Your client-side keys and your unencrypted data are never sent to AWS. It's important that you safely manage your encryption keys. If you lose them, you can't decrypt your data.

This is how it works:

- **When uploading an object** — You provide a client-side root key to the Amazon S3 encryption client. The client uses the root key only to encrypt the data encryption key that it generates randomly.

The following steps describe the process:

1. The Amazon S3 encryption client generates a one-time-use symmetric encryption key (also known as a *data encryption key* or *data key*) locally. It uses the data key to encrypt the data of a single Amazon S3 object. The client generates a separate data key for each object.
 2. The client encrypts the data encryption key using the root key that you provide. The client uploads the encrypted data key and its material description as part of the object metadata. The client uses the material description to determine which client-side root key to use for decryption.
 3. The client uploads the encrypted data to Amazon S3 and saves the encrypted data key as object metadata (`x-amz-meta-x-amz-key`) in Amazon S3.
- **When downloading an object** — The client downloads the encrypted object from Amazon S3. Using the material description from the object's metadata, the client determines which root key to use to decrypt the data key. The client uses that root key to decrypt the data key and then uses the data key to decrypt the object.

The client-side root key that you provide can be either a symmetric encryption key or a public/private key pair. The following code examples show how to use each type of key.

For more information, see [Client-Side Data Encryption with the AWS SDK for Java and Amazon S3](#) and [AWS SDK support for client-side encryption](#).

Note

If you get a cipher-encryption error message when you use the encryption API for the first time, your version of the JDK might have a Java Cryptography Extension (JCE) jurisdiction policy file that limits the maximum key length for encryption and decryption transformations to 128 bits. The AWS SDK requires a maximum key length of 256 bits.

To check your maximum key length, use the `getMaxAllowedKeyLength()` method of the `javax.crypto.Cipher` class. To remove the key-length restriction, install the [Java Cryptography Extension \(JCE\) Unlimited Strength Jurisdiction Policy Files](#).

Example

The following code example shows how to do these tasks:

- Generate a 256-bit AES key.
- Use the AES key to encrypt data on the client side before sending it to Amazon S3.
- Use the AES key to decrypt data received from Amazon S3.
- Print out a string representation of the decrypted object.

For instructions on creating and testing a working example, see [Testing the Amazon S3 Java Code Examples \(p. 1192\)](#).

```
KeyGenerator keyGenerator = KeyGenerator.getInstance("AES");
keyGenerator.init(256);

// --
// generate a symmetric encryption key for testing
SecretKey secretKey = keyGenerator.generateKey();

String s3ObjectKey = "EncryptedContent2.txt";
String s3ObjectContent = "This is the 2nd content to encrypt";
// --

AmazonS3EncryptionV2 s3Encryption = AmazonS3EncryptionClientV2Builder.standard()
    .withRegion(Regions.DEFAULT_REGION)
    .withClientConfiguration(new ClientConfiguration())
    .withCryptoConfiguration(new
CryptoConfigurationV2().withCryptoMode(CryptoMode.AuthenticatedEncryption)
    .withEncryptionMaterialsProvider(new StaticEncryptionMaterialsProvider(new
EncryptionMaterials(secretKey)))
    .build());

s3Encryption.putObject(bucket_name, s3ObjectKey, s3ObjectContent);
System.out.println(s3Encryption.getObjectAsString(bucket_name, s3ObjectKey));
s3Encryption.shutdown();
```

Example

The following code example shows how to do these tasks:

- Generate a 2048-bit RSA key pair for testing purposes.
- Use the RSA keys to encrypt data on the client side before sending it to Amazon S3.
- Use the RSA keys to decrypt data received from Amazon S3.
- Print out a string representation of the decrypted object.

For instructions on creating and testing a working example, see [Testing the Amazon S3 Java Code Examples \(p. 1192\)](#).

```
KeyPairGenerator keyPairGenerator = KeyPairGenerator.getInstance("RSA");
keyPairGenerator.initialize(2048);

// --
// generate an asymmetric key pair for testing
```

```
KeyPair keyPair = keyPairGenerator.generateKeyPair();

String s3ObjectKey = "EncryptedContent3.txt";
String s3ObjectContent = "This is the 3rd content to encrypt";
// --

AmazonS3EncryptionV2 s3Encryption = AmazonS3EncryptionClientV2Builder.standard()
    .withRegion(Regions.US_WEST_2)
    .withCryptoConfiguration(new
CryptoConfigurationV2().withCryptoMode(CryptoMode.StrictAuthenticatedEncryption))
    .withEncryptionMaterialsProvider(new StaticEncryptionMaterialsProvider(new
EncryptionMaterials(keyPair)))
    .build();

s3Encryption.putObject(bucket_name, s3ObjectKey, s3ObjectContent);
System.out.println(s3Encryption.getObjectAsString(bucket_name, s3ObjectKey));
s3Encryption.shutdown();
```

Internetwork traffic privacy

This topic describes how Amazon S3 secures connections from the service to other locations.

Traffic between service and on-premises clients and applications

The following connections can be combined with AWS PrivateLink to provide connectivity between your private network and AWS:

- An AWS Site-to-Site VPN connection. For more information, see [What is AWS Site-to-Site VPN?](#)
- An AWS Direct Connect connection. For more information, see [What is AWS Direct Connect?](#)

Access to Amazon S3 via the network is through AWS published APIs. Clients must support Transport Layer Security (TLS) 1.0. We recommend TLS 1.2 or above. Clients must also support cipher suites with Perfect Forward Secrecy (PFS), such as Ephemeral Diffie-Hellman (DHE) or Elliptic Curve Diffie-Hellman Ephemeral (ECDHE). Most modern systems such as Java 7 and later support these modes. Additionally, you must sign requests using an access key ID and a secret access key that are associated with an IAM principal, or you can use the [AWS Security Token Service \(STS\)](#) to generate temporary security credentials to sign requests.

Traffic between AWS resources in the same Region

A virtual private cloud (VPC) endpoint for Amazon S3 is a logical entity within a VPC that allows connectivity only to Amazon S3. The VPC routes requests to Amazon S3 and routes responses back to the VPC. For more information, see [VPC Endpoints](#) in the *VPC User Guide*. For example bucket policies that you can use to control S3 bucket access from VPC endpoints, see [Controlling access from VPC endpoints with bucket policies \(p. 489\)](#).

AWS PrivateLink for Amazon S3

With AWS PrivateLink for Amazon S3, you can provision *interface VPC endpoints* (interface endpoints) in your virtual private cloud (VPC). These endpoints are directly accessible from applications that are on premises over VPN and AWS Direct Connect, or in a different AWS Region over VPC peering.

Interface endpoints are represented by one or more elastic network interfaces (ENIs) that are assigned private IP addresses from subnets in your VPC. Requests that are made to interface endpoints for Amazon S3 are automatically routed to Amazon S3 on the Amazon network. You can also access interface endpoints in your VPC from on-premises applications through AWS Direct Connect or AWS Virtual Private Network (AWS VPN). For more information about how to connect your VPC with your on-premises network, see the [AWS Direct Connect User Guide](#) and the [AWS Site-to-Site VPN User Guide](#).

For general information about interface endpoints, see [Interface VPC endpoints \(AWS PrivateLink\)](#) in the [AWS PrivateLink Guide](#).

Topics

- [Types of VPC endpoints for Amazon S3 \(p. 386\)](#)
- [Restrictions and limitations of AWS PrivateLink for Amazon S3 \(p. 387\)](#)
- [Creating a VPC endpoint \(p. 387\)](#)
- [Accessing Amazon S3 interface endpoints \(p. 387\)](#)
- [Accessing buckets and S3 access points from S3 interface endpoints \(p. 387\)](#)
- [Updating an on-premises DNS configuration \(p. 391\)](#)
- [Creating a VPC endpoint policy for Amazon S3 \(p. 392\)](#)

Types of VPC endpoints for Amazon S3

You can use two types of VPC endpoints to access Amazon S3: *gateway endpoints* and *interface endpoints* (using AWS PrivateLink). A *gateway endpoint* is a gateway that you specify in your route table to access Amazon S3 from your VPC over the AWS network. *Interface endpoints* extend the functionality of gateway endpoints by using private IP addresses to route requests to Amazon S3 from within your VPC, on premises, or from a VPC in another AWS Region using VPC peering or AWS Transit Gateway. For more information, see [What is VPC peering](#) and [Transit Gateway vs VPC peering](#).

Interface endpoints are compatible with gateway endpoints. If you have an existing gateway endpoint in the VPC, you can use both types of endpoints in the same VPC.

Gateway endpoints for Amazon S3	Interface endpoints for Amazon S3
In both cases, your network traffic remains on the AWS network.	
Use Amazon S3 public IP addresses	Use private IP addresses from your VPC to access Amazon S3
Use the same Amazon S3 DNS names	Require endpoint-specific Amazon S3 DNS names
Does not allow access from on premises	Allow access from on premises
Does not allow access from another AWS Region	Allow access from a VPC in another AWS Region using VPC peering or AWS Transit Gateway
Not billed	Billed

For more information about gateway endpoints, see [Gateway VPC endpoints](#) in the [AWS PrivateLink Guide](#).

Restrictions and limitations of AWS PrivateLink for Amazon S3

VPC limitations apply to AWS PrivateLink for Amazon S3. For more information, see [Interface endpoint properties and limitations](#) and [AWS PrivateLink quotas](#) in the *AWS PrivateLink Guide*. In addition, the following restrictions apply.

AWS PrivateLink for Amazon S3 does not support the following:

- [Federal Information Processing Standard \(FIPS\) endpoints](#)
- [Website endpoints \(p. 1116\)](#)
- [Legacy global endpoints \(p. 1180\)](#)
- Using [CopyObject API](#) or [UploadPartCopy API](#) between buckets in different AWS Regions
- Transport Layer Security (TLS) 1.0

Creating a VPC endpoint

To create a VPC interface endpoint, see [Create a VPC endpoint](#) in the *AWS PrivateLink Guide*.

Accessing Amazon S3 interface endpoints

Important

To access Amazon S3 using AWS PrivateLink, you *must* update your applications to use endpoint-specific DNS names.

When you create an interface endpoint, Amazon S3 generates two types of endpoint-specific, S3 DNS names: *Regional* and *zonal*.

- *Regional* DNS names include a unique VPC endpoint ID, a service identifier, the AWS Region, and `vpce.amazonaws.com` in its name. For example, for VPC endpoint ID `vpce-1a2b3c4d`, the DNS name generated might be similar to `vpce-1a2b3c4d-5e6f.s3.us-east-1.vpce.amazonaws.com`.
- *Zonal* DNS names include the Availability Zone—for example, `vpce-1a2b3c4d-5e6f-us-east-1a.s3.us-east-1.vpce.amazonaws.com`. You might use this option if your architecture isolates Availability Zones. For example, you could use it for fault containment or to reduce Regional data transfer costs.

Endpoint-specific S3 DNS names can be resolved from the S3 public DNS domain.

Note

Amazon S3 interface endpoints do *not* support the private DNS feature of interface endpoints. For more information about [Private DNS for interface endpoints](#), see the *AWS PrivateLink Guide*.

Accessing buckets and S3 access points from S3 interface endpoints

You can use the AWS CLI or AWS SDK to access buckets, S3 access points, and S3-control APIs through S3 interface endpoints.

The following image shows the VPC console **Details** tab, where you can find the DNS name of a VPC endpoint. In this example, the *VPC endpoint ID* (*vpce-id*) is `vpce-0e25b8cdd720f900e` and the

DNS name is *.vpce-0e25b8cdd720f900e-argc85vg.s3.us-east-1.vpce.amazonaws.com.
Remember to replace * when using the DNS name. For example, to access a bucket, the DNS name would be bucket.vpce-0e25b8cdd720f900e-argc85vg.s3.us-east-1.vpce.amazonaws.com.

Details	Subnets	Security Groups	Policy	Notifications	Tags
Endpoint ID	vpce-0e25b8cdd720f900e				VPC ID
Status	available				vpc-0e0ccb9d87b1734bd VPCStack VPC
Creation time	January 8, 2021 at 1:30:11 AM UTC-8				Status message
Endpoint type	Interface				Service name com.amazonaws.us-east-1.s3
					DNS names *.vpce-0e25b8cdd720f900e-argc85vg.s3.us-east-1.vpce.amazonaws.com (Z7HUB22UULQXV)

For more about how to view your endpoint-specific DNS names, see [Viewing endpoint service private DNS name configuration](#) in the *VPC User Guide*.

AWS CLI examples

Use the --region and --endpoint-url parameters to access S3 buckets, S3 access points, or S3 control APIs through S3 interface endpoints.

Example: Use the endpoint URL to list objects in your bucket

In the following example, replace the region `us-east-1`, VPC endpoint ID `vpce-1a2b3c4d-5e6f.s3.us-east-1.vpce.amazonaws.com`, and bucket name `my-bucket` with appropriate information.

```
aws s3 --region us-east-1 --endpoint-url https://bucket.vpce-1a2b3c4d-5e6f.s3.us-east-1.vpce.amazonaws.com ls s3://my-bucket/
```

Example: Use the endpoint URL to list objects from an access point

In the following example, replace the ARN `us-east-1:123456789012:accesspoint/test`, region `us-east-1`, and VPC endpoint ID `vpce-1a2b3c4d-5e6f.s3.us-east-1.vpce.amazonaws.com` with appropriate information.

```
aws s3api list-objects-v2 --bucket arn:aws:s3:us-east-1:123456789012:accesspoint/test --region us-east-1 --endpoint-url https://accesspoint.vpce-1a2b3c4d-5e6f.s3.us-east-1.vpce.amazonaws.com
```

Example: Use the endpoint URL to list jobs with S3 control

In the following example, replace the region `us-east-1`, VPC endpoint ID `vpce-1a2b3c4d-5e6f.s3.us-east-1.vpce.amazonaws.com`, and account ID `12345678` with appropriate information.

```
aws s3control --region us-east-1 --endpoint-url https://control.vpce-1a2b3c4d-5e6f.s3.us-east-1.vpce.amazonaws.com list-jobs --account-id 12345678
```

AWS SDK examples

Update your SDKs to the latest version, and configure your clients to use an endpoint URL for accessing a bucket, access point, or S3 control API through S3 interface endpoints.

SDK for Python (Boto3)

Example: Use an endpoint URL to access an S3 bucket

In the following example, replace the region `us-east-1` and VPC endpoint ID `vpce-1a2b3c4d-5e6f.s3.us-east-1.vpce.amazonaws.com` with appropriate information.

```
s3_client = session.client(  
    service_name='s3',  
    region_name='us-east-1',  
    endpoint_url='https://bucket.vpce-1a2b3c4d-5e6f.s3.us-east-1.vpce.amazonaws.com'  
)
```

Example: Use an endpoint URL to access an S3 access point

In the following example, replace the region `us-east-1` and VPC endpoint ID `vpce-1a2b3c4d-5e6f.s3.us-east-1.vpce.amazonaws.com` with appropriate information.

```
ap_client = session.client(  
    service_name='s3',  
    region_name='us-east-1',  
    endpoint_url='https://accesspoint.vpce-1a2b3c4d-5e6f.s3.us-east-1.vpce.amazonaws.com'  
)
```

Example: Use an endpoint URL to access the S3 control API

In the following example, replace the region `us-east-1` and VPC endpoint ID `vpce-1a2b3c4d-5e6f.s3.us-east-1.vpce.amazonaws.com` with appropriate information.

```
control_client = session.client(  
    service_name='s3control',  
    region_name='us-east-1',  
    endpoint_url='https://control.vpce-1a2b3c4d-5e6f.s3.us-east-1.vpce.amazonaws.com'  
)
```

SDK for Java 1.x

Example: Use an endpoint URL to access an S3 bucket

In the following example, replace the VPC endpoint ID `vpce-1a2b3c4d-5e6f.s3.us-east-1.vpce.amazonaws.com` with appropriate information.

```
// bucket client  
final AmazonS3 s3 = AmazonS3ClientBuilder.standard().withEndpointConfiguration(  
    new AwsClientBuilder.EndpointConfiguration(  
        "https://bucket.vpce-1a2b3c4d-5e6f.s3.us-east-1.vpce.amazonaws.com",  
        Regions.DEFAULT_REGION.getName()  
    )  
).build();  
List<Bucket> buckets = s3.listBuckets();
```

Example: Use an endpoint URL to access an S3 access point

In the following example, replace the VPC endpoint ID `vpce-1a2b3c4d-5e6f.s3.us-east-1.vpce.amazonaws.com` and ARN `us-east-1:123456789012:accesspoint/prod` with appropriate information.

```
// accesspoint client  
final AmazonS3 s3accesspoint =  
    AmazonS3ClientBuilder.standard().withEndpointConfiguration(  
        new AwsClientBuilder.EndpointConfiguration(  
            "https://accesspoint.vpce-1a2b3c4d-5e6f.s3.us-east-1.vpce.amazonaws.com",  
            Regions.DEFAULT_REGION.getName()
```

```
)  
.build();  
ObjectListing objects = s3accesspoint.listObjects("arn:aws:s3:us-  
east-1:123456789012:accesspoint/prod");
```

Example: Use an endpoint URL to access the S3 control API

In the following example, replace the VPC endpoint ID `vpce-1a2b3c4d-5e6f.s3.us-east-1.vpce.amazonaws.com` with appropriate information.

```
// control client  
final AWSS3Control s3control = AWSS3ControlClient.builder().withEndpointConfiguration(  
    new AwsClientBuilder.EndpointConfiguration(  
        "https://control.vpce-1a2b3c4d-5e6f.s3.us-east-1.vpce.amazonaws.com",  
        Regions.DEFAULT_REGION.getName()  
    )  
>.  
).build();  
final ListJobsResult jobs = s3control.listJobs(new ListJobsRequest());
```

SDK for Java 2.x

Example: Use an endpoint URL to access an S3 bucket

In the following example, replace the VPC endpoint ID `vpce-1a2b3c4d-5e6f.s3.us-east-1.vpce.amazonaws.com` and the Region `Region.US_EAST_1` with appropriate information.

```
// bucket client  
Region region = Region.US_EAST_1;  
s3Client = S3Client.builder().region(region)  
  
.endpointOverride(URI.create("https://bucket.vpce-1a2b3c4d-5e6f.s3.us-  
east-1.vpce.amazonaws.com"))  
.build()
```

Example: Use an endpoint URL to access an S3 access point

In the following example, replace the VPC endpoint ID `vpce-1a2b3c4d-5e6f.s3.us-east-1.vpce.amazonaws.com` and the Region `Region.US_EAST_1` with appropriate information.

```
// accesspoint client  
Region region = Region.US_EAST_1;  
s3Client = S3Client.builder().region(region)  
  
.endpointOverride(URI.create("https://accesspoint.vpce-1a2b3c4d-5e6f.s3.us-  
east-1.vpce.amazonaws.com"))  
.build()
```

Example: Use an endpoint URL to access the S3 control API

In the following example, replace the VPC endpoint ID `vpce-1a2b3c4d-5e6f.s3.us-east-1.vpce.amazonaws.com` and the Region `Region.US_EAST_1` with appropriate information.

```
// control client  
Region region = Region.US_EAST_1;  
s3ControlClient = S3ControlClient.builder().region(region)
```

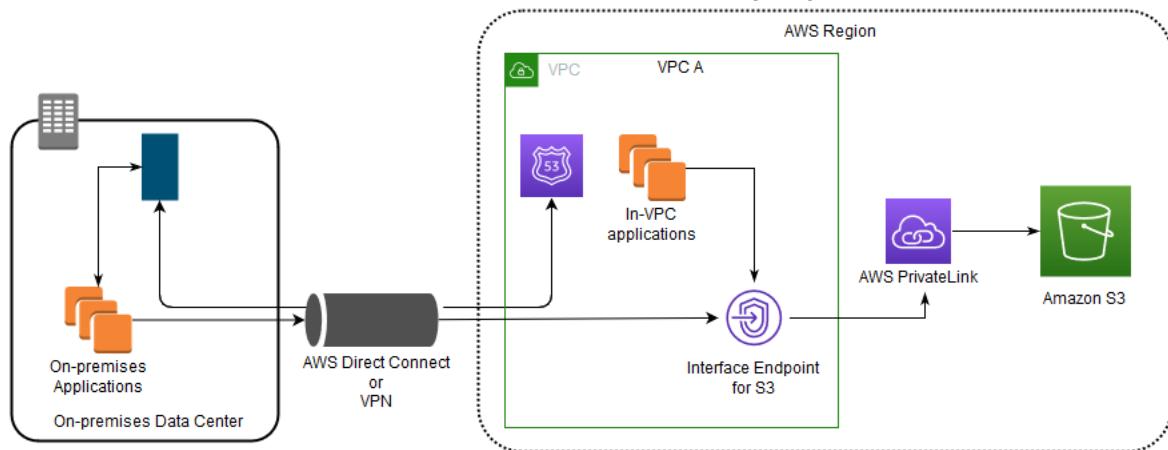
```
.endpointOverride(URI.create("https://control.vpce-1a2b3c4d-5e6f.s3.us-east-1.vpce.amazonaws.com"))
.build()
```

Updating an on-premises DNS configuration

When using endpoint-specific DNS names to access the interface endpoints for Amazon S3, you don't have to update your on-premises DNS resolver. You can resolve the endpoint-specific DNS name with the private IP address of the interface endpoint from the public Amazon S3 DNS domain.

Using interface endpoints to access Amazon S3 without a gateway endpoint or an internet gateway in the VPC

Interface endpoints in your VPC can route both in-VPC applications and on-premises applications to Amazon S3 over the Amazon network, as illustrated in the following diagram.

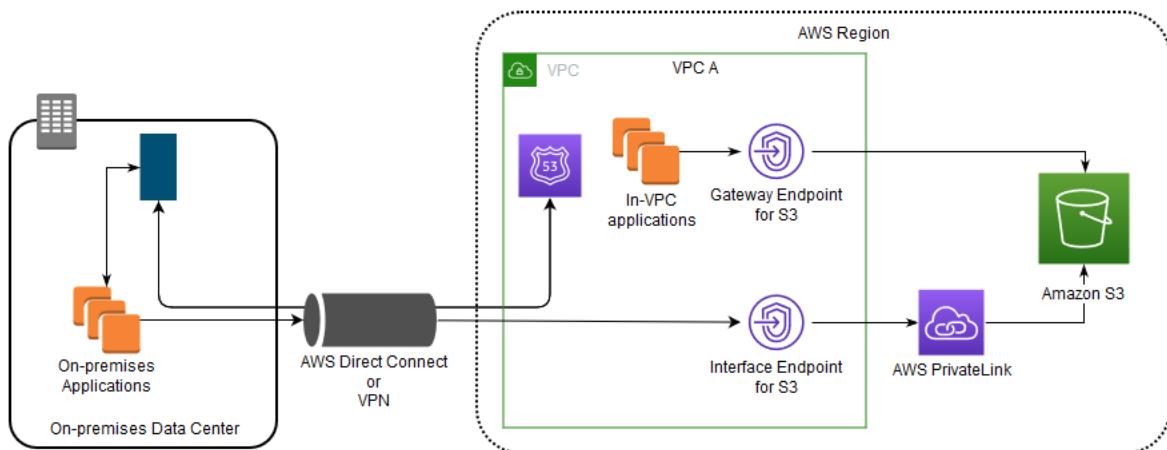


The diagram illustrates the following:

- Your on-premises network uses AWS Direct Connect or AWS VPN to connect to VPC A.
- Your applications on-premises and in VPC A use endpoint-specific DNS names to access Amazon S3 through the S3 interface endpoint.
- On-premises applications send data to the interface endpoint in the VPC through AWS Direct Connect (or AWS VPN). AWS PrivateLink moves the data from the interface endpoint to Amazon S3 over the AWS network.
- In-VPC applications also send traffic to the interface endpoint. AWS PrivateLink moves the data from the interface endpoint to Amazon S3 over the AWS network.

Using gateway endpoints and interface endpoints together in the same VPC to access Amazon S3

You can create interface endpoints and retain the existing gateway endpoint in the same VPC, as the following diagram shows. By doing this, you allow in-VPC applications to continue accessing Amazon S3 through the gateway endpoint, which is not billed. Then, only your on-premises applications would use interface endpoints to access Amazon S3. To access S3 this way, you must update your on-premises applications to use endpoint-specific DNS names for Amazon S3.



The diagram illustrates the following:

- On-premises applications use endpoint-specific DNS names to send data to the interface endpoint within the VPC through AWS Direct Connect (or AWS VPN). AWS PrivateLink moves the data from the interface endpoint to Amazon S3 over the AWS network.
- Using default Regional Amazon S3 names, in-VPC applications send data to the gateway endpoint that connects to Amazon S3 over the AWS network.

For more information about gateway endpoints, see [Gateway VPC endpoints](#) in the *VPC User Guide*.

Creating a VPC endpoint policy for Amazon S3

You can attach an endpoint policy to your VPC endpoint that controls access to Amazon S3. The policy specifies the following information:

- The AWS Identity and Access Management (IAM) principal that can perform actions
- The actions that can be performed
- The resources on which actions can be performed

You can also use Amazon S3 bucket policies to restrict access to specific buckets from a specific VPC endpoint using the `aws:sourceVpc` condition in your bucket policy. The following examples show policies that restrict access to a bucket or to an endpoint.

Topics

- [Example: Restricting access to a specific bucket from a VPC endpoint \(p. 393\)](#)
- [Example: Restricting access to buckets in a specific account from a VPC endpoint \(p. 393\)](#)
- [Example: Restricting access to a specific VPC endpoint in the S3 bucket policy \(p. 394\)](#)

Important

- When applying the Amazon S3 bucket policies for VPC endpoints described in this section, you might block your access to the bucket without intending to do so. Bucket permissions that are intended to specifically limit bucket access to connections originating from your VPC endpoint can block all connections to the bucket. For information about how to fix this issue, see [My bucket policy has the wrong VPC or VPC endpoint ID. How can I fix the policy so that I can access the bucket?](#) in the *AWS Support Knowledge Center*.
- Before using the following example policy, replace the VPC endpoint ID with an appropriate value for your use case. Otherwise, you won't be able to access your bucket.

- This policy disables *console* access to the specified bucket, because console requests don't originate from the specified VPC endpoint.

Example: Restricting access to a specific bucket from a VPC endpoint

You can create an endpoint policy that restricts access to specific Amazon S3 buckets only. This is useful if you have other AWS services in your VPC that use buckets. The following bucket policy restricts access to *DOC-EXAMPLE-BUCKET1* only. Replace *DOC-EXAMPLE-BUCKET1* with the name of your bucket.

```
{  
    "Version": "2012-10-17",  
    "Id": "Policy1415115909151",  
    "Statement": [  
        { "Sid": "Access-to-specific-bucket-only",  
          "Principal": "*",  
          "Action": [  
              "s3:GetObject",  
              "s3:PutObject"  
            ],  
          "Effect": "Allow",  
          "Resource": ["arn:aws:s3:::DOC-EXAMPLE-BUCKET1",  
                      "arn:aws:s3:::DOC-EXAMPLE-BUCKET1/*"]  
        }  
    ]  
}
```

Example: Restricting access to buckets in a specific account from a VPC endpoint

You can create a policy that restricts access only to the S3 buckets in a specific AWS account. Use this to prevent clients within your VPC from accessing buckets that you do not own. The following example creates a policy that restricts access to resources owned by a single AWS account ID, *111122223333*.

```
{  
    "Statement": [  
        {  
            "Sid": "Access-to-bucket-in-specific-account-only",  
            "Principal": "*",  
            "Action": [  
                "s3:GetObject",  
                "s3:PutObject"  
            ],  
            "Effect": "Deny",  
            "Resource": "arn:aws:s3:::*",  
            "Condition": {  
                "StringNotEquals": {  
                    "aws:ResourceAccount": "111122223333"  
                }  
            }  
        }  
    ]  
}
```

Note

You can use either the `aws:ResourceAccount` or `s3:ResourceAccount` key in your IAM policy to specify the AWS account ID of the resource being accessed. However, be aware that some AWS services rely on access to AWS managed buckets. Therefore, using the

`aws:ResourceAccount` or `s3:ResourceAccount` key in your IAM policy might also impact access to these resources.

Example: Restricting access to a specific VPC endpoint in the S3 bucket policy

Example: Restricting access to a specific VPC endpoint in the S3 bucket policy

The following Amazon S3 bucket policy allows access to a specific bucket, `DOC-EXAMPLE-BUCKET2`, from endpoint `vpce-1a2b3c4d` only. The policy denies all access to the bucket if the specified endpoint is not being used. The `aws:sourceVpce` condition is used to specify the endpoint and does not require an Amazon Resource Name (ARN) for the VPC endpoint resource, only the endpoint ID. Replace `DOC-EXAMPLE-BUCKET2` and `vpce-1a2b3c4d` with a real bucket name and endpoint.

```
{  
    "Version": "2012-10-17",  
    "Id": "Policy1415115909152",  
    "Statement": [  
        { "Sid": "Access-to-specific-VPCE-only",  
          "Principal": "*",  
          "Action": "s3:*",  
          "Effect": "Deny",  
          "Resource": ["arn:aws:s3:::DOC-EXAMPLE-BUCKET2",  
                      "arn:aws:s3:::DOC-EXAMPLE-BUCKET2/*"],  
          "Condition": {"StringNotEquals": {"aws:sourceVpce": "vpce-1a2b3c4d"}}  
        }  
    ]  
}
```

For more policy examples, see [Endpoints for Amazon S3](#) in the *VPC User Guide*.

For more information about VPC connectivity, see [Network-to-VPC connectivity options](#) in the AWS whitepaper [Amazon Virtual Private Cloud Connectivity Options](#).

Identity and access management in Amazon S3

By default, all Amazon S3 resources—buckets, objects, and related subresources (for example, lifecycle configuration and website configuration)—are private. Only the resource owner, the AWS account that created it, can access the resource. The resource owner can optionally grant access permissions to others by writing an access policy.

Amazon S3 offers access policy options broadly categorized as resource-based policies and user policies. Access policies that you attach to your resources (buckets and objects) are referred to as *resource-based policies*. For example, bucket policies and access point policies are resource-based policies. You can also attach access policies to users in your account. These are called *user policies*. You can choose to use resource-based policies, user policies, or some combination of these to manage permissions to your Amazon S3 resources. You can also use access control lists (ACLs) to grant basic read and write permissions to other AWS accounts.

By default, when another AWS account uploads an object to your S3 bucket, that account (the object writer) owns the object, has access to it, and can grant other users access to it through ACLs. You can use Object Ownership to change this default behavior so that ACLs are disabled and you, as the bucket owner, automatically own every object in your bucket. As a result, access control for your data is based on policies, such as IAM policies, S3 bucket policies, virtual private cloud (VPC) endpoint policies, and AWS Organizations service control policies (SCPs).

A majority of modern use cases in Amazon S3 no longer require the use of ACLs, and we recommend that you disable ACLs except in unusual circumstances where you need to control access for each

object individually. With Object Ownership, you can disable ACLs and rely on policies for access control. When you disable ACLs, you can easily maintain a bucket with objects uploaded by different AWS accounts. You, as the bucket owner, own all the objects in the bucket and can manage access to them using policies. For more information, see [Controlling ownership of objects and disabling ACLs for your bucket \(p. 601\)](#).

For more information about managing access to your Amazon S3 objects and buckets, see the topics below.

Topics

- [Overview of managing access \(p. 395\)](#)
- [Access policy guidelines \(p. 400\)](#)
- [How Amazon S3 authorizes a request \(p. 404\)](#)
- [Bucket policies and user policies \(p. 411\)](#)
- [AWS managed policies for Amazon S3 \(p. 552\)](#)
- [Managing access with ACLs \(p. 554\)](#)
- [Using cross-origin resource sharing \(CORS\) \(p. 573\)](#)
- [Blocking public access to your Amazon S3 storage \(p. 584\)](#)
- [Reviewing bucket access using Access Analyzer for S3 \(p. 593\)](#)
- [Verifying bucket ownership with bucket owner condition \(p. 598\)](#)

Overview of managing access

When granting permissions in Amazon S3, you decide who is getting the permissions, which Amazon S3 resources they are getting permissions for, and the specific actions you want to allow on those resources. The following sections provide an overview of Amazon S3 resources and how to determine the best method to control access to them.

Topics

- [Amazon S3 resources: buckets and objects \(p. 395\)](#)
- [Amazon S3 bucket and object ownership \(p. 396\)](#)
- [Resource operations \(p. 397\)](#)
- [Managing access to resources \(p. 397\)](#)
- [Which access control method should I use? \(p. 400\)](#)

Amazon S3 resources: buckets and objects

In AWS, a resource is an entity that you can work with. In Amazon S3, *buckets* and *objects* are the resources, and both have associated subresources.

Bucket subresources include the following:

- `lifecycle` – Stores lifecycle configuration information. For more information, see [Managing your storage lifecycle \(p. 701\)](#).
- `website` – Stores website configuration information if you configure your bucket for website hosting. For information, see [Hosting a static website using Amazon S3 \(p. 1116\)](#).
- `versioning` – Stores versioning configuration. For more information, see [PUT Bucket versioning in the Amazon Simple Storage Service API Reference](#).
- `policy and acl` (access control list) – Store access permission information for the bucket.
- `cors` (cross-origin resource sharing) – Supports configuring your bucket to allow cross-origin requests. For more information, see [Using cross-origin resource sharing \(CORS\) \(p. 573\)](#).

- **object ownership** – Enables the bucket owner to take ownership of new objects in the bucket, regardless of who uploads them. For more information, see [Controlling ownership of objects and disabling ACLs for your bucket \(p. 601\)](#).
- **logging** – Enables you to request Amazon S3 to save bucket access logs.

Object subresources include the following:

- **acl** – Stores a list of access permissions on the object. For more information, see [Access control list \(ACL\) overview \(p. 554\)](#).
- **restore** – Supports temporarily restoring an archived object. For more information, see [POST Object restore](#) in the *Amazon Simple Storage Service API Reference*.

An object in the S3 Glacier Flexible Retrieval storage class is an archived object. To access the object, you must first initiate a restore request, which restores a copy of the archived object. In the request, you specify the number of days that you want the restored copy to exist. For more information about archiving objects, see [Managing your storage lifecycle \(p. 701\)](#).

Amazon S3 bucket and object ownership

Buckets and objects are Amazon S3 resources. By default, only the resource owner can access these resources. The resource owner refers to the AWS account that creates the resource. For example:

- The AWS account that you use to create buckets and upload objects owns those resources.
- If you upload an object using AWS Identity and Access Management (IAM) user or role credentials, the AWS account that the user or role belongs to owns the object.
- A bucket owner can grant cross-account permissions to another AWS account (or users in another account) to upload objects. In this case, the AWS account that uploads objects owns those objects. The bucket owner does not have permissions on the objects that other accounts own, with the following exceptions:
 - The bucket owner pays the bills. The bucket owner can deny access to any objects, or delete any objects in the bucket, regardless of who owns them.
 - The bucket owner can archive any objects or restore archived objects regardless of who owns them. Archival refers to the storage class used to store the objects. For more information, see [Managing your storage lifecycle \(p. 701\)](#).

Ownership and request authentication

All requests to a bucket are either authenticated or unauthenticated. Authenticated requests must include a signature value that authenticates the request sender, and unauthenticated requests do not. For more information about request authentication, see [Making requests \(p. 1138\)](#).

A bucket owner can allow unauthenticated requests. For example, unauthenticated [PUT Object](#) requests are allowed when a bucket has a public bucket policy, or when a bucket ACL grants **WRITE** or **FULL_CONTROL** access to the All Users group or the anonymous user specifically. For more information about public bucket policies and public access control lists (ACLs), see [The meaning of "public" \(p. 586\)](#).

All unauthenticated requests are made by the anonymous user. This user is represented in ACLs by the specific canonical user ID `65a011a29cdf8ec533ec3d1ccaae921c`. If an object is uploaded to a bucket through an unauthenticated request, the anonymous user owns the object. The default object ACL grants **FULL_CONTROL** to the anonymous user as the object's owner. Therefore, Amazon S3 allows unauthenticated requests to retrieve the object or modify its ACL.

To prevent objects from being modified by the anonymous user, we recommend that you do not implement bucket policies that allow anonymous public writes to your bucket or use ACLs that allow

the anonymous user write access to your bucket. You can enforce this recommended behavior by using Amazon S3 Block Public Access.

For more information about blocking public access, see [Blocking public access to your Amazon S3 storage \(p. 584\)](#). For more information about ACLs, see [Access control list \(ACL\) overview \(p. 554\)](#).

Important

We recommend that you don't use the AWS account root user credentials to make authenticated requests. Instead, create an IAM user and grant that user full access. We refer to these users as *administrator users*. You can use the administrator user credentials, instead of AWS account root user credentials, to interact with AWS and perform tasks, such as create a bucket, create users, and grant permissions. For more information, see [AWS account root user credentials and IAM user credentials](#) in the *AWS General Reference* and [Security best practices in IAM](#) in the *IAM User Guide*.

Resource operations

Amazon S3 provides a set of operations to work with the Amazon S3 resources. For a list of available operations, see [Actions defined by Amazon S3 \(p. 431\)](#).

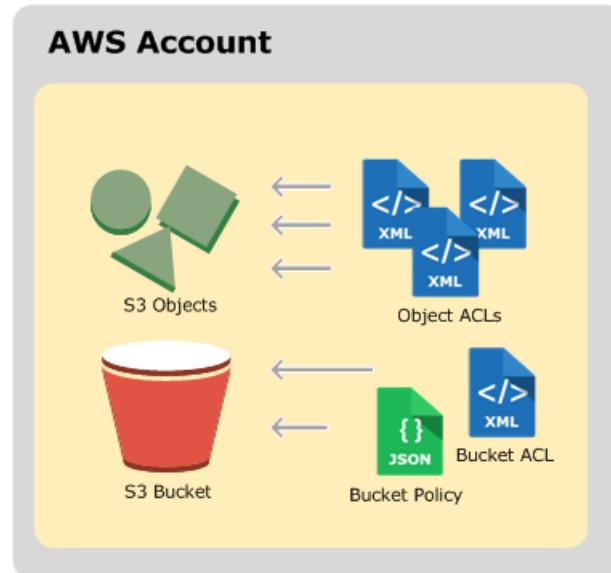
Managing access to resources

Managing access refers to granting others (AWS accounts and users) permission to perform the resource operations by writing an access policy. For example, you can grant `PUT Object` permission to a user in an AWS account so the user can upload objects to your bucket. In addition to granting permissions to individual users and accounts, you can grant permissions to everyone (also referred as anonymous access) or to all authenticated users (users with AWS credentials). For example, if you configure your bucket as a website, you may want to make objects public by granting the `GET Object` permission to everyone.

Access policy options

Access policy describes who has access to what. You can associate an access policy with a resource (bucket and object) or a user. Accordingly, you can categorize the available Amazon S3 access policies as follows:

- **Resource-based policies** – Bucket policies and access control lists (ACLs) are resource-based because you attach them to your Amazon S3 resources.



- **ACL** – Each bucket and object has an ACL associated with it. An ACL is a list of grants identifying grantee and permission granted. You use ACLs to grant basic read/write permissions to other AWS accounts. ACLs use an Amazon S3-specific XML schema.

The following is an example bucket ACL. The grant in the ACL shows a bucket owner as having full control permission.

```
<?xml version="1.0" encoding="UTF-8"?>
<AccessControlPolicy xmlns="http://s3.amazonaws.com/doc/2006-03-01/">
  <Owner>
    <ID>*** Owner-Canonical-User-ID ***</ID>
    <DisplayName>owner-display-name</DisplayName>
  </Owner>
  <AccessControlList>
    <Grant>
      <Grantee xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        xsi:type="Canonical User">
        <ID>*** Owner-Canonical-User-ID ***</ID>
        <DisplayName>display-name</DisplayName>
      </Grantee>
      <Permission>FULL_CONTROL</Permission>
    </Grant>
  </AccessControlList>
</AccessControlPolicy>
```

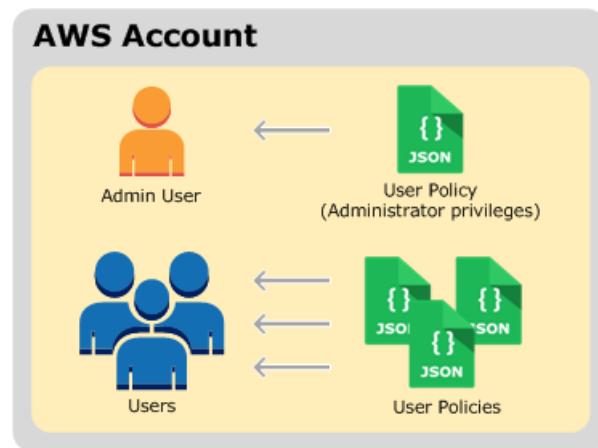
Both bucket and object ACLs use the same XML schema.

- **Bucket policy** – For your bucket, you can add a bucket policy to grant other AWS accounts or IAM users permissions for the bucket and the objects in it. Any object permissions apply only to the objects that the bucket owner creates. Bucket policies supplement, and in many cases, replace ACL-based access policies.

The following is an example bucket policy. You express bucket policy (and user policy) using a JSON file. The policy grants anonymous read permission on all objects in a bucket. The bucket policy has one statement, which allows the `s3:GetObject` action (read permission) on objects in a bucket named `examplebucket`. By specifying the `principal` with a wild card (*), the policy grants anonymous access, and should be used carefully. For example, the following bucket policy would make objects publicly accessible.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "GrantAnonymousReadPermissions",
      "Effect": "Allow",
      "Principal": "*",
      "Action": ["s3:GetObject"],
      "Resource": ["arn:aws:s3:::awsexamplebucket1/*"]
    }
  ]
}
```

- **User policies** – You can use IAM to manage access to your Amazon S3 resources. You can create IAM users, groups, and roles in your account and attach access policies to them granting them access to AWS resources, including Amazon S3.



For more information about IAM, see [AWS Identity and Access Management \(IAM\)](#).

The following is an example of a user policy. You cannot grant anonymous permissions in an IAM user policy, because the policy is attached to a user. The example policy allows the associated user that it's attached to perform six different Amazon S3 actions on a bucket and the objects in it. You can attach this policy to a specific IAM user, group, or role.

```
{  
    "Version": "2012-10-17",  
    "Statement": [  
        {  
            "Sid": "AssignUserActions",  
            "Effect": "Allow",  
            "Action": [  
                "s3:PutObject",  
                "s3:GetObject",  
                "s3>ListBucket",  
                "s3>DeleteObject",  
                "s3:GetBucketLocation"  
            ],  
            "Resource": [  
                "arn:aws:s3:::awsexamplebucket1/*",  
                "arn:aws:s3:::awsexamplebucket1"  
            ]  
        },  
        {  
            "Sid": "ExampleStatement2",  
            "Effect": "Allow",  
            "Action": "s3>ListAllMyBuckets",  
            "Resource": "*"  
        }  
    ]  
}
```

When Amazon S3 receives a request, it must evaluate all the access policies to determine whether to authorize or deny the request. For more information about how Amazon S3 evaluates these policies, see [How Amazon S3 authorizes a request \(p. 404\)](#).

Access Analyzer for S3

On the Amazon S3 console, you can use Access Analyzer for S3 to review all buckets that have bucket access control lists (ACLs), bucket policies, or access point policies that grant public or shared access. Access Analyzer for S3 alerts you to buckets that are configured to allow access to anyone on the

internet or other AWS accounts, including AWS accounts outside of your organization. For each public or shared bucket, you receive findings that report the source and level of public or shared access.

In Access Analyzer for S3, you can block all public access to a bucket with a single click. We recommend that you block all access to your buckets unless you require public access to support a specific use case. Before you block all public access, ensure that your applications will continue to work correctly without public access. For more information, see [Blocking public access to your Amazon S3 storage \(p. 584\)](#).

You can also drill down into bucket-level permission settings to configure granular levels of access. For specific and verified use cases that require public or shared access, you can acknowledge and record your intent for the bucket to remain public or shared by archiving the findings for the bucket. You can revisit and modify these bucket configurations at any time. You can also download your findings as a CSV report for auditing purposes.

Access Analyzer for S3 is available at no extra cost on the Amazon S3 console. Access Analyzer for S3 is powered by AWS Identity and Access Management (IAM) Access Analyzer. To use Access Analyzer for S3 on the Amazon S3 console, you must visit the IAM console and create an account-level analyzer in IAM Access Analyzer on a per-Region basis.

For more information about Access Analyzer for S3, see [Reviewing bucket access using Access Analyzer for S3 \(p. 593\)](#).

Which access control method should I use?

With the options available to write an access policy, the following questions arise:

- When should I use which access control method? For example, to grant bucket permissions, should I use a bucket policy or bucket ACL?

I own a bucket and the objects in the bucket. Should I use a resource-based access policy or an IAM user policy?

If I use a resource-based access policy, should I use a bucket policy or an object ACL to manage object permissions?

- I own a bucket, but I don't own all of the objects in it. How are access permissions managed for the objects that somebody else owns?
- If I grant access by using a combination of these access policy options, how does Amazon S3 determine if a user has permission to perform a requested operation?

The following sections explain these access control alternatives, how Amazon S3 evaluates access control mechanisms, and when to use which access control method. They also provide example walkthroughs.

- [Access policy guidelines \(p. 400\)](#)
- [How Amazon S3 authorizes a request \(p. 404\)](#)
- [Example walkthroughs: Managing access to your Amazon S3 resources \(p. 524\)](#)
- [Access control best practices \(p. 22\)](#)

Access policy guidelines

Amazon S3 supports resource-based policies and user policies to manage access to your Amazon S3 resources. For more information, see [Managing access to resources \(p. 397\)](#). Resource-based policies include bucket policies, bucket access control lists (ACLs), and object ACLs. This section describes specific scenarios for using resource-based access policies to manage access to your Amazon S3 resources.

Topics

- [When to use an ACL-based access policy \(bucket and object ACLs\) \(p. 401\)](#)
- [When to use a bucket policy \(p. 402\)](#)
- [When to use a user policy \(p. 403\)](#)
- [Related topics \(p. 403\)](#)

When to use an ACL-based access policy (bucket and object ACLs)

Both buckets and objects have associated ACLs that you can use to grant permissions.

By default, when another AWS account uploads an object to your S3 bucket, that account (the object writer) owns the object, has access to it, and can grant other users access to it through ACLs. You can use Object Ownership to change this default behavior so that ACLs are disabled and you, as the bucket owner, automatically own every object in your bucket. As a result, access control for your data is based on policies, such as IAM policies, S3 bucket policies, virtual private cloud (VPC) endpoint policies, and AWS Organizations service control policies (SCPs).

A majority of modern use cases in Amazon S3 no longer require the use of ACLs, and we recommend that you disable ACLs except in unusual circumstances where you need to control access for each object individually. With Object Ownership, you can disable ACLs and rely on policies for access control. When you disable ACLs, you can easily maintain a bucket with objects uploaded by different AWS accounts. You, as the bucket owner, own all the objects in the bucket and can manage access to them using policies. For more information, see [Controlling ownership of objects and disabling ACLs for your bucket \(p. 601\)](#).

Important

If your bucket uses the bucket owner enforced setting for S3 Object Ownership, you must use policies to grant access to your bucket and the objects in it. Requests to set ACLs or update ACLs fail and return the `AccessControlListNotSupported` error code. Requests to read ACLs are still supported.

When to use an object ACL

The following are the scenarios when you would use object ACLs to manage permissions.

Objects are not owned by the bucket owner

An object ACL is the only way to manage access to objects that are not owned by the bucket owner. An AWS account that owns the bucket can grant another AWS account permission to upload objects. The bucket owner does not own these objects. The AWS account that created the object must grant permissions using object ACLs.

Note

A bucket owner cannot grant permissions on objects it does not own. For example, a bucket policy granting object permissions applies only to objects owned by the bucket owner. However, the bucket owner, who pays the bills, can write a bucket policy to deny access to any objects in the bucket, regardless of who owns it. The bucket owner can also delete any objects in the bucket.

You need to manage permissions at the object level

Suppose that the permissions vary by object and you need to manage permissions at the object level. You can write a single policy statement granting an AWS account read permission on millions of objects with a specific [key name prefix](#). For example, you could grant read permission on objects starting with the key name prefix `logs`. However, if your access permissions vary by object, granting permissions to individual objects using a bucket policy might not be practical. Also, the bucket policies are limited to 20 KB in size.

In this case, you might find using object ACLs a good alternative. However, even an object ACL is also limited to a maximum of 100 grants. For more information, see [Access control list \(ACL\) overview \(p. 554\)](#).

Object ACLs control only object-level permissions

There is a single bucket policy for the entire bucket, but object ACLs are specified per object.

An AWS account that owns a bucket can grant another AWS account permission to manage an access policy. Doing so allows that account to change anything in the policy. To better manage permissions, you might choose not to give such a broad permission, and instead grant the other account only the `READ-ACP` and `WRITE-ACP` permissions on a subset of objects. This limits the account to manage permissions only on specific objects by updating individual object ACLs.

If you want to use ACLs to manage permissions at the object level and you also want to own new objects written to your bucket, you can apply the bucket owner preferred setting for Object Ownership. A bucket with the bucket owner preferred setting continues to accept and honor bucket and object ACLs. With this setting, new objects that are written with the `bucket-owner-full-control` canned ACL will be automatically owned by the bucket owner rather than the object writer. All other ACL behaviors remain in place. To require all Amazon S3 PUT operations to include the `bucket-owner-full-control` canned ACL, you can [add a bucket policy \(p. 623\)](#) that allows only object uploads using this ACL.

Alternatives to using ACLs

In addition to an object ACL, there are other ways an object owner can manage object permissions:

- If the AWS account that owns the object also owns the bucket, it can write a bucket policy to manage the object permissions.
- If the AWS account that owns the object wants to grant permission to a user in its account, it can use a user policy.
- If you, as the bucket owner, want to automatically own and have full control over every object in your bucket, you can apply the bucket owner enforced setting for Object Ownership to disable ACLs. As a result, access control for your data is based on policies. For more information, see [Controlling ownership of objects and disabling ACLs for your bucket \(p. 601\)](#).

When to use a bucket ACL

The only recommended use case for bucket ACLs is to grant permissions to certain AWS services like the Amazon CloudFront `awslogsdelivery` account. When you create or update a distribution and enable CloudFront logging, CloudFront updates the bucket ACL to give the `awslogsdelivery` account `FULL_CONTROL` permissions to write logs to your bucket. For more information, see [Permissions required to configure standard logging and to access your log files](#) in the *Amazon CloudFront Developer Guide*. If the bucket that stores the logs uses the bucket owner enforced setting for S3 Object Ownership to disable ACLs, CloudFront cannot write logs to the bucket. For more information, see [Controlling ownership of objects and disabling ACLs for your bucket \(p. 601\)](#).

When to use a bucket policy

If an AWS account that owns a bucket wants to grant permission to users in its account, it can use either a bucket policy or a user policy. However, in the following scenarios, you must use a bucket policy.

You want to manage cross-account permissions for all Amazon S3 permissions

You can use ACLs to grant cross-account permissions to other accounts. But ACLs support only a finite set of permissions, and these don't include all Amazon S3 permissions. For more information, see [What permissions can I grant? \(p. 557\)](#) For example, you can't grant permissions on bucket subresources. For more information, see [Identity and access management in Amazon S3 \(p. 394\)](#).

Both bucket and user policies support granting permission for all Amazon S3 operations. (For more information, see [Amazon S3 actions \(p. 415\)](#).) However, the user policies are for managing permissions for users in your account. For cross-account permissions to other AWS accounts or users in another account, you must use a bucket policy.

When to use a user policy

In general, you can use either a user policy or a bucket policy to manage permissions. You can choose to manage permissions by creating users and managing permissions individually by attaching policies to users (or user groups). Or, you might find that resource-based policies, such as a bucket policy, work better for your scenario.

With AWS Identity and Access Management (IAM) you can create multiple users within your AWS account and manage their permissions through user policies. An IAM user must have permissions from the parent account to which it belongs, and from the AWS account that owns the resource that the user wants to access. The permissions can be granted as follows:

- **Permission from the parent account** – The parent account can grant permissions to its user by attaching a user policy.
- **Permission from the resource owner** – The resource owner can grant permission to either the IAM user (using a bucket policy) or the parent account (using a bucket policy, bucket ACL, or object ACL).

This is similar to a child who wants to play with a toy that belongs to someone else. To play with the toy, the child must get permission from a parent and permission from the toy owner.

For more information, see [Bucket policies and user policies \(p. 411\)](#).

Permission delegation

If an AWS account owns a resource, it can grant those permissions to another AWS account. That account can then delegate those permissions, or a subset of them, to users in the account. This is referred to as permission delegation. But an account that receives permissions from another account cannot delegate permission cross-account to another AWS account.

Related topics

We recommend that you first review all introductory topics that explain how you manage access to your Amazon S3 resources and related guidelines. For more information, see [Identity and access management in Amazon S3 \(p. 394\)](#). You can then use the following topics for more information about specific access policy options.

- [Access control list \(ACL\) overview \(p. 554\)](#)
- [Controlling ownership of objects and disabling ACLs for your bucket \(p. 601\)](#)

How Amazon S3 authorizes a request

When Amazon S3 receives a request—for example, a bucket or an object operation—it first verifies that the requester has the necessary permissions. Amazon S3 evaluates all the relevant access policies, user policies, and resource-based policies (bucket policy, bucket ACL, object ACL) in deciding whether to authorize the request.

In order to determine whether the requester has permission to perform the specific operation, Amazon S3 does the following, in order, when it receives a request:

1. Converts all the relevant access policies (user policy, bucket policy, ACLs) at run time into a set of policies for evaluation.
2. Evaluates the resulting set of policies in the following steps. In each step, Amazon S3 evaluates a subset of policies in a specific context, based on the context authority.
 - a. **User context** – In the user context, the parent account to which the user belongs is the context authority.

Amazon S3 evaluates a subset of policies owned by the parent account. This subset includes the user policy that the parent attaches to the user. If the parent also owns the resource in the request (bucket, object), Amazon S3 also evaluates the corresponding resource policies (bucket policy, bucket ACL, and object ACL) at the same time.

A user must have permission from the parent account to perform the operation.

This step applies only if the request is made by a user in an AWS account. If the request is made using root credentials of an AWS account, Amazon S3 skips this step.

- b. **Bucket context** – In the bucket context, Amazon S3 evaluates policies owned by the AWS account that owns the bucket.

If the request is for a bucket operation, the requester must have permission from the bucket owner. If the request is for an object, Amazon S3 evaluates all the policies owned by the bucket owner to check if the bucket owner has not explicitly denied access to the object. If there is an explicit deny set, Amazon S3 does not authorize the request.

- c. **Object context** – If the request is for an object, Amazon S3 evaluates the subset of policies owned by the object owner.

Following are some of the example scenarios that illustrate how Amazon S3 authorizes a request.

Example Requester is an IAM principal

If the requester is an IAM principal, Amazon S3 must determine if the parent AWS account to which the principal belongs has granted the principal necessary permission to perform the operation. In addition, if the request is for a bucket operation, such as a request to list the bucket content, Amazon S3 must verify that the bucket owner has granted permission for the requester to perform the operation. To perform a specific operation on a resource, an IAM principal needs permission from both the parent AWS account to which it belongs and the AWS account that owns the resource.

Example Requester is an IAM principal - If the request is for an operation on an object that the bucket owner doesn't own.

If the request is for an operation on an object that the bucket owner does not own, in addition to making sure the requester has permissions from the object owner, Amazon S3 must also check the bucket policy to ensure the bucket owner has not set explicit deny on the object. A bucket owner (who pays the bill) can explicitly deny access to objects in the bucket regardless of who owns it. The bucket owner can also delete any object in the bucket.

By default, when another AWS account uploads an object to your S3 bucket, that account (the object writer) owns the object, has access to it, and can grant other users access to it through ACLs. You can use Object Ownership to change this default behavior so that ACLs are disabled and you, as the bucket owner, automatically own every object in your bucket. As a result, access control for your data is based on policies, such as IAM policies, S3 bucket policies, virtual private cloud (VPC) endpoint policies, and AWS Organizations service control policies (SCPs). For more information, see [Controlling ownership of objects and disabling ACLs for your bucket \(p. 601\)](#).

For more information about how Amazon S3 evaluates access policies to authorize or deny requests for bucket operations and object operations, see the following topics:

Topics

- [How Amazon S3 authorizes a request for a bucket operation \(p. 405\)](#)
- [How Amazon S3 authorizes a request for an object operation \(p. 408\)](#)

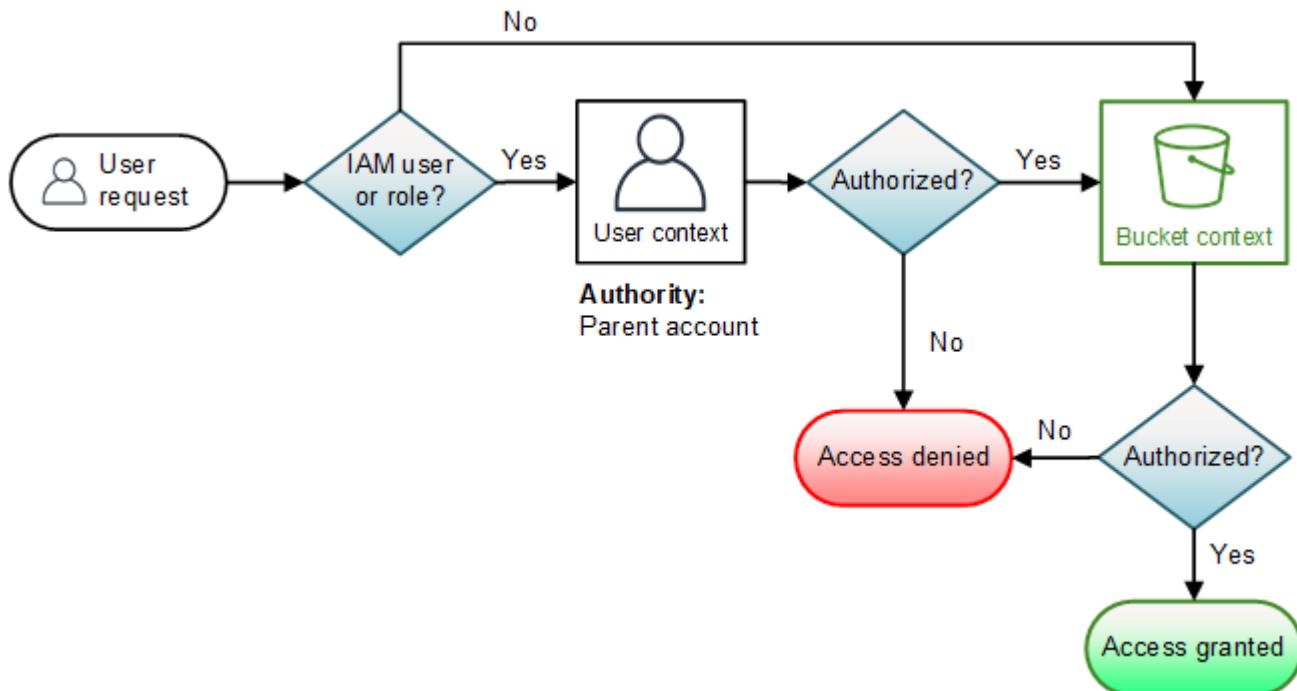
How Amazon S3 authorizes a request for a bucket operation

When Amazon S3 receives a request for a bucket operation, Amazon S3 converts all the relevant permissions into a set of policies to evaluate at run time. Relevant permissions include resource-based permissions (for example, bucket policies and bucket access control lists) and IAM user policies if the request is from an IAM principal. Amazon S3 then evaluates the resulting set of policies in a series of steps according to a specific context—user context or bucket context.

1. **User context** – If the requester is an IAM principal, the principal must have permission from the parent AWS account to which it belongs. In this step, Amazon S3 evaluates a subset of policies owned by the parent account (also referred to as the context authority). This subset of policies includes the user policy that the parent account attaches to the principal. If the parent also owns the resource in the request (in this case, the bucket), Amazon S3 also evaluates the corresponding resource policies (bucket policy and bucket ACL) at the same time. Whenever a request for a bucket operation is made, the server access logs record the canonical ID of the requester. For more information, see [Logging requests using server access logging \(p. 978\)](#).
2. **Bucket context** – The requester must have permissions from the bucket owner to perform a specific bucket operation. In this step, Amazon S3 evaluates a subset of policies owned by the AWS account that owns the bucket.

The bucket owner can grant permission by using a bucket policy or bucket ACL. Note that, if the AWS account that owns the bucket is also the parent account of an IAM principal, then it can configure bucket permissions in a user policy.

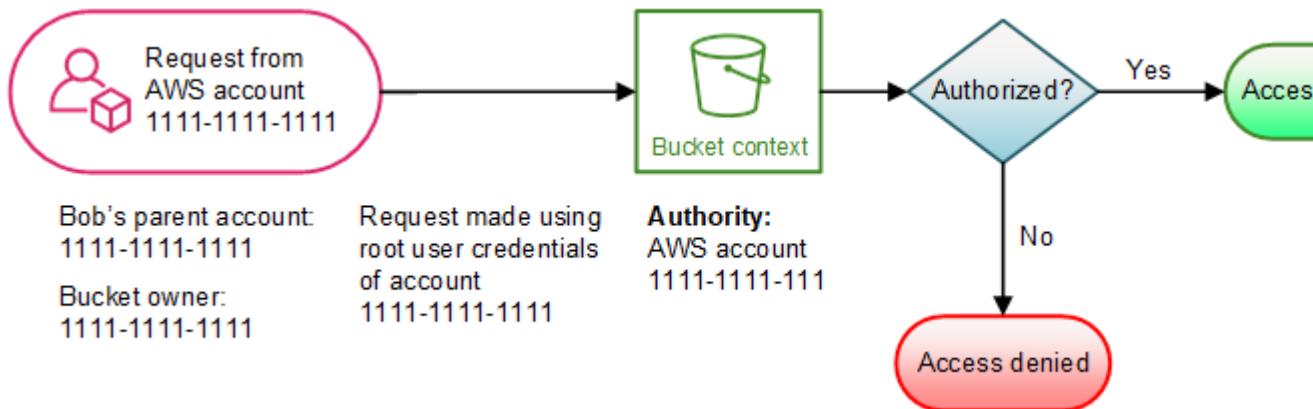
The following is a graphical illustration of the context-based evaluation for bucket operation.



The following examples illustrate the evaluation logic.

Example 1: Bucket operation requested by bucket owner

In this example, the bucket owner sends a request for a bucket operation using the root credentials of the AWS account.

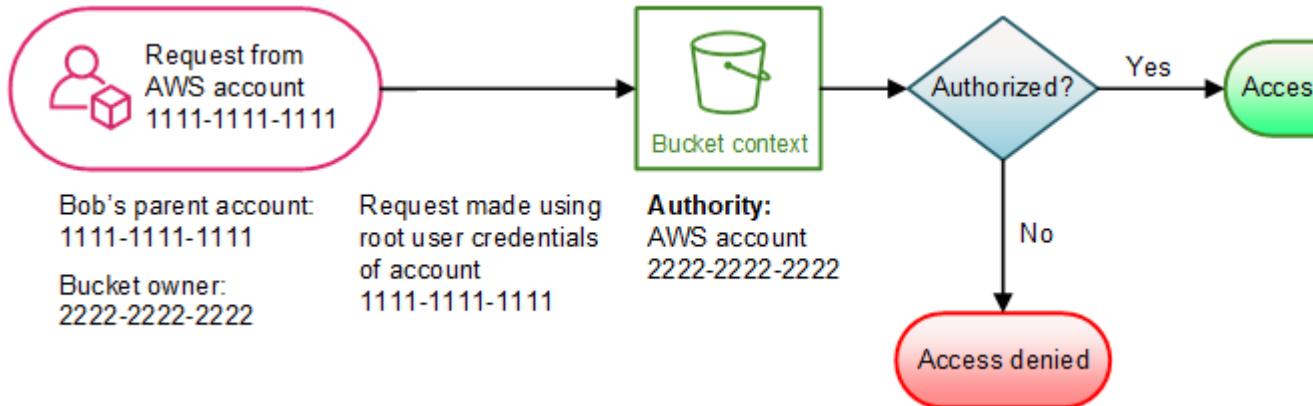


Amazon S3 performs the context evaluation as follows:

1. Because the request is made by using root credentials of an AWS account, the user context is not evaluated.
2. In the bucket context, Amazon S3 reviews the bucket policy to determine if the requester has permission to perform the operation. Amazon S3 authorizes the request.

Example 2: Bucket operation requested by an AWS account that is not the bucket owner

In this example, a request is made using root credentials of AWS account 1111-1111-1111 for a bucket operation owned by AWS account 2222-2222-2222. No IAM users are involved in this request.

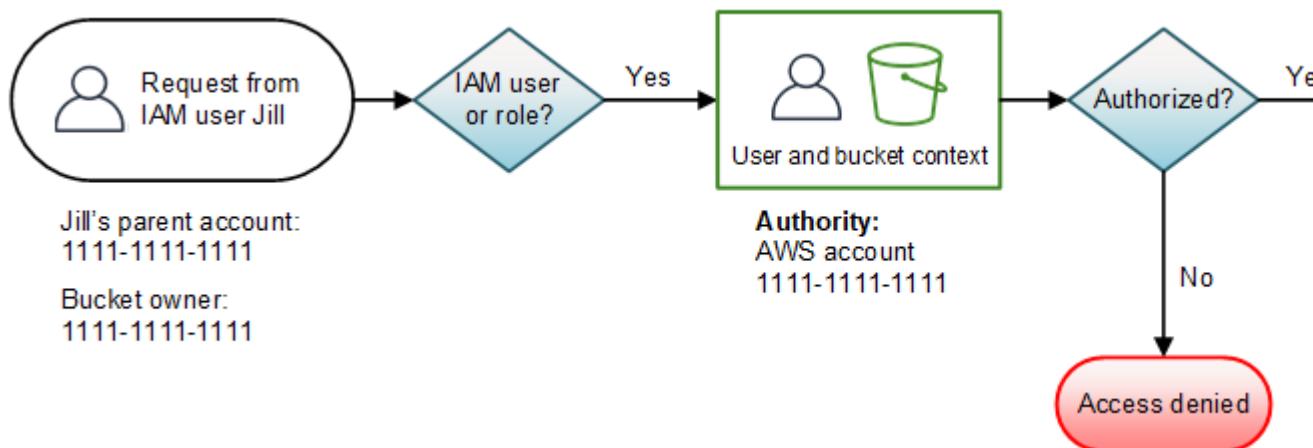


In this case, Amazon S3 evaluates the context as follows:

1. Because the request is made using root credentials of an AWS account, the user context is not evaluated.
2. In the bucket context, Amazon S3 examines the bucket policy. If the bucket owner (AWS account 2222-2222-2222) has not authorized AWS account 1111-1111-1111 to perform the requested operation, Amazon S3 denies the request. Otherwise, Amazon S3 grants the request and performs the operation.

Example 3: Bucket operation requested by an IAM principal whose parent AWS account is also the bucket owner

In the example, the request is sent by Jill, an IAM user in AWS account 1111-1111-1111, which also owns the bucket.



Amazon S3 performs the following context evaluation:

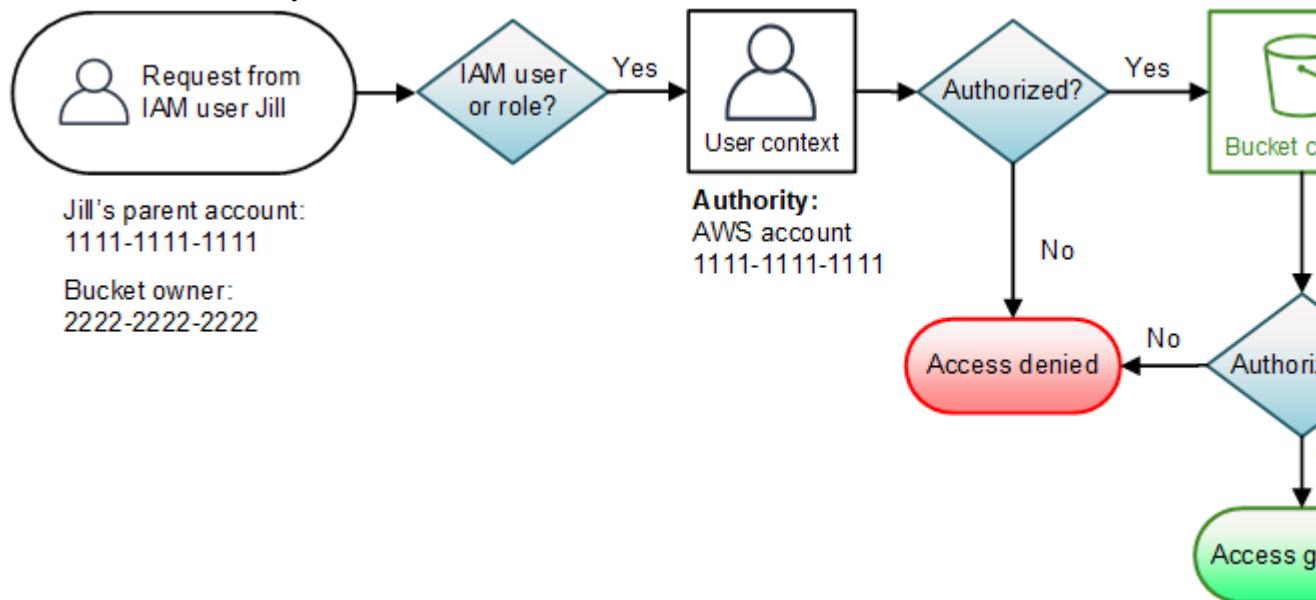
1. Because the request is from an IAM principal, in the user context, Amazon S3 evaluates all policies that belong to the parent AWS account to determine if Jill has permission to perform the operation.

In this example, parent AWS account 1111-1111-1111, to which the principal belongs, is also the bucket owner. As a result, in addition to the user policy, Amazon S3 also evaluates the bucket policy and bucket ACL in the same context, because they belong to the same account.

- Because Amazon S3 evaluated the bucket policy and bucket ACL as part of the user context, it does not evaluate the bucket context.

Example 4: Bucket operation requested by an IAM principal whose parent AWS account is not the bucket owner

In this example, the request is sent by Jill, an IAM user whose parent AWS account is 1111-1111-1111, but the bucket is owned by another AWS account, 2222-2222-2222.



Jill will need permissions from both the parent AWS account and the bucket owner. Amazon S3 evaluates the context as follows:

- Because the request is from an IAM principal, Amazon S3 evaluates the user context by reviewing the policies authored by the account to verify that Jill has the necessary permissions. If Jill has permission, then Amazon S3 moves on to evaluate the bucket context; if not, it denies the request.
- In the bucket context, Amazon S3 verifies that bucket owner 2222-2222-2222 has granted Jill (or her parent AWS account) permission to perform the requested operation. If she has that permission, Amazon S3 grants the request and performs the operation; otherwise, Amazon S3 denies the request.

How Amazon S3 authorizes a request for an object operation

When Amazon S3 receives a request for an object operation, it converts all the relevant permissions—resource-based permissions (object access control list (ACL), bucket policy, bucket ACL) and IAM user policies—into a set of policies to be evaluated at run time. It then evaluates the resulting set of policies in a series of steps. In each step, it evaluates a subset of policies in three specific contexts—user context, bucket context, and object context.

- User context** – If the requester is an IAM principal, the principal must have permission from the parent AWS account to which it belongs. In this step, Amazon S3 evaluates a subset of policies owned by the parent account (also referred as the context authority). This subset of policies includes the user policy

that the parent attaches to the principal. If the parent also owns the resource in the request (bucket, object), Amazon S3 evaluates the corresponding resource policies (bucket policy, bucket ACL, and object ACL) at the same time.

Note

If the parent AWS account owns the resource (bucket or object), it can grant resource permissions to its IAM principal by using either the user policy or the resource policy.

2. **Bucket context** – In this context, Amazon S3 evaluates policies owned by the AWS account that owns the bucket.

If the AWS account that owns the object in the request is not same as the bucket owner, in the bucket context Amazon S3 checks the policies if the bucket owner has explicitly denied access to the object. If there is an explicit deny set on the object, Amazon S3 does not authorize the request.

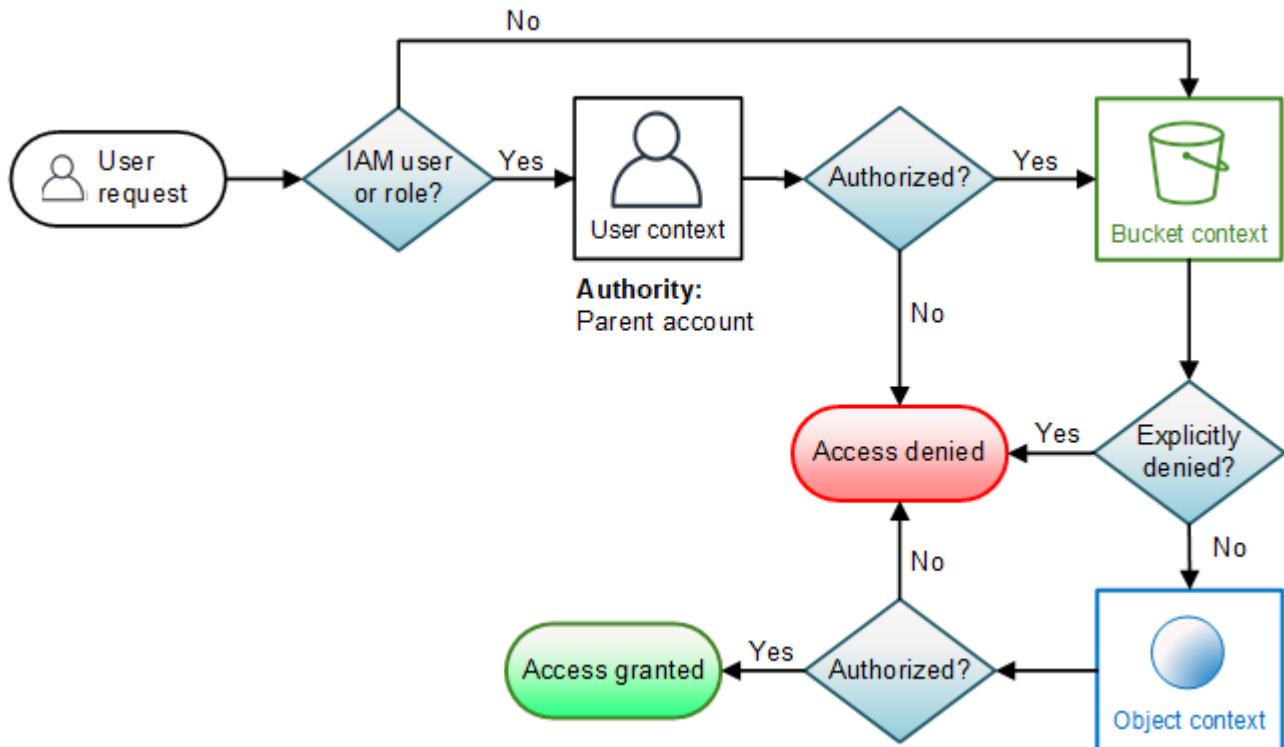
3. **Object context** – The requester must have permissions from the object owner to perform a specific object operation. In this step, Amazon S3 evaluates the object ACL.

Note

If bucket and object owners are the same, access to the object can be granted in the bucket policy, which is evaluated at the bucket context. If the owners are different, the object owners must use an object ACL to grant permissions. If the AWS account that owns the object is also the parent account to which the IAM principal belongs, it can configure object permissions in a user policy, which is evaluated at the user context. For more information about using these access policy alternatives, see [Access policy guidelines \(p. 400\)](#).

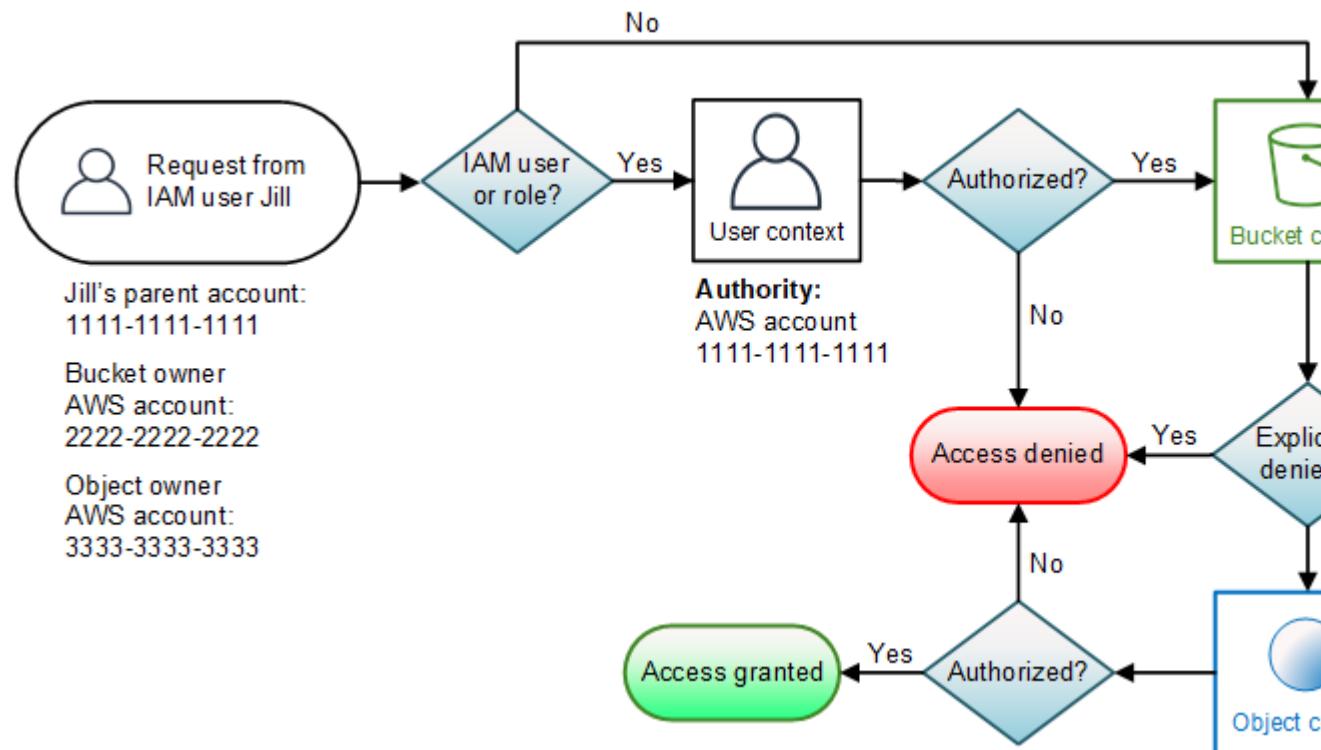
If you as the bucket owner want to own all the objects in your bucket and use bucket policies or IAM-based policies to manage access to these objects, you can apply the bucket owner enforced setting for Object Ownership. With this setting, you as the bucket owner automatically own and have full control over every object in your bucket. Bucket and object ACLs can't be edited and are no longer considered for access. For more information, see [Controlling ownership of objects and disabling ACLs for your bucket \(p. 601\)](#).

The following is an illustration of the context-based evaluation for an object operation.



Example 1: Object operation request

In this example, IAM user Jill, whose parent AWS account is 1111-1111-1111, sends an object operation request (for example, Get object) for an object owned by AWS account 3333-3333-3333 in a bucket owned by AWS account 2222-2222-2222.



Jill will need permission from the parent AWS account, the bucket owner, and the object owner. Amazon S3 evaluates the context as follows:

- Because the request is from an IAM principal, Amazon S3 evaluates the user context to verify that the parent AWS account 1111-1111-1111 has given Jill permission to perform the requested operation. If she has that permission, Amazon S3 evaluates the bucket context. Otherwise, Amazon S3 denies the request.
- In the bucket context, the bucket owner, AWS account 2222-2222-2222, is the context authority. Amazon S3 evaluates the bucket policy to determine if the bucket owner has explicitly denied Jill access to the object.
- In the object context, the context authority is AWS account 3333-3333-3333, the object owner. Amazon S3 evaluates the object ACL to determine if Jill has permission to access the object. If she does, Amazon S3 authorizes the request.

Bucket policies and user policies

Bucket policies and user policies are two access policy options available for granting permission to your Amazon S3 resources. Both use JSON-based access policy language.

The topics in this section describe the key policy language elements, with emphasis on Amazon S3-specific details, and provide example bucket and user policies. We recommend that you first review the introductory topics that explain the basic concepts and options available for you to manage access to your Amazon S3 resources. For more information, see [Identity and access management in Amazon S3 \(p. 394\)](#).

Important

Bucket policies are limited to 20 KB in size.

Topics

- [Policies and Permissions in Amazon S3 \(p. 411\)](#)
- [Using bucket policies \(p. 487\)](#)
- [Using IAM user policies \(p. 499\)](#)
- [Example walkthroughs: Managing access to your Amazon S3 resources \(p. 524\)](#)
- [Using service-linked roles for Amazon S3 Storage Lens \(p. 550\)](#)

Policies and Permissions in Amazon S3

This page provides an overview of bucket and user policies in Amazon S3 and describes the basic elements of a policy. Each listed element links to more details about that element and examples of how to use it.

For a complete list of Amazon S3 actions, resources, and conditions, see [Actions, resources, and condition keys for Amazon S3 \(p. 431\)](#)

In its most basic sense, a policy contains the following elements:

- **Resources (p. 412)** – Buckets, objects, access points, and jobs are the Amazon S3 resources for which you can allow or deny permissions. In a policy, you use the Amazon Resource Name (ARN) to identify the resource. For more information, see [Amazon S3 resources \(p. 412\)](#).
- **Actions (p. 415)** – For each resource, Amazon S3 supports a set of operations. You identify resource operations that you will allow (or deny) by using action keywords.

For example, the `s3>ListBucket` permission allows the user to use the Amazon S3 [GET Bucket \(List Objects\)](#) operation. For more information about using Amazon S3 actions, see [Amazon S3 actions \(p. 415\)](#). For a complete list of Amazon S3 actions, see [Actions](#).

- **Effect** – What the effect will be when the user requests the specific action—this can be either *allow* or *deny*.

If you do not explicitly grant access to (allow) a resource, access is implicitly denied. You can also explicitly deny access to a resource. You might do this to make sure that a user can't access the resource, even if a different policy grants access. For more information, see [IAM JSON Policy Elements: Effect](#).

- **Principal (p. 414)** – The account or user who is allowed access to the actions and resources in the statement. In a bucket policy, the principal is the user, account, service, or other entity that is the recipient of this permission. For more information, see [Principals \(p. 414\)](#).
- **Condition (p. 420)** – Conditions for when a policy is in effect. You can use AWS-wide keys and Amazon S3-specific keys to specify conditions in an Amazon S3 access policy. For more information, see [Amazon S3 condition key examples \(p. 420\)](#).

The following example bucket policy shows the effect, principal, action, and resource elements. The policy allows Dave, a user in account **Account-ID**, s3:GetObject, s3:GetBucketLocation, and s3>ListBucket Amazon S3 permissions on the awsexamplebucket1 bucket.

```
{  
    "Version": "2012-10-17",  
    "Id": "ExamplePolicy01",  
    "Statement": [  
        {  
            "Sid": "ExampleStatement01",  
            "Effect": "Allow",  
            "Principal": {  
                "AWS": "arn:aws:iam::123456789012:user/Dave"  
            },  
            "Action": [  
                "s3:GetObject",  
                "s3:GetBucketLocation",  
                "s3>ListBucket"  
            ],  
            "Resource": [  
                "arn:aws:s3::::awsexamplebucket1/*",  
                "arn:aws:s3::::awsexamplebucket1"  
            ]  
        }  
    ]  
}
```

For more, see the topics below. For complete policy language information, see [Policies and Permissions](#) and [IAM JSON Policy Reference](#) in the *IAM User Guide*.

Topics

- [Amazon S3 resources \(p. 412\)](#)
- [Principals \(p. 414\)](#)
- [Amazon S3 actions \(p. 415\)](#)
- [Amazon S3 condition keys \(p. 418\)](#)
- [Actions, resources, and condition keys for Amazon S3 \(p. 431\)](#)

Amazon S3 resources

The following common Amazon Resource Name (ARN) format identifies resources in AWS:

```
arn:partition:service:region:namespace:relative-id
```

For information about ARNs, see [Amazon Resource Names \(ARNs\)](#) in the *AWS General Reference*.

For information about resources, see [IAM JSON Policy Elements: Resource](#) in the *IAM User Guide*.

An Amazon S3 ARN excludes the AWS Region and namespace, but includes the following:

- **Partition** - aws is a common partition name. If your resources are in the China (Beijing) Region, aws-cn is the partition name.
- **Service** - s3.
- **Relative ID** - bucket-name or a bucket-name/object-key. You can use wild cards.

The ARN format for Amazon S3 resources reduces to the following:

```
arn:aws:s3:::bucket_name/key_name
```

For a complete list of Amazon S3 resources, see [Actions, resources, and condition keys for Amazon S3 \(p. 431\)](#).

To find the ARN for an S3 bucket, you can look at the Amazon S3 console **Bucket Policy** or **CORS configuration** permissions pages. For more information, see the following topics:

- [Adding a bucket policy using the Amazon S3 console \(p. 488\)](#)
- [CORS configuration \(p. 574\)](#)

Amazon S3 ARN examples

The following are examples of Amazon S3 resource ARNs.

Bucket Name and Object Key Specified

The following ARN identifies the `/developers/design_info.doc` object in the `examplebucket` bucket.

```
arn:aws:s3:::examplebucket/developers/design_info.doc
```

Wildcards

You can use wildcards as part of the resource ARN. You can use wildcard characters (*) and (?) within any ARN segment (the parts separated by colons). An asterisk (*) represents any combination of zero or more characters, and a question mark (?) represents any single character. You can use multiple * or ? characters in each segment, but a wildcard cannot span segments.

- The following ARN uses the wildcard * in the relative-ID part of the ARN to identify all objects in the `examplebucket` bucket.

```
arn:aws:s3:::examplebucket/*
```

- The following ARN uses * to indicate all Amazon S3 resources (all S3 buckets and objects).

```
arn:aws:s3::::*
```

- The following ARN uses both wildcards, * and ?, in the relative-ID part. It identifies all objects in buckets such as `example1bucket`, `example2bucket`, `example3bucket`, and so on.

```
arn:aws:s3:::example?bucket/*
```

Policy Variables

You can use policy variables in Amazon S3 ARNs. At policy evaluation time, these predefined variables are replaced by their corresponding values. Suppose that you organize your bucket as a collection of folders, one folder for each of your users. The folder name is the same as the user name. To grant users permission to their folders, you can specify a policy variable in the resource ARN:

```
arn:aws:s3:::bucket_name/developers/${aws:username}/
```

At runtime, when the policy is evaluated, the variable `${aws:username}` in the resource ARN is substituted with the user name making the request.

Principals

The `Principal` element specifies the user, account, service, or other entity that is allowed or denied access to a resource. The following are examples of specifying `Principal`. For more information, see [Principal](#) in the *IAM User Guide*.

Grant permissions to an AWS account

To grant permissions to an AWS account, identify the account using the following format.

```
"AWS" : "account-ARN"
```

The following are examples.

```
"Principal": {"AWS": "arn:aws:iam::AccountIDWithoutHyphens:root"}
```

```
"Principal": {"AWS": ["arn:aws:iam::AccountID1WithoutHyphens:root", "arn:aws:iam::AccountID2WithoutHyphens:root"]}
```

Amazon S3 also supports a canonical user ID, which is an obfuscated form of the AWS account ID. You can specify this ID using the following format.

```
"CanonicalUser": "64-digit-alphanumeric-value"
```

The following is an example.

```
"Principal": {"CanonicalUser": "64-digit-alphanumeric-value"}
```

For information about how to find the canonical user ID for your account, see [Finding Your Account Canonical User ID](#).

Important

When you use a canonical user ID in a policy, Amazon S3 might change the canonical ID to the corresponding AWS account ID. This does not impact the policy because both of these IDs identify the same account.

Grant permissions to an IAM user

To grant permission to an IAM user within your account, you must provide an `"AWS" : "user-ARN"` name-value pair.

```
"Principal": {"AWS": "arn:aws:iam::account-number-without-hyphens:user/username"}
```

For detailed examples that provide step-by-step instructions, see [Example 1: Bucket owner granting its users bucket permissions \(p. 527\)](#) and [Example 3: Bucket owner granting permissions to objects it does not own \(p. 536\)](#).

Grant anonymous permissions

To grant permission to everyone, also referred as anonymous access, you set the wildcard ("`*`") as the `Principal` value. For example, if you configure your bucket as a website, you want all the objects in the bucket to be publicly accessible.

```
"Principal": "*"
```

```
"Principal": {"AWS": "*"}"
```

Using "Principal": "*" with an Allow effect in a resource-based policy allows anyone, even if they're not signed in to AWS, to access your resource.

Using "Principal" : { "AWS" : "*" } with an Allow effect in a resource-based policy allows any root user, IAM user, assumed-role session, or federated user in any account in the same partition to access your resource. For more information, see [All principals](#) in the *IAM User Guide*.

You cannot use a wildcard to match part of a principal name or ARN.

Important

Because anyone can create an AWS account, the **security level** of these two methods is equivalent, even though they function differently.

Warning

Use caution when granting anonymous access to your Amazon S3 bucket. When you grant anonymous access, anyone in the world can access your bucket. We highly recommend that you never grant any kind of anonymous write access to your S3 bucket.

Require access through CloudFront URLs

You can require that your users access your Amazon S3 content by using Amazon CloudFront URLs instead of Amazon S3 URLs. To do this, create a CloudFront origin access identity (OAI). Then, change the permissions either on your bucket or on the objects in your bucket. The format for specifying the OAI in a Principal statement is as follows.

```
"Principal": {"CanonicalUser": "Amazon S3 Canonical User ID assigned to origin access identity"}
```

For more information, see [Using an Origin Access Identity to Restrict Access to Your Amazon S3 Content](#) in the *Amazon CloudFront Developer Guide*.

Amazon S3 actions

Amazon S3 defines a set of permissions that you can specify in a policy. These are keywords, each of which maps to a specific Amazon S3 operation. For more information about Amazon S3 operations, see [Actions](#) in the *Amazon Simple Storage Service API Reference*.

To see how to specify permissions in an Amazon S3 policy, review the following example policies. For a list of Amazon S3 actions, resources, and condition keys for use in policies, see [Actions, resources, and condition keys for Amazon S3 \(p. 431\)](#). For a complete list of Amazon S3 actions, see [Actions](#).

Topics

- [Example — Object operations \(p. 415\)](#)
- [Example — Bucket operations \(p. 416\)](#)
- [Example — Bucket subresource operations \(p. 416\)](#)
- [Example — Account operations \(p. 418\)](#)

Example — Object operations

The following example bucket policy grants the s3:PutObject and the s3:PutObjectAcl permissions to a user (Dave). If you remove the Principal element, you can attach the policy to a user. These are object operations. Accordingly, the relative-id portion of the Resource ARN identifies objects (awsexamplebucket1/*). For more information, see [Amazon S3 resources \(p. 412\)](#).

```
{  
    "Version": "2012-10-17",  
    "Statement": [  
        {
```

```
        "Sid": "statement1",
        "Effect": "Allow",
        "Principal": {
            "AWS": "arn:aws:iam::12345678901:user/Dave"
        },
        "Action": [
            "s3:PutObject",
            "s3:PutObjectAcl"
        ],
        "Resource": "arn:aws:s3:::awsexamplebucket1/*"
    }
}
```

Permissions for All Amazon S3 Actions

You can use a wildcard to grant permission for all Amazon S3 actions.

```
"Action": "*"
```

Example — Bucket operations

The following example user policy grants the `s3:CreateBucket`, `s3>ListAllMyBuckets`, and the `s3:GetBucketLocation` permissions to a user. For all these permissions, you set the `relative-id` part of the Resource ARN to `"*"`. For all other bucket actions, you must specify a bucket name. For more information, see [Amazon S3 resources \(p. 412\)](#).

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "statement1",
            "Effect": "Allow",
            "Action": [
                "s3:CreateBucket",
                "s3>ListAllMyBuckets",
                "s3:GetBucketLocation"
            ],
            "Resource": [
                "arn:aws:s3:::/*"
            ]
        }
    ]
}
```

Policy for console access

If a user wants to use the AWS Management Console to view buckets and the contents of any of those buckets, the user must have the `s3>ListAllMyBuckets` and `s3:GetBucketLocation` permissions. For an example, see *Policy for Console Access* in the blog post [Writing IAM Policies: How to Grant Access to an S3 Bucket](#).

Example — Bucket subresource operations

The following user policy grants the `s3:GetBucketAcl` permission on the `DOC-EXAMPLE-BUCKET1` bucket to user Dave.

```
{
    "Version": "2012-10-17",
    "Statement": [
```

```
{
    "Sid": "statement1",
    "Effect": "Allow",
    "Principal": {
        "AWS": "arn:aws:iam::123456789012:user/Dave"
    },
    "Action": [
        "s3:GetBucketAcl"
    ],
    "Resource": [
        "arn:aws:s3:::DOC-EXAMPLE-BUCKET1"
    ]
}
]
```

DELETE Object permissions

You can delete objects either by explicitly calling the DELETE Object API or by configuring its lifecycle (see [Managing your storage lifecycle \(p. 701\)](#)) so that Amazon S3 can remove the objects when their lifetime expires. To explicitly block users or accounts from deleting objects, you must explicitly deny them `s3:DeleteObject`, `s3:DeleteObjectVersion`, and `s3:PutLifecycleConfiguration` permissions.

Explicit deny

By default, users have no permissions. But as you create users, add users to groups, and grant them permissions, they might get certain permissions that you didn't intend to grant. To avoid such permission loopholes, you can write a stricter access policy by adding explicit deny.

The preceding bucket policy grants the `s3:GetBucketAcl` permission `DOC-EXAMPLE-BUCKET1` bucket to user Dave. In this example, you explicitly deny the user Dave DELETE Object permissions. Explicit deny always supersedes any other permission granted. The following is the revised access policy example with explicit deny added.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "statement1",
            "Effect": "Allow",
            "Principal": {
                "AWS": "arn:aws:iam::123456789012:user/Dave"
            },
            "Action": [
                "s3:GetObjectVersion",
                "s3:GetBucketAcl"
            ],
            "Resource": [
                "arn:aws:s3:::DOC-EXAMPLE-BUCKET1",
                "arn:aws:s3:::DOC-EXAMPLE-BUCKET1/*"
            ]
        },
        {
            "Sid": "statement2",
            "Effect": "Deny",
            "Principal": {
                "AWS": "arn:aws:iam::123456789012:user/Dave"
            },
            "Action": [
                "s3:DeleteObject",
                "s3:DeleteObjectVersion",
                "s3:PutLifecycleConfiguration"
            ]
        }
    ]
}
```

```

        ],
        "Resource": [
            "arn:aws:s3:::DOC-EXAMPLE-BUCKET1",
            "arn:aws:s3:::DOC-EXAMPLE-BUCKET1/*"
        ]
    }
}

```

Example — Account operations

The following example user policy grants the `s3:GetAccountPublicAccessBlock` permission to a user. For these permissions, you set the `Resource` value to `"*"`. For more information, see [Amazon S3 resources \(p. 412\)](#).

```

{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "statement1",
            "Effect": "Allow",
            "Action": [
                "s3:GetAccountPublicAccessBlock"
            ],
            "Resource": [
                "*"
            ]
        }
    ]
}

```

Amazon S3 condition keys

The access policy language enables you to specify conditions when granting permissions. To specify conditions for when a policy is in effect, you can use the optional `Condition` element, or `Condition` block, to specify conditions for when a policy is in effect. You can use predefined AWS-wide keys and Amazon S3-specific keys to specify conditions in an Amazon S3 access policy.

In the `Condition` element, you build expressions in which you use Boolean operators (equal, less than, etc.) to match your condition against values in the request. For example, when granting a user permission to upload an object, the bucket owner can require that the object be publicly readable by adding the `StringEquals` condition, as shown here.

```

{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "statement1",
            "Effect": "Allow",
            "Action": "s3:PutObject",
            "Resource": [
                "arn:aws:s3:::awsexamplebucket1/*"
            ],
            "Condition": {
                "StringEquals": {
                    "s3:x-amz-acl": "public-read"
                }
            }
        }
    ]
}

```

In the example, the Condition block specifies the `StringEquals` condition that is applied to the specified key-value pair, `"s3:x-amz-acl": ["public-read"]`. There is a set of predefined keys that you can use in expressing a condition. The example uses the `s3:x-amz-acl` condition key. This condition requires the user to include the `x-amz-acl` header with value `public-read` in every PUT object request.

Topics

- [AWS-wide condition keys \(p. 419\)](#)
- [Amazon S3-specific condition keys \(p. 420\)](#)
- [Amazon S3 condition key examples \(p. 420\)](#)

AWS-wide condition keys

AWS provides a set of common keys that are supported by all AWS services that support policies. These keys are called *AWS-wide keys* and use the prefix `aws:`. For a complete list of AWS-wide condition keys, see [Available AWS Keys for Conditions](#) in the *IAM User Guide*. You can use AWS-wide condition keys in Amazon S3. The following example bucket policy allows authenticated users permission to use the `s3:GetObject` action if the request originates from a specific range of IP addresses (`192.0.2.0.*`), unless the IP address is `192.0.2.188`. In the condition block, the `IpAddress` and the `NotIpAddress` are conditions, and each condition is provided a key-value pair for evaluation. Both the key-value pairs in this example use the `aws:SourceIp` AWS-wide key.

Note

The `IpAddress` and `NotIpAddress` key values specified in the condition uses CIDR notation as described in RFC 4632. For more information, see <http://www.rfc-editor.org/rfc/rfc4632.txt>.

```
{  
    "Version": "2012-10-17",  
    "Id": "S3PolicyId1",  
    "Statement": [  
        {  
            "Sid": "statement1",  
            "Effect": "Allow",  
            "Principal": "*",  
            "Action": "s3:GetObject",  
            "Resource": "arn:aws:s3:::awsexamplebucket1/*",  
            "Condition": {  
                "IpAddress": {  
                    "aws:SourceIp": "192.0.2.0/24"  
                },  
                "NotIpAddress": {  
                    "aws:SourceIp": "192.0.2.188/32"  
                }  
            }  
        }  
    ]  
}
```

You can also use other AWS-wide condition keys in Amazon S3 policies. For example, you can specify the `aws:SourceVpce` and `aws:SourceVpc` condition keys in bucket policies for VPC endpoints. For specific examples, see [Controlling access from VPC endpoints with bucket policies \(p. 489\)](#).

Note

For some AWS global condition keys, only certain resource types are supported. Therefore, check whether Amazon S3 supports the global condition key and resource type that you want to use, or if you'll need to use an Amazon S3-specific condition key instead. For a complete list of supported resource types and condition keys for Amazon S3, see [Actions, resources, and conditions](#).

Amazon S3-specific condition keys

You can use Amazon S3 condition keys with specific Amazon S3 actions. Each condition key maps to the same name request header allowed by the API on which the condition can be set. Amazon S3-specific condition keys dictate the behavior of the same name request headers. For a complete list of Amazon S3-specific condition keys, see [Actions, resources, and condition keys for Amazon S3 \(p. 431\)](#).

For example, the condition key `s3:x-amz-acl` that you can use to grant condition permission for the `s3:PutObject` permission defines behavior of the `x-amz-acl` request header that the PUT Object API supports. The condition key `s3:VersionId` that you can use to grant conditional permission for the `s3:GetObjectVersion` permission defines behavior of the `versionId` query parameter that you set in a GET Object request.

The following bucket policy grants the `s3:PutObject` permission for two AWS accounts if the request includes the `x-amz-acl` header making the object publicly readable. The Condition block uses the `StringEquals` condition, and it is provided a key-value pair, `"s3:x-amz-acl": ["public-read"]`, for evaluation. In the key-value pair, the `s3:x-amz-acl` is an Amazon S3-specific key, as indicated by the prefix `s3:`.

```
{  
    "Version": "2012-10-17",  
    "Statement": [  
        {  
            "Sid": "AddCannedAcl",  
            "Effect": "Allow",  
            "Principal": {  
                "AWS": [  
                    "arn:aws:iam::Account1-ID:root",  
                    "arn:aws:iam::Account2-ID:root"  
                ]  
            },  
            "Action": "s3:PutObject",  
            "Resource": ["arn:aws:s3:::awsexamplebucket1/*"],  
            "Condition": {  
                "StringEquals": {  
                    "s3:x-amz-acl": ["public-read"]  
                }  
            }  
        }  
    ]  
}
```

Important

Not all conditions make sense for all actions. For example, it makes sense to include an `s3:LocationConstraint` condition on a policy that grants the `s3:CreateBucket` Amazon S3 permission. However, it does not make sense to include this condition on a policy that grants the `s3:GetObject` permission. Amazon S3 can test for semantic errors of this type that involve Amazon S3-specific conditions. However, if you are creating a policy for an IAM user and you include a semantically invalid Amazon S3 condition, no error is reported because IAM cannot validate Amazon S3 conditions.

Amazon S3 condition key examples

You can use access policy language to specify conditions when you grant permissions. You can use the optional `Condition` element, or `Condition` block to specify conditions for when a policy is in effect.

For policies that use Amazon S3 condition keys for object and bucket operations, see the following examples. For more information about condition keys, see [Amazon S3 condition keys \(p. 418\)](#). For a complete list of Amazon S3 actions, condition keys, and resources that you can specify in policies, see [Actions, resources, and condition keys for Amazon S3 \(p. 431\)](#).

Examples — Amazon S3 condition keys for object operations

This section provides examples that show you how you can use Amazon S3-specific condition keys for object operations. For a complete list of Amazon S3 actions, condition keys, and resources that you can specify in policies, see [Actions, resources, and condition keys for Amazon S3 \(p. 431\)](#).

Several of the example policies show how you can use conditions keys with [PUT Object](#) operations. PUT Object operations allow access control list (ACL)-specific headers that you can use to grant ACL-based permissions. Using these keys, the bucket owner can set a condition to require specific access permissions when the user uploads an object. You can also grant ACL-based permissions with the PutObjectAcl operation. For more information, see [PutObjectAcl](#) in the *Amazon S3 Amazon Simple Storage Service API Reference*. For more information about ACLs, see [Access control list \(ACL\) overview \(p. 554\)](#).

Topics

- [Example 1: Granting s3:PutObject permission with a condition requiring the bucket owner to get full control \(p. 421\)](#)
- [Example 2: Granting s3:PutObject permission requiring objects stored using server-side encryption \(p. 423\)](#)
- [Example 3: Granting s3:PutObject permission to copy objects with a restriction on the copy source \(p. 423\)](#)
- [Example 4: Granting access to a specific version of an object \(p. 425\)](#)
- [Example 5: Restricting object uploads to objects with a specific storage class \(p. 425\)](#)
- [Example 6: Granting permissions based on object tags \(p. 426\)](#)
- [Example 7: Restricting access by the AWS account ID of the bucket owner \(p. 426\)](#)
- [Example 8: Requiring a minimum TLS version \(p. 426\)](#)

Example 1: Granting s3:PutObject permission with a condition requiring the bucket owner to get full control

The [PUT Object](#) operation allows access control list (ACL)-specific headers that you can use to grant ACL-based permissions. Using these keys, the bucket owner can set a condition to require specific access permissions when the user uploads an object.

Suppose that Account A owns a bucket, and the account administrator wants to grant Dave, a user in Account B, permissions to upload objects. By default, objects that Dave uploads are owned by Account B, and Account A has no permissions on these objects. Because the bucket owner is paying the bills, it wants full permissions on the objects that Dave uploads. The Account A administrator can do this by granting the s3:PutObject permission to Dave, with a condition that the request include ACL-specific headers that either grant full permission explicitly or use a canned ACL. For more information, see [PUT Object](#).

Require the x-amz-full-control header

You can require the x-amz-full-control header in the request with full control permission to the bucket owner. The following bucket policy grants the s3:PutObject permission to user Dave with a condition using the s3:x-amz-grant-full-control condition key, which requires the request to include the x-amz-full-control header.

```
{  
    "Version": "2012-10-17",  
    "Statement": [  
        {  
            "Sid": "statement1",  
            "Effect": "Allow",  
            "Principal": {  
                "AWS": "arn:aws:iam::AccountB-ID:user/Dave"  
            },  
            "Action": "s3:PutObject",  
            "Condition": {  
                "StringEquals": {  
                    "x-amz-grant-full-control": "true"  
                }  
            }  
        }  
    ]  
}
```

```

        "Resource": "arn:aws:s3:::awsexamplebucket1/*",
        "Condition": {
            "StringEquals": {
                "s3:x-amz-grant-full-control": "id=AccountA-CanonicalUserID"
            }
        }
    ]
}

```

Note

This example is about cross-account permission. However, if Dave (who is getting the permission) belongs to the AWS account that owns the bucket, this conditional permission is not necessary. This is because the parent account to which Dave belongs owns objects that the user uploads.

Add explicit deny

The preceding bucket policy grants conditional permission to user Dave in Account B. While this policy is in effect, it is possible for Dave to get the same permission without any condition via some other policy. For example, Dave can belong to a group, and you grant the group s3:PutObject permission without any condition. To avoid such permission loopholes, you can write a stricter access policy by adding explicit deny. In this example, you explicitly deny the user Dave upload permission if he does not include the necessary headers in the request granting full permissions to the bucket owner. Explicit deny always supersedes any other permission granted. The following is the revised access policy example with explicit deny added.

```

{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "statement1",
            "Effect": "Allow",
            "Principal": {
                "AWS": "arn:aws:iam::AccountB-ID:user/AccountBadmin"
            },
            "Action": "s3:PutObject",
            "Resource": "arn:aws:s3:::awsexamplebucket1/*",
            "Condition": {
                "StringEquals": {
                    "s3:x-amz-grant-full-control": "id=AccountA-CanonicalUserID"
                }
            }
        },
        {
            "Sid": "statement2",
            "Effect": "Deny",
            "Principal": {
                "AWS": "arn:aws:iam::AccountB-ID:user/AccountBadmin"
            },
            "Action": "s3:PutObject",
            "Resource": "arn:aws:s3:::awsexamplebucket1/*",
            "Condition": {
                "StringNotEquals": {
                    "s3:x-amz-grant-full-control": "id=AccountA-CanonicalUserID"
                }
            }
        }
    ]
}

```

Test the policy with the AWS CLI

If you have two AWS accounts, you can test the policy using the AWS Command Line Interface (AWS CLI). You attach the policy and use Dave's credentials to test the permission using the following AWS CLI `put-object` command. You provide Dave's credentials by adding the `--profile` parameter. You grant full control permission to the bucket owner by adding the `--grant-full-control` parameter. For more information about setting up and using the AWS CLI, see [Developing with Amazon S3 using the AWS CLI \(p. 1183\)](#).

```
aws s3api put-object --bucket examplebucket --key HappyFace.jpg --body c:\HappyFace.jpg --grant-full-control id="AccountA-CanonicalUserID" --profile AccountBUserProfile
```

Require the `x-amz-acl` header

You can require the `x-amz-acl` header with a canned ACL granting full control permission to the bucket owner. To require the `x-amz-acl` header in the request, you can replace the key-value pair in the Condition block and specify the `s3:x-amz-acl` condition key, as shown in the following example.

```
"Condition": {  
    "StringEquals": {  
        "s3:x-amz-acl": "bucket-owner-full-control"  
    }  
}
```

To test the permission using the AWS CLI, you specify the `--acl` parameter. The AWS CLI then adds the `x-amz-acl` header when it sends the request.

```
aws s3api put-object --bucket examplebucket --key HappyFace.jpg --body c:\HappyFace.jpg --acl "bucket-owner-full-control" --profile AccountBadmin
```

Example 2: Granting `s3:PutObject` permission requiring objects stored using server-side encryption

Suppose that Account A owns a bucket. The account administrator wants to grant Jane, a user in Account A, permission to upload objects with a condition that Jane always request server-side encryption so that Amazon S3 saves objects encrypted. The Account A administrator can accomplish using the `s3:x-amz-server-side-encryption` condition key as shown. The key-value pair in the Condition block specifies the `s3:x-amz-server-side-encryption` key.

```
"Condition": {  
    "StringNotEquals": {  
        "s3:x-amz-server-side-encryption": "AES256"  
    }  
}
```

When testing the permission using the AWS CLI, you must add the required parameter using the `--server-side-encryption` parameter.

```
aws s3api put-object --bucket examplebucket --key HappyFace.jpg --body c:\HappyFace.jpg --server-side-encryption "AES256" --profile AccountBadmin
```

Example 3: Granting `s3:PutObject` permission to copy objects with a restriction on the copy source

In the PUT Object request, when you specify a source object, it is a copy operation (see [PUT Object - Copy](#)). Accordingly, the bucket owner can grant a user permission to copy objects with restrictions on the source, for example:

- Allow copying objects only from the `sourcebucket` bucket.

- Allow copying objects from the source bucket and only the objects whose key name prefix starts with public/ f (for example, sourcebucket/public/*).
- Allow copying only a specific object from the sourcebucket (for example, sourcebucket/example.jpg).

The following bucket policy grants user (Dave) s3:PutObject permission. It allows him to copy objects only with a condition that the request include the s3:x-amz-copy-source header and the header value specify the /awsexamplebucket1/public/* key name prefix.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "cross-account permission to user in your own account",
            "Effect": "Allow",
            "Principal": {
                "AWS": "arn:aws:iam::123456789012:user/Dave"
            },
            "Action": "s3:PutObject",
            "Resource": "arn:aws:s3:::awsexamplebucket1/*"
        },
        {
            "Sid": "Deny your user permission to upload object if copy source is not / bucket/folder",
            "Effect": "Deny",
            "Principal": {
                "AWS": "arn:aws:iam::123456789012:user/Dave"
            },
            "Action": "s3:PutObject",
            "Resource": "arn:aws:s3:::awsexamplebucket1/*",
            "Condition": {
                "StringNotLike": {
                    "s3:x-amz-copy-source": "awsexamplebucket1/public/*"
                }
            }
        }
    ]
}
```

Test the policy with the AWS CLI

You can test the permission using the AWS CLI copy-object command. You specify the source by adding the --copy-source parameter; the key name prefix must match the prefix allowed in the policy. You need to provide the user Dave credentials using the --profile parameter. For more information about setting up the AWS CLI, see [Developing with Amazon S3 using the AWS CLI \(p. 1183\)](#).

```
aws s3api copy-object --bucket awsexamplebucket1 --key HappyFace.jpg
--copy-source examplebucket/public/PublicHappyFace1.jpg --profile AccountADave
```

Give permission to copy only a specific object

The preceding policy uses the StringNotLike condition. To grant permission to copy only a specific object, you must change the condition from StringNotLike to StringNotEquals and then specify the exact object key as shown.

```
"Condition": {
    "StringNotEquals": {
        "s3:x-amz-copy-source": "awsexamplebucket1/public/PublicHappyFace1.jpg"
    }
}
```

Example 4: Granting access to a specific version of an object

Suppose that Account A owns a version-enabled bucket. The bucket has several versions of the HappyFace.jpg object. The account administrator now wants to grant its user Dave permission to get only a specific version of the object. The account administrator can accomplish this by granting Dave s3:GetObjectVersion permission conditionally as shown below. The key-value pair in the Condition block specifies the s3:VersionId condition key. In this case, Dave needs to know the exact object version ID to retrieve the object.

For more information, see [GetObject](#) in the *Amazon Simple Storage Service API Reference*.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "statement1",
            "Effect": "Allow",
            "Principal": {
                "AWS": "arn:aws:iam::123456789012:user/Dave"
            },
            "Action": "s3:GetObjectVersion",
            "Resource": "arn:aws:s3:::examplebucketversionenabled/HappyFace.jpg"
        },
        {
            "Sid": "statement2",
            "Effect": "Deny",
            "Principal": {
                "AWS": "arn:aws:iam::123456789012:user/Dave"
            },
            "Action": "s3:GetObjectVersion",
            "Resource": "arn:aws:s3:::examplebucketversionenabled/HappyFace.jpg",
            "Condition": {
                "StringNotEquals": {
                    "s3:VersionId": "AaaHbAQitwiL_h47_44lRO2DDfLlBO5e"
                }
            }
        }
    ]
}
```

Test the policy with the AWS CLI

You can test the permissions using the AWS CLI get-object command with the --version-id parameter identifying the specific object version. The command retrieves the object and saves it to the OutputFile.jpg file.

```
aws s3api get-object --bucket examplebucketversionenabled --key HappyFace.jpg
OutputFile.jpg --version-id AaaHbAQitwiL_h47_44lRO2DDfLlBO5e --profile AccountADave
```

Example 5: Restricting object uploads to objects with a specific storage class

Suppose that Account A, represented by account ID 123456789012, owns a bucket. The account administrator wants to restrict Dave, a user in Account A, to be able to only upload objects to the bucket that are stored with the STANDARD_IA storage class. To restrict object uploads to a specific storage class, the Account A administrator can use the s3:x-amz-storage-class condition key, as shown in the following example bucket policy.

```
{
    "Version": "2012-10-17",
    "Statement": [
```

```
{  
    "Sid": "statement1",  
    "Effect": "Allow",  
    "Principal": {  
        "AWS": "arn:aws:iam::123456789012:user/Dave"  
    },  
    "Action": "s3:PutObject",  
    "Resource": "arn:aws:s3:::DOC-EXAMPLE-BUCKET1/*",  
    "Condition": {  
        "StringEquals": {  
            "s3:x-amz-storage-class": [  
                "STANDARD_IA"  
            ]  
        }  
    }  
}
```

Example 6: Granting permissions based on object tags

For examples on how to use object tagging condition keys with Amazon S3 operations, see [Tagging and access control policies \(p. 828\)](#).

Example 7: Restricting access by the AWS account ID of the bucket owner

You can use either the `aws:ResourceAccount` or `s3:ResourceAccount` key to write IAM or Virtual Private Cloud endpoint policies that restrict user or application access to the Amazon S3 buckets that are owned by a specific AWS account ID. You can use this condition key to restrict clients within your VPC from accessing buckets that you do not own.

However, be aware that some AWS services rely on access to AWS managed buckets. Therefore, using the `aws:ResourceAccount` or `s3:ResourceAccount` key in your IAM policy might also impact access to these resources.

For more information and examples, see the following resources:

- [Restrict access to buckets in a specified AWS account](#) in the [AWS PrivateLink Guide](#)
- [Restrict access to buckets that Amazon ECR uses](#) in the [Amazon ECR Guide](#)
- [Provide required access to Systems Manager for AWS managed Amazon S3 buckets](#) in the [AWS Systems Manager Guide](#)
- [Limit access to Amazon S3 buckets owned by specific AWS accounts](#) in the [AWS Storage Blog](#)

You can use the `aws:ResourceAccount` or `s3:ResourceAccount` condition key to write IAM or Virtual Private Cloud Endpoint policies that restrict user or application access to the Amazon S3 buckets that are owned by a specific AWS account ID. You can use this condition key to restrict clients within your VPC from accessing buckets that you do not own.

For information and examples, see the following resources:

- [Restricting access to buckets in a specified AWS account](#) in the [AWS PrivateLink Guide](#)
- [Limit access to AWS-owned by specific AWS accounts](#) in the [AWS Storage Blog](#)

Example 8: Requiring a minimum TLS version

You can use the `s3:TlsVersion` condition key to write IAM, Virtual Private Cloud Endpoint (VPCE), or bucket policies that restrict user or application access to Amazon S3 buckets based on the TLS version used by the client. You can use this condition key to write policies that require a minimum TLS version.

Example

This example bucket policy *denies* PutObject requests by clients that have a TLS version lower than 1.2, for example, 1.1 or 1.0.

```
{  
    "Version": "2012-10-17",  
    "Statement": [  
        {  
            "Effect": "Deny",  
            "Principal": "*",  
            "Action": "s3:PutObject",  
            "Resource": [  
                "arn:aws:s3:::DOC-EXAMPLE-BUCKET1",  
                "arn:aws:s3:::DOC-EXAMPLE-BUCKET1/*"  
            ],  
            "Condition": {  
                "NumericLessThan": {  
                    "s3:TlsVersion": 1.2  
                }  
            }  
        }  
    ]  
}
```

Example

This example bucket policy *allows* PutObject requests by clients that have a TLS version higher than 1.1, for example, 1.2, 1.3 or higher.

```
{  
    "Version": "2012-10-17",  
    "Statement": [  
        {  
            "Effect": "Allow",  
            "Principal": "*",  
            "Action": "s3:PutObject",  
            "Resource": [  
                "arn:aws:s3:::DOC-EXAMPLE-BUCKET1",  
                "arn:aws:s3:::DOC-EXAMPLE-BUCKET1/*"  
            ],  
            "Condition": {  
                "NumericGreaterThan": {  
                    "s3:TlsVersion": 1.1  
                }  
            }  
        }  
    ]  
}
```

Examples — Amazon S3 condition keys for bucket operations

This section provides example policies that show you how you can use Amazon S3-specific condition keys for bucket operations.

Topics

- [Example 1: Granting a user permission to create a bucket only in a specific Region \(p. 428\)](#)
- [Example 2: Getting a list of objects in a bucket with a specific prefix \(p. 429\)](#)

- [Example 3: Setting the maximum number of keys \(p. 431\)](#)

Example 1: Granting a user permission to create a bucket only in a specific Region

Suppose that an AWS account administrator wants to grant its user (Dave) permission to create a bucket in the South America (São Paulo) Region only. The account administrator can attach the following user policy granting the s3:CreateBucket permission with a condition as shown. The key-value pair in the Condition block specifies the s3:LocationConstraint key and the sa-east-1 Region as its value.

Note

In this example, the bucket owner is granting permission to one of its users, so either a bucket policy or a user policy can be used. This example shows a user policy.

For a list of Amazon S3 Regions, see [Regions and Endpoints](#) in the *AWS General Reference*.

```
{  
    "Version": "2012-10-17",  
    "Statement": [  
        {  
            "Sid": "statement1",  
            "Effect": "Allow",  
            "Action": "s3:CreateBucket",  
            "Resource": "arn:aws:s3:::*",  
            "Condition": {  
                "StringLike": {  
                    "s3:LocationConstraint": "sa-east-1"  
                }  
            }  
        }  
    ]  
}
```

Add explicit deny

The preceding policy restricts the user from creating a bucket in any other Region except sa-east-1. However, some other policy might grant this user permission to create buckets in another Region. For example, if the user belongs to a group, the group might have a policy attached to it that allows all users in the group permission to create buckets in another Region. To ensure that the user does not get permission to create buckets in any other Region, you can add an explicit deny statement in the above policy.

The Deny statement uses the StringNotLike condition. That is, a create bucket request is denied if the location constraint is not sa-east-1. The explicit deny does not allow the user to create a bucket in any other Region, no matter what other permission the user gets. The below policy includes an explicit deny statement.

```
{  
    "Version": "2012-10-17",  
    "Statement": [  
        {  
            "Sid": "statement1",  
            "Effect": "Allow",  
            "Action": "s3:CreateBucket",  
            "Resource": "arn:aws:s3:::*",  
            "Condition": {  
                "StringLike": {  
                    "s3:LocationConstraint": "sa-east-1"  
                }  
            }  
        },  
        {  
            "Effect": "Deny",  
            "Action": "s3:CreateBucket",  
            "Resource": "arn:aws:s3:::*",  
            "Condition": {  
                "StringNotLike": {  
                    "s3:LocationConstraint": "sa-east-1"  
                }  
            }  
        }  
    ]  
}
```

```
        "Sid": "statement2",
        "Effect": "Deny",
        "Action": "s3:CreateBucket",
        "Resource": "arn:aws:s3:::*",
        "Condition": {
            "StringNotLike": {
                "s3:LocationConstraint": "sa-east-1"
            }
        }
    ]
}
```

Test the policy with the AWS CLI

You can test the policy using the following `create-bucket` AWS CLI command. This example uses the `bucketconfig.txt` file to specify the location constraint. Note the Windows file path. You need to update the bucket name and path as appropriate. You must provide user credentials using the `--profile` parameter. For more information about setting up and using the AWS CLI, see [Developing with Amazon S3 using the AWS CLI \(p. 1183\)](#).

```
aws s3api create-bucket --bucket examplebucket --profile AccountADave --create-bucket-configuration file:///c:/Users/someUser/bucketconfig.txt
```

The `bucketconfig.txt` file specifies the configuration as follows.

```
{"LocationConstraint": "sa-east-1"}
```

Example 2: Getting a list of objects in a bucket with a specific prefix

You can use the `s3:prefix` condition key to limit the response of the [GET Bucket \(ListObjects\)](#) API to key names with a specific prefix. If you are the bucket owner, you can restrict a user to list the contents of a specific prefix in the bucket. This condition key is useful if objects in the bucket are organized by key name prefixes. The Amazon S3 console uses key name prefixes to show a folder concept. Only the console supports the concept of folders; the Amazon S3 API supports only buckets and objects. For more information about using prefixes and delimiters to filter access permissions, see [Controlling access to a bucket with user policies \(p. 499\)](#).

For example, if you have two objects with key names `public/object1.jpg` and `public/object2.jpg`, the console shows the objects under the `public` folder. In the Amazon S3 API, these are objects with prefixes, not objects in folders. However, in the Amazon S3 API, if you organize your object keys using such prefixes, you can grant `s3>ListBucket` permission with the `s3:prefix` condition that will allow the user to get a list of key names with those specific prefixes.

In this example, the bucket owner and the parent account to which the user belongs are the same. So the bucket owner can use either a bucket policy or a user policy. For more information about other condition keys that you can use with the GET Bucket (ListObjects) API, see [ListObjects](#).

User policy

The following user policy grants the `s3>ListBucket` permission (see [GET Bucket \(List Objects\)](#)) with a condition that requires the user to specify the `prefix` in the request with the value `projects`.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "statement1",
            "Effect": "Allow",
            "Action": "s3>ListBucket",
            "Condition": {
                "StringEquals": {
                    "s3:prefix": "projects"
                }
            }
        }
    ]
}
```

```

    "Resource": "arn:aws:s3:::awsexamplebucket1",
    "Condition" : {
        "StringEquals" : {
            "s3:prefix": "projects"
        }
    },
    {
        "Sid": "statement2",
        "Effect": "Deny",
        "Action": "s3>ListBucket",
        "Resource": "arn:aws:s3:::awsexamplebucket1",
        "Condition" : {
            "StringNotEquals" : {
                "s3:prefix": "projects"
            }
        }
    }
]
}

```

The condition restricts the user to listing object keys with the `projects` prefix. The added explicit deny denies the user request for listing keys with any other prefix no matter what other permissions the user might have. For example, it is possible that the user gets permission to list object keys without any restriction, either by updates to the preceding user policy or via a bucket policy. Because explicit deny always supersedes, the user request to list keys other than the `projects` prefix is denied.

Bucket policy

If you add the `Principal` element to the above user policy, identifying the user, you now have a bucket policy as shown.

```

{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "statement1",
            "Effect": "Allow",
            "Principal": {
                "AWS": "arn:aws:iam::123456789012:user/bucket-owner"
            },
            "Action": "s3>ListBucket",
            "Resource": "arn:aws:s3:::awsexamplebucket1",
            "Condition" : {
                "StringEquals" : {
                    "s3:prefix": "projects"
                }
            }
        },
        {
            "Sid": "statement2",
            "Effect": "Deny",
            "Principal": {
                "AWS": "arn:aws:iam::123456789012:user/bucket-owner"
            },
            "Action": "s3>ListBucket",
            "Resource": "arn:aws:s3:::awsexamplebucket1",
            "Condition" : {
                "StringNotEquals" : {
                    "s3:prefix": "projects"
                }
            }
        }
    ]
}

```

}

Test the policy with the AWS CLI

You can test the policy using the following `list-object` AWS CLI command. In the command, you provide user credentials using the `--profile` parameter. For more information about setting up and using the AWS CLI, see [Developing with Amazon S3 using the AWS CLI \(p. 1183\)](#).

```
aws s3api list-objects --bucket awsexamplebucket1 --prefix examplefolder --profile AccountADave
```

If the bucket is version-enabled, to list the objects in the bucket, you must grant the `s3>ListBucketVersions` permission in the preceding policy, instead of `s3>ListBucket` permission. This permission also supports the `s3:prefix` condition key.

Example 3: Setting the maximum number of keys

You can use the `s3:max-keys` condition key to set the maximum number of keys that requester can return in a [GET Bucket \(ListObjects\)](#) or [ListObjectVersions](#) request. By default, the API returns up to 1,000 keys. For a list of numeric condition operators that you can use with `s3:max-keys` and accompanying examples, see [Numeric Condition Operators](#) in the [IAM User Guide](#).

Actions, resources, and condition keys for Amazon S3

Amazon S3 (service prefix: `s3`) provides the following service-specific resources, actions, and condition context keys for use in IAM permission policies.

Note

You can use the actions listed below in IAM policies and Amazon S3 bucket policies to grant permissions for specific Amazon S3 API operations. Most actions have the same name as the API operations they are associated with. However, in some cases, the API operation and action names are different. Additionally, a single action can control access to more than one operation, and some operations require several different actions.

References:

- Learn how to [configure this service](#).
- View a list of the [API operations available for this service](#).
- Learn how to secure this service and its resources by [using IAM permission policies](#).

Topics

- [Actions defined by Amazon S3 \(p. 431\)](#)
- [Resource types defined by Amazon S3 \(p. 484\)](#)
- [Condition keys for Amazon S3 \(p. 484\)](#)

Actions defined by Amazon S3

You can specify the following actions in the `Action` element of an IAM policy statement. Use policies to grant permissions to perform an operation in AWS. When you use an action in a policy, you usually allow or deny access to the API operation or CLI command with the same name. However, in some cases, a single action controls access to more than one operation. Alternatively, some operations require several different actions.

The **Resource types** column indicates whether each action supports resource-level permissions. If there is no value for this column, you must specify all resources ("*") in the `Resource` element of your policy statement. If the column includes a resource type, then you can specify an ARN of that type in

a statement with that action. Required resources are indicated in the table with an asterisk (*). If you specify a resource-level permission ARN in a statement using this action, then it must be of this type. Some actions support multiple resource types. If the resource type is optional (not indicated as required), then you can choose to use one but not the other.

Actions	Description	Access level	Resource types (*required)	Condition keys	Dependent actions
AbortMultipartUpload	Grants permission to abort a multipart upload	Write	object* (p. 484)		s3:DataAccessPointArn (p. 485) s3:DataAccessPointAccount (p. 485) s3:AccessPointNetworkOrigin (p. 485) s3:authType (p. 485) s3:ResourceAccount (p. 485) s3:signatureAge (p. 486) s3:signatureversion (p. 486) s3:TlsVersion (p. 485) s3:x-amz-content-sha256 (p. 486)
BypassGovernanceRetention	Grants permission to allow intervention of governance-mode object retention settings	Permissions management	object* (p. 484)		s3:DataAccessPointAccount (p. 485) s3:DataAccessPointArn (p. 485) s3:AccessPointNetworkOrigin (p. 485) s3:RequestObjectTag/<key> (p. 485) s3:RequestObjectTagKeys (p. 485) s3:authType (p. 485) s3:ResourceAccount (p. 485) s3:signatureAge (p. 486) s3:signatureversion (p. 486) s3:TlsVersion (p. 485) s3:x-amz-acl (p. 486) s3:x-amz-content-sha256 (p. 486)

Actions	Description	Access level	Resource types (*required)	Condition keys	Dependent actions
				s3:x-amz-copy-source (p. 486) s3:x-amz-grant-full-control (p. 486) s3:x-amz-grant-read (p. 486) s3:x-amz-grant-read-acp (p. 486) s3:x-amz-grant-write (p. 486) s3:x-amz-grant-write-acp (p. 486) s3:x-amz-metadata-directive (p. 486) s3:x-amz-server-side-encryption (p. 486) s3:x-amz-server-side-encryption-aws-kms-key-id (p. 487) s3:x-amz-server-side-encryption-customer-algorithm (p. 487) s3:x-amz-storage-class (p. 487)	

Actions	Description	Access level	Resource types (*required)	Condition keys	Dependent actions
				s3:x-amz-website-redirect-location (p. 487) s3:object-lock-mode (p. 486) s3:object-lock-retain-until-date (p. 486) s3:object-lock-remaining-retention-days (p. 486) s3:object-lock-legal-hold (p. 486)	
CreateAccessPoint	Grants permission to create a new access point	Write	accesspoint* (p. 484) s3:DataAccessPointAccount (p. 485) s3:DataAccessPointArn (p. 485) s3:AccessPointNetworkOrigin (p. 485) s3:authType (p. 485) s3:locationconstraint (p. 486) s3:ResourceAccount (p. 485) s3:signatureAge (p. 486) s3:signatureversion (p. 486) s3:TlsVersion (p. 485) s3:x-amz-acl (p. 486) s3:x-amz-content-sha256 (p. 486)		
CreateAccessPointForObjectLambda		Write	objectlambdaaccesspoint* (p. 484)		

Actions	Description	Access level	Resource types (*required)	Condition keys	Dependent actions
	Grants permission to create an object lambda enabled accesspoint			s3:DataAccessPointAccount (p. 485) s3:DataAccessPointArn (p. 485) s3:AccessPointNetworkOrigin (p. 485) s3:authType (p. 485) s3:ResourceAccount (p. 485) s3:signatureAge (p. 486) s3:signatureversion (p. 486) s3:TlsVersion (p. 485) s3:x-amz-content-sha256 (p. 486)	
CreateBucket	Grants permission to create a new bucket	Write	bucket* (p. 484)		

Actions	Description	Access level	Resource types (*required)	Condition keys	Dependent actions
				s3:authType (p. 485) s3:locationconstraint (p. 486) s3:ResourceAccount (p. 485) s3:signatureAge (p. 486) s3:signatureversion (p. 486) s3:TlsVersion (p. 485) s3:x-amz-acl (p. 486) s3:x-amz-content-sha256 (p. 486) s3:x-amz-grant-full-control (p. 486) s3:x-amz-grant-read (p. 486) s3:x-amz-grant-read-acp (p. 486) s3:x-amz-grant-write (p. 486) s3:x-amz-grant-write-acp (p. 486)	

Actions	Description	Access level	Resource types (*required)	Condition keys	Dependent actions
CreateJob	Grants permission to create a new Amazon S3 Batch Operations job	Write		s3:authType (iam:PassRole) s3:ResourceAccount (p. 485) s3:signatureAge (p. 486) s3:signatureversion (p. 486) s3:TlsVersion (p. 485) s3:x-amz-content-sha256 (p. 486) s3:RequestJobPriority (p. 485) s3:RequestJobOperation (p. 485) aws:TagKeys (p. 485) aws:RequestTag/ \${TagKey} (p. 485)	
CreateMultipartUpload	Initiates a multipart upload and returns an upload ID	Write	object* (p. 484)		
				s3:DataAccessPointArn (p. 485) s3:DataAccessPointAccount (p. 485) s3:AccessPointNetworkOrigin (p. 485) s3:authType (p. 485) s3:ResourceAccount (p. 485) s3:signatureAge (p. 486) s3:signatureversion (p. 486) s3:TlsVersion (p. 485) s3:x-amz-content-sha256 (p. 486)	
DeleteAccessPoint	Grants permission to delete the access point named in the URI	Write	accesspoint* (p. 484)		

Actions	Description	Access level	Resource types (*required)	Condition keys	Dependent actions
					s3:DataAccessPointArn (p. 485) s3:DataAccessPointAccount (p. 485) s3:AccessPointNetworkOrigin (p. 485) s3:authType (p. 485) s3:ResourceAccount (p. 485) s3:signatureAge (p. 486) s3:signatureversion (p. 486) s3:TlsVersion (p. 485) s3:x-amz-content-sha256 (p. 486)
DeleteAccessPointObjectLambdaPolicy	Grants permission to delete the objectlambdaenabled access point named in the URI	Write	objectlambdaaccesspoint* (p. 484)		s3:DataAccessPointArn (p. 485) s3:DataAccessPointAccount (p. 485) s3:AccessPointNetworkOrigin (p. 485) s3:authType (p. 485) s3:ResourceAccount (p. 485) s3:signatureAge (p. 486) s3:signatureversion (p. 486) s3:TlsVersion (p. 485) s3:x-amz-content-sha256 (p. 486)
DeleteAccessPointPolicy	Grants permission to delete the policy on a specified access point	Permissions management	accesspoint* (p. 484)		

Actions	Description	Access level	Resource types (*required)	Condition keys	Dependent actions
					s3:DataAccessPointArn (p. 485) s3:DataAccessPointAccount (p. 485) s3:AccessPointNetworkOrigin (p. 485) s3:authType (p. 485) s3:ResourceAccount (p. 485) s3:signatureAge (p. 486) s3:signatureversion (p. 486) s3:TlsVersion (p. 485) s3:x-amz-content-sha256 (p. 486)
DeleteAccessPoint	Grants permission to delete the policy object specified object lambda enabled access point	Permissions management	objectlambdaaccesspoint* (p. 484)		s3:DataAccessPointArn (p. 485) s3:DataAccessPointAccount (p. 485) s3:AccessPointNetworkOrigin (p. 485) s3:authType (p. 485) s3:ResourceAccount (p. 485) s3:signatureAge (p. 486) s3:signatureversion (p. 486) s3:TlsVersion (p. 485) s3:x-amz-content-sha256 (p. 486)
DeleteBucket	Grants permission to delete the bucket named in the URI	Write	bucket* (p. 484)		

Actions	Description	Access level	Resource types (*required)	Condition keys	Dependent actions
DeleteBucketOwnershipControls	Grants permission to delete ownership controls on a bucket	Write	bucket* (p. 484)		
			Permissions management	s3:authType (p. 485) s3:ResourceAccount (p. 485) s3:signatureAge (p. 486) s3:signatureversion (p. 486) s3:TlsVersion (p. 485) s3:x-amz-content-sha256 (p. 486)	
DeleteBucketPolicy	Grants permission to delete the policy on a specified bucket	Write	bucket* (p. 484)	s3:authType (p. 485) s3:ResourceAccount (p. 485) s3:signatureAge (p. 486) s3:signatureversion (p. 486) s3:TlsVersion (p. 485) s3:x-amz-content-sha256 (p. 486)	
DeleteJobTagging		Tagging	job* (p. 484)		

Actions	Description	Access level	Resource types (*required)	Condition keys	Dependent actions
	Grants permission to remove tags from an existing Amazon S3 Batch Operations job			s3:authType (p. 485) s3:ResourceAccount (p. 485) s3:signatureAge (p. 486) s3:signatureversion (p. 486) s3:TlsVersion (p. 485) s3:x-amz-content-sha256 (p. 486) s3:ExistingJobPriority (p. 485) s3:ExistingJobOperation (p. 485)	
DeleteObject	Grants permission to remove the null version of an object and insert a delete marker, which becomes the current version of the object	Write	object* (p. 484)	s3:DataAccessPointAccount (p. 485) s3:DataAccessPointArn (p. 485) s3:AccessPointNetworkOrigin (p. 485) s3:authType (p. 485) s3:ResourceAccount (p. 485) s3:signatureAge (p. 486) s3:signatureversion (p. 486) s3:TlsVersion (p. 485) s3:x-amz-content-sha256 (p. 486)	

Actions	Description	Access level	Resource types (*required)	Condition keys	Dependent actions
DeleteObjectTagging	Grants permission to use the <code>Tagging</code> subresource to remove the entire tag set from the specified object	Tagging	object* (p. 484)		
DeleteObjectVersionTagging	Grants permission to remove a specific version of an object	Write	object* (p. 484)		
		Tagging	object* (p. 484)		

Actions	Description	Access level	Resource types (*required)	Condition keys	Dependent actions
	Grants permission to remove the entire tag set for a specific version of the object				s3:DataAccessPointAccount (p. 485) s3:DataAccessPointArn (p. 485) s3:AccessPointNetworkOrigin (p. 485) s3:ExistingObjectTag/<key> (p. 485) s3:authType (p. 485) s3:ResourceAccount (p. 485) s3:signatureAge (p. 486) s3:signatureversion (p. 486) s3:TlsVersion (p. 485) s3:versionid (p. 486) s3:x-amz-content-sha256 (p. 486)
DeleteStorageLensConfiguration	Grants permission to delete an existing Amazon S3 Storage Lens configuration	Write	storageLensConfiguration*	(p. 484)	
				s3:authType (p. 485) s3:ResourceAccount (p. 485) s3:signatureAge (p. 486) s3:signatureversion (p. 486) s3:TlsVersion (p. 485) s3:x-amz-content-sha256 (p. 486)	
DeleteStorageLensConfigurationTags	Grants permission to remove tags from an existing Amazon S3 Storage Lens configuration	Tagging	storageLensConfiguration*	(p. 484)	
				s3:authType (p. 485) s3:ResourceAccount (p. 485) s3:signatureAge (p. 486) s3:signatureversion (p. 486) s3:TlsVersion (p. 485) s3:x-amz-content-sha256 (p. 486)	

Actions	Description	Access level	Resource types (*required)	Condition keys	Dependent actions
DescribeJob	Grants permission to retrieve the configuration parameters and status for a batch operations job	Read	job* (p. 484)	s3:authType (p. 485) s3:ResourceAccount (p. 485) s3:signatureAge (p. 486) s3:signatureversion (p. 486) s3:TlsVersion (p. 485) s3:x-amz-content-sha256 (p. 486)	
GetAccelerateConfiguration	Grants permission to uses the accelerate subresource to return the Transfer Acceleration state of a bucket, which is either Enabled or Suspended	Read	bucket* (p. 484)	s3:authType (p. 485) s3:ResourceAccount (p. 485) s3:signatureAge (p. 486) s3:signatureversion (p. 486) s3:TlsVersion (p. 485) s3:x-amz-content-sha256 (p. 486)	
GetAccessPoint	Grants permission to return configuration information about the specified access point	Read		s3:DataAccessPointAccount (p. 485) s3:DataAccessPointArn (p. 485) s3:AccessPointNetworkOrigin (p. 485) s3:authType (p. 485) s3:ResourceAccount (p. 485) s3:signatureAge (p. 486) s3:signatureversion (p. 486) s3:TlsVersion (p. 485) s3:x-amz-content-sha256 (p. 486)	
GetAccessPointConfigurationForObjectLambda		Read	objectlambdaaccesspoint* (p. 484)		

Actions	Description	Access level	Resource types (*required)	Condition keys	Dependent actions
	Grants permission to retrieve the configuration of the object lambda enabled access point				s3:DataAccessPointArn (p. 485) s3:DataAccessPointAccount (p. 485) s3:AccessPointNetworkOrigin (p. 485) s3:authType (p. 485) s3:ResourceAccount (p. 485) s3:signatureAge (p. 486) s3:signatureversion (p. 486) s3:TlsVersion (p. 485) s3:x-amz-content-sha256 (p. 486)
GetAccessPointForObjectLambda	Grants permission to create an object lambda enabled accesspoint	Read	objectlambdaaccesspoint* (p. 484)		s3:DataAccessPointAccount (p. 485) s3:DataAccessPointArn (p. 485) s3:AccessPointNetworkOrigin (p. 485) s3:authType (p. 485) s3:ResourceAccount (p. 485) s3:signatureAge (p. 486) s3:signatureversion (p. 486) s3:TlsVersion (p. 485) s3:x-amz-content-sha256 (p. 486)
GetAccessPointPolicy		Read	accesspoint* (p. 484)		

Actions	Description	Access level	Resource types (*required)	Condition keys	Dependent actions
	Grants permission to returns the access point policy associated with the specified access point				s3:DataAccessPointAccount (p. 485) s3:DataAccessPointArn (p. 485) s3:AccessPointNetworkOrigin (p. 485) s3:authType (p. 485) s3:ResourceAccount (p. 485) s3:signatureAge (p. 486) s3:signatureversion (p. 486) s3:TlsVersion (p. 485) s3:x-amz-content-sha256 (p. 486)
GetAccessPointPolicy	Grants permission to returns the access point policy associated with the specified object lambda enabled access point	Read	objectlambdaaccesspoint* (p. 484)		s3:DataAccessPointAccount (p. 485) s3:DataAccessPointArn (p. 485) s3:AccessPointNetworkOrigin (p. 485) s3:authType (p. 485) s3:ResourceAccount (p. 485) s3:signatureAge (p. 486) s3:signatureversion (p. 486) s3:TlsVersion (p. 485) s3:x-amz-content-sha256 (p. 486)
GetAccessPointPolicyStatus		Read	accesspoint* (p. 484)		

Actions	Description	Access level	Resource types (*required)	Condition keys	Dependent actions
	Grants permission to return the policy status for a specific access point policy				s3:DataAccessPointAccount (p. 485) s3:DataAccessPointArn (p. 485) s3:AccessPointNetworkOrigin (p. 485) s3:authType (p. 485) s3:ResourceAccount (p. 485) s3:signatureAge (p. 486) s3:signatureversion (p. 486) s3:TlsVersion (p. 485) s3:x-amz-content-sha256 (p. 486)
GetAccessPointPolicyStatusForObjectLambda	Grants permission to return the policy status for a specific object lambda access point policy	Read	objectlambdaaccesspoint* (p. 484)		s3:DataAccessPointAccount (p. 485) s3:DataAccessPointArn (p. 485) s3:AccessPointNetworkOrigin (p. 485) s3:authType (p. 485) s3:ResourceAccount (p. 485) s3:signatureAge (p. 486) s3:signatureversion (p. 486) s3:TlsVersion (p. 485) s3:x-amz-content-sha256 (p. 486)
GetAccountPublicAccessBlock	Grants permission to retrieve the PublicAccessBlock configuration for an AWS account	Read			s3:authType (p. 485) s3:ResourceAccount (p. 485) s3:signatureAge (p. 486) s3:signatureversion (p. 486) s3:TlsVersion (p. 485) s3:x-amz-content-sha256 (p. 486)

Actions	Description	Access level	Resource types (*required)	Condition keys	Dependent actions
GetAnalyticsConfiguration	Grants permission to get an analytics configuration from an Amazon S3 bucket, identified by the analytics configuration ID	Read	bucket* (p. 484) 	s3:authType (p. 485) s3:ResourceAccount (p. 485) s3:signatureAge (p. 486) s3:signatureversion (p. 486) s3:TlsVersion (p. 485) s3:x-amz-content-sha256 (p. 486)	
GetBucketAcl	Grants permission to use the acl subresource to return the access control list (ACL) of an Amazon S3 bucket	Read	bucket* (p. 484) 	s3:authType (p. 485) s3:ResourceAccount (p. 485) s3:signatureAge (p. 486) s3:signatureversion (p. 486) s3:TlsVersion (p. 485) s3:x-amz-content-sha256 (p. 486)	
GetBucketCORS	Grants permission to return the CORS configuration information set for an Amazon S3 bucket	Read	bucket* (p. 484) 	s3:authType (p. 485) s3:ResourceAccount (p. 485) s3:signatureAge (p. 486) s3:signatureversion (p. 486) s3:TlsVersion (p. 485) s3:x-amz-content-sha256 (p. 486)	
GetBucketLocation	Grants permission to return the Region that an Amazon S3 bucket resides in	Read	bucket* (p. 484)		

Actions	Description	Access level	Resource types (*required)	Condition keys	Dependent actions
GetBucketLogging	Grants permission to return the logging status of an Amazon S3 bucket and the permissions users have to view or modify that status	Read	bucket* (p. 484) 	s3:authType (p. 485) 	
GetBucketNotification	Grants permission to get the notification configuration of an Amazon S3 bucket	Read	bucket* (p. 484) 	s3:authType (p. 485) 	
GetBucketObjectLock	Grants permission to get the Object Lock configuration of an Amazon S3 bucket	Read	bucket* (p. 484) 	s3:authType (p. 485) 	
GetBucketOwnershipControls	Grants permission to retrieve ownership controls on a bucket	Read	bucket* (p. 484)		

Actions	Description	Access level	Resource types (*required)	Condition keys	Dependent actions
				s3:authType (p. 485) s3:ResourceAccount (p. 485) s3:signatureAge (p. 486) s3:signatureversion (p. 486) s3:TlsVersion (p. 485) s3:x-amz-content-sha256 (p. 486)	
GetBucketPolicy	Grants permission to return the policy of the specified bucket	Read	bucket* (p. 484)		
				s3:authType (p. 485) s3:ResourceAccount (p. 485) s3:signatureAge (p. 486) s3:signatureversion (p. 486) s3:TlsVersion (p. 485) s3:x-amz-content-sha256 (p. 486)	
GetBucketPolicyStatus	Grants permission to retrieve the policy status for a specific Amazon S3 bucket, which indicates whether the bucket is public	Read	bucket* (p. 484)		
				s3:authType (p. 485) s3:ResourceAccount (p. 485) s3:signatureAge (p. 486) s3:signatureversion (p. 486) s3:TlsVersion (p. 485) s3:x-amz-content-sha256 (p. 486)	
GetBucketPublicAccessBlock		Read	bucket* (p. 484)		

Actions	Description	Access level	Resource types (*required)	Condition keys	Dependent actions
	Grants permission to retrieve the PublicAccessBlock configuration for an Amazon S3 bucket			s3:authType (p. 485) s3:ResourceAccount (p. 485) s3:signatureAge (p. 486) s3:signatureversion (p. 486) s3:TlsVersion (p. 485) s3:x-amz-content-sha256 (p. 486)	
GetBucketRequestPayment	Grants permission to return the request payment configuration for an Amazon S3 bucket	Read	bucket* (p. 484)	s3:authType (p. 485) s3:ResourceAccount (p. 485) s3:signatureAge (p. 486) s3:signatureversion (p. 486) s3:TlsVersion (p. 485) s3:x-amz-content-sha256 (p. 486)	
GetBucketTagging	Grants permission to return the tag set associated with an Amazon S3 bucket	Read	bucket* (p. 484)	s3:authType (p. 485) s3:ResourceAccount (p. 485) s3:signatureAge (p. 486) s3:signatureversion (p. 486) s3:TlsVersion (p. 485) s3:x-amz-content-sha256 (p. 486)	
GetBucketVersioning		Read	bucket* (p. 484)		

Actions	Description	Access level	Resource types (*required)	Condition keys	Dependent actions
	Grants permission to return the versioning state of an Amazon S3 bucket			s3:authType (p. 485) s3:ResourceAccount (p. 485) s3:signatureAge (p. 486) s3:signatureversion (p. 486) s3:TlsVersion (p. 485) s3:x-amz-content-sha256 (p. 486)	
GetBucketWebsite	Grants permission to return the website configuration for an Amazon S3 bucket	Read	bucket* (p. 484)		
				s3:authType (p. 485) s3:ResourceAccount (p. 485) s3:signatureAge (p. 486) s3:signatureversion (p. 486) s3:TlsVersion (p. 485) s3:x-amz-content-sha256 (p. 486)	
GetEncryptionConfiguration	Grants permission to return the encryption configuration for an Amazon S3 bucket	Read	bucket* (p. 484)		
				s3:authType (p. 485) s3:ResourceAccount (p. 485) s3:signatureAge (p. 486) s3:signatureversion (p. 486) s3:TlsVersion (p. 485) s3:x-amz-content-sha256 (p. 486)	

Actions	Description	Access level	Resource types (*required)	Condition keys	Dependent actions
GetIntelligentTierList	Grants permission to get an or list all Amazon S3 Intelligent Tiering configuration in a S3 Bucket	Read	bucket* (p. 484) 	s3:authType (p. 485) s3:ResourceAccount (p. 485) s3:signatureAge (p. 486) s3:signatureversion (p. 486) s3:TlsVersion (p. 485) s3:x-amz-content-sha256 (p. 486)	
GetInventoryConfiguration	Grants permission to return an inventory configuration from an Amazon S3 bucket, identified by the inventory configuration ID	Read	bucket* (p. 484) 	s3:authType (p. 485) s3:ResourceAccount (p. 485) s3:signatureAge (p. 486) s3:signatureversion (p. 486) s3:TlsVersion (p. 485) s3:x-amz-content-sha256 (p. 486)	
GetJobTagging	Grants permission to return the tag set of an existing Amazon S3 Batch Operations job	Read	job* (p. 484) 	s3:authType (p. 485) s3:ResourceAccount (p. 485) s3:signatureAge (p. 486) s3:signatureversion (p. 486) s3:TlsVersion (p. 485) s3:x-amz-content-sha256 (p. 486)	

Actions	Description	Access level	Resource types (*required)	Condition keys	Dependent actions
GetLifecycleConfiguration	Grants permission to return the lifecycle configuration information set on an Amazon S3 bucket	Read	bucket* (p. 484) 	s3:authType (p. 485) s3:ResourceAccount (p. 485) s3:signatureAge (p. 486) s3:signatureversion (p. 486) s3:TlsVersion (p. 485) s3:x-amz-content-sha256 (p. 486)	
GetMetricsConfiguration	Grants permission to get a metrics configuration from an Amazon S3 bucket	Read	bucket* (p. 484) 	s3:authType (p. 485) s3:ResourceAccount (p. 485) s3:signatureAge (p. 486) s3:signatureversion (p. 486) s3:TlsVersion (p. 485) s3:x-amz-content-sha256 (p. 486)	
GetObject	Grants permission to retrieve objects from Amazon S3	Read	object* (p. 484) 	s3:DataAccessPointAccount (p. 485) s3:DataAccessPointArn (p. 485) s3:AccessPointNetworkOrigin (p. 485) s3:ExistingObjectTag/<key> (p. 485) s3:authType (p. 485) s3:ResourceAccount (p. 485) s3:signatureAge (p. 486) s3:signatureversion (p. 486) s3:TlsVersion (p. 485) s3:x-amz-content-sha256 (p. 486)	

Actions	Description	Access level	Resource types (*required)	Condition keys	Dependent actions
GetObjectAcl	Grants permission to return the access control list (ACL) of an object	Read	object* (p. 484)		s3:DataAccessPointAccount (p. 485) s3:DataAccessPointArn (p. 485) s3:AccessPointNetworkOrigin (p. 485) s3:ExistingObjectTag/<key> (p. 485) s3:authType (p. 485) s3:ResourceAccount (p. 485) s3:signatureAge (p. 486) s3:signatureversion (p. 486) s3:TlsVersion (p. 485) s3:x-amz-content-sha256 (p. 486)
GetObjectLegalHold	Grants permission to get an object's current Legal Hold status	Read	object* (p. 484)		s3:DataAccessPointAccount (p. 485) s3:DataAccessPointArn (p. 485) s3:AccessPointNetworkOrigin (p. 485) s3:authType (p. 485) s3:ResourceAccount (p. 485) s3:signatureAge (p. 486) s3:signatureversion (p. 486) s3:TlsVersion (p. 485) s3:x-amz-content-sha256 (p. 486)
GetObjectRetention	Grants permission to retrieve the retention settings for an object	Read	object* (p. 484)		

Actions	Description	Access level	Resource types (*required)	Condition keys	Dependent actions
					s3:DataAccessPointAccount (p. 485) s3:DataAccessPointArn (p. 485) s3:AccessPointNetworkOrigin (p. 485) s3:authType (p. 485) s3:ResourceAccount (p. 485) s3:signatureAge (p. 486) s3:signatureversion (p. 486) s3:TlsVersion (p. 485) s3:x-amz-content-sha256 (p. 486)
GetObjectTagging	Grants permission to return the tag set of an object	Read	object* (p. 484)		s3:DataAccessPointAccount (p. 485) s3:DataAccessPointArn (p. 485) s3:AccessPointNetworkOrigin (p. 485) s3:ExistingObjectTag/<key> (p. 485) s3:authType (p. 485) s3:ResourceAccount (p. 485) s3:signatureAge (p. 486) s3:signatureversion (p. 486) s3:TlsVersion (p. 485) s3:x-amz-content-sha256 (p. 486)
	s3:authType (p. 485) s3:ResourceAccount (p. 485) s3:signatureAge (p. 486) s3:signatureversion (p. 486) s3:TlsVersion (p. 485) s3:x-amz-content-sha256 (p. 486)				

Actions	Description	Access level	Resource types (*required)	Condition keys	Dependent actions
GetObjectVersion	Grants permission to retrieve a specific version of an object	Read	object* (p. 484)		s3:DataAccessPointAccount (p. 485) s3:DataAccessPointArn (p. 485) s3:AccessPointNetworkOrigin (p. 485) s3:ExistingObjectTag/<key> (p. 485) s3:authType (p. 485) s3:ResourceAccount (p. 485) s3:signatureAge (p. 486) s3:signatureversion (p. 486) s3:TlsVersion (p. 485) s3:versionid (p. 486) s3:x-amz-content-sha256 (p. 486)
GetObjectVersion	Grants permission to return the access control list (ACL) of a specific object version	Read	object* (p. 484)		s3:DataAccessPointAccount (p. 485) s3:DataAccessPointArn (p. 485) s3:AccessPointNetworkOrigin (p. 485) s3:ExistingObjectTag/<key> (p. 485) s3:authType (p. 485) s3:ResourceAccount (p. 485) s3:signatureAge (p. 486) s3:signatureversion (p. 486) s3:TlsVersion (p. 485) s3:versionid (p. 486) s3:x-amz-content-sha256 (p. 486)

Actions	Description	Access level	Resource types (*required)	Condition keys	Dependent actions
GetObjectVersion	Grants permission to replicate both unencrypted objects and objects encrypted with SSE-S3 or SSE-KMS	Read	object* (p. 484)		
GetObjectVersion	Grants permission to return the TagSet for a specific version of the object	Read	object* (p. 484)		

Actions	Description	Access level	Resource types (*required)	Condition keys	Dependent actions
GetReplicationConfiguration	Grants permission to get the replication configuration information set on an Amazon S3 bucket	Read	bucket* (p. 484) 	s3:authType (p. 485) s3:ResourceAccount (p. 485) s3:signatureAge (p. 486) s3:signatureversion (p. 486) s3:TlsVersion (p. 485) s3:x-amz-content-sha256 (p. 486)	
GetStorageLensConfiguration	Grants permission to get an Amazon S3 Storage Lens configuration	Read	storagelensconfiguration* (p. 484) 	s3:authType (p. 485) s3:ResourceAccount (p. 485) s3:signatureAge (p. 486) s3:signatureversion (p. 486) s3:TlsVersion (p. 485) s3:x-amz-content-sha256 (p. 486)	
GetStorageLensConfigurationIfExists	Grants permission to get the tag configuration for an existing Amazon S3 Storage Lens configuration	Read	storagelensconfiguration* (p. 484) 	s3:authType (p. 485) s3:ResourceAccount (p. 485) s3:signatureAge (p. 486) s3:signatureversion (p. 486) s3:TlsVersion (p. 485) s3:x-amz-content-sha256 (p. 486)	
GetStorageLensDashboard		Read	storagelensconfiguration* (p. 484)		

Actions	Description	Access level	Resource types (*required)	Condition keys	Dependent actions
	Grants permission to get an Amazon S3 Storage Lens dashboard			s3:authType (p. 485) s3:ResourceAccount (p. 485) s3:signatureAge (p. 486) s3:signatureversion (p. 486) s3:TlsVersion (p. 485) s3:x-amz-content-sha256 (p. 486)	
ListAccessPoints	Grants permission to list access points	Read		s3:authType (p. 485) s3:ResourceAccount (p. 485) s3:signatureAge (p. 486) s3:signatureversion (p. 486) s3:TlsVersion (p. 485) s3:x-amz-content-sha256 (p. 486)	
ListAccessPointsForObjectEnabledAccesspoints	Grants permission to list object for object enabled accesspoints accesspoints	Read		s3:authType (p. 485) s3:ResourceAccount (p. 485) s3:signatureAge (p. 486) s3:signatureversion (p. 486) s3:TlsVersion (p. 485) s3:x-amz-content-sha256 (p. 486)	
ListAllMyBuckets	Grants permission to list all buckets owned by the authenticated sender of the request	List		s3:authType (p. 485) s3:ResourceAccount (p. 485) s3:signatureAge (p. 486) s3:signatureversion (p. 486) s3:TlsVersion (p. 485) s3:x-amz-content-sha256 (p. 486)	

Actions	Description	Access level	Resource types (*required)	Condition keys	Dependent actions
ListBucket	Grants permission to list some or all of the objects in an Amazon S3 bucket (up to 1000)	List	bucket* (p. 484)		s3:DataAccessPointAccount (p. 485) s3:DataAccessPointArn (p. 485) s3:AccessPointNetworkOrigin (p. 485) s3:authType (p. 485) s3:delimiter (p. 485) s3:max-keys (p. 486) s3:prefix (p. 486) s3:ResourceAccount (p. 485) s3:signatureAge (p. 486) s3:signatureversion (p. 486) s3:TlsVersion (p. 485) s3:x-amz-content-sha256 (p. 486)
ListBucketMultipartUploads	Grants permission to list incomplete multipart uploads	List	bucket* (p. 484)		s3:DataAccessPointAccount (p. 485) s3:DataAccessPointArn (p. 485) s3:AccessPointNetworkOrigin (p. 485) s3:authType (p. 485) s3:ResourceAccount (p. 485) s3:signatureAge (p. 486) s3:signatureversion (p. 486) s3:TlsVersion (p. 485) s3:x-amz-content-sha256 (p. 486)

Actions	Description	Access level	Resource types (*required)	Condition keys	Dependent actions
ListBucketVersions	Grants permission to list metadata about all the versions of objects in an Amazon S3 bucket	List	bucket* (p. 484)		s3:DataAccessPointAccount (p. 485) s3:DataAccessPointArn (p. 485) s3:AccessPointNetworkOrigin (p. 485) s3:authType (p. 485) s3:delimiter (p. 485) s3:max-keys (p. 486) s3:prefix (p. 486) s3:ResourceAccount (p. 485) s3:signatureAge (p. 486) s3:signatureversion (p. 486) s3:TlsVersion (p. 485) s3:x-amz-content-sha256 (p. 486)
ListJobs	Grants permission to list current jobs and jobs that have ended recently	List			s3:authType (p. 485) s3:ResourceAccount (p. 485) s3:signatureAge (p. 486) s3:signatureversion (p. 486) s3:TlsVersion (p. 485) s3:x-amz-content-sha256 (p. 486)
ListMultipartUploadParts		List	object* (p. 484)		

Actions	Description	Access level	Resource types (*required)	Condition keys	Dependent actions
	Grants permission to list the parts that have been uploaded for a specific multipart upload				s3:DataAccessPointAccount (p. 485) s3:DataAccessPointArn (p. 485) s3:AccessPointNetworkOrigin (p. 485) s3:authType (p. 485) s3:ResourceAccount (p. 485) s3:signatureAge (p. 486) s3:signatureversion (p. 486) s3:TlsVersion (p. 485) s3:x-amz-content-sha256 (p. 486)
ListStorageLensConfigurations	Grants permission to list Amazon S3 Storage Lens configurations	List			s3:authType (p. 485) s3:ResourceAccount (p. 485) s3:signatureAge (p. 486) s3:signatureversion (p. 486) s3:TlsVersion (p. 485) s3:x-amz-content-sha256 (p. 486)
ObjectOwnerOverrideReplicaOwnership	Grants permission to change object ownership	Permissions management	object* (p. 484) s3:authType (p. 485) s3:ResourceAccount (p. 485) s3:signatureAge (p. 486) s3:signatureversion (p. 486) s3:TlsVersion (p. 485) s3:x-amz-content-sha256 (p. 486)		

Actions	Description	Access level	Resource types (*required)	Condition keys	Dependent actions
PutAccelerateConfiguration	Grants permission to use the <code>accelerate</code> subresource to set the Transfer Acceleration state of an existing S3 bucket	Write	bucket* (p. 484)		
		Write	objectlambdaaccesspoint* (p. 484)		
PutAccessPointPolicy		Permissions management	accesspoint* (p. 484)		

Actions	Description	Access level	Resource types (*required)	Condition keys	Dependent actions
	Grants permission to associate an access policy with a specified access point			s3:DataAccessPointAccount (p. 485) s3:DataAccessPointArn (p. 485) s3:AccessPointNetworkOrigin (p. 485) s3:authType (p. 485) s3:ResourceAccount (p. 485) s3:signatureAge (p. 486) s3:signatureversion (p. 486) s3:TlsVersion (p. 485) s3:x-amz-content-sha256 (p. 486)	
PutAccessPointPolicy	Grants permission to associate any AccessPolicy with a specified object lambda enabled access point	Permissions management	objectlambdaaccesspoint* (p. 484)		s3:DataAccessPointAccount (p. 485) s3:DataAccessPointArn (p. 485) s3:AccessPointNetworkOrigin (p. 485) s3:authType (p. 485) s3:ResourceAccount (p. 485) s3:signatureAge (p. 486) s3:signatureversion (p. 486) s3:TlsVersion (p. 485) s3:x-amz-content-sha256 (p. 486)
PutAccountPublicAccessBlock	Grants permission to create or modify the PublicAccessBlock configuration for an AWS account	Permissions management			s3:authType (p. 485) s3:ResourceAccount (p. 485) s3:signatureAge (p. 486) s3:signatureversion (p. 486) s3:TlsVersion (p. 485) s3:x-amz-content-sha256 (p. 486)

Actions	Description	Access level	Resource types (*required)	Condition keys	Dependent actions
PutAnalyticsConfiguration	Grants permission to set an analytics configuration for the bucket, specified by the analytics configuration ID	Write	bucket* (p. 484)		
PutBucketAcl	Grants permission to set the permissions on an existing bucket using access control lists (ACLs)	Permissions management	bucket* (p. 484)	s3:authType (p. 485) s3:ResourceAccount (p. 485) s3:signatureAge (p. 486) s3:signatureversion (p. 486) s3:TlsVersion (p. 485) s3:x-amz-content-sha256 (p. 486)	

Actions	Description	Access level	Resource types (*required)	Condition keys	Dependent actions
PutBucketCORS	Grants permission to set the CORS configuration for an Amazon S3 bucket	Write	bucket* (p. 484)		
		Write	bucket* (p. 484)	s3:authType (p. 485) s3:ResourceAccount (p. 485) s3:signatureAge (p. 486) s3:signatureversion (p. 486) s3:TlsVersion (p. 485) s3:x-amz-content-sha256 (p. 486)	
PutBucketLogging	Grants permission to set the logging parameters for an Amazon S3 bucket	Write	bucket* (p. 484)	s3:authType (p. 485) s3:ResourceAccount (p. 485) s3:signatureAge (p. 486) s3:signatureversion (p. 486) s3:TlsVersion (p. 485) s3:x-amz-content-sha256 (p. 486)	
PutBucketNotification	Grants permission to receive notifications when certain events happen in an Amazon S3 bucket	Write	bucket* (p. 484)	s3:authType (p. 485) s3:ResourceAccount (p. 485) s3:signatureAge (p. 486) s3:signatureversion (p. 486) s3:TlsVersion (p. 485) s3:x-amz-content-sha256 (p. 486)	
	PutBucketObjectLockConfiguration	Write	bucket* (p. 484)		

Actions	Description	Access level	Resource types (*required)	Condition keys	Dependent actions
	Grants permission to put Object Lock configuration on a specific bucket			s3:authType (p. 485) s3:ResourceAccount (p. 485) s3:signatureAge (p. 486) s3:TlsVersion (p. 485) s3:signatureversion (p. 486)	
PutBucketOwnershipReplace	Grants permission to add or replace ownership controls on a bucket	Write	bucket* (p. 484)	s3:authType (p. 485) s3:ResourceAccount (p. 485) s3:signatureAge (p. 486) s3:signatureversion (p. 486) s3:TlsVersion (p. 485) s3:x-amz-content-sha256 (p. 486)	
PutBucketPolicy	Grants permission to add or replace a bucket policy on a bucket	Permissions management	bucket* (p. 484)	s3:authType (p. 485) s3:ResourceAccount (p. 485) s3:signatureAge (p. 486) s3:signatureversion (p. 486) s3:TlsVersion (p. 485) s3:x-amz-content-sha256 (p. 486)	
PutBucketPublicAccessBlock	Grants permission to create or modify the PublicAccessBlock configuration for a specific Amazon S3 bucket	Permissions management	bucket* (p. 484)	s3:authType (p. 485) s3:ResourceAccount (p. 485) s3:signatureAge (p. 486) s3:signatureversion (p. 486) s3:TlsVersion (p. 485) s3:x-amz-content-sha256 (p. 486)	

Actions	Description	Access level	Resource types (*required)	Condition keys	Dependent actions
PutBucketRequestPayment	Grants permission to set the payment configuration of a bucket	Write	bucket* (p. 484)		
		Tagging	bucket* (p. 484)		
PutBucketVersioning	Grants permission to set the versioning state of an existing Amazon S3 bucket	Write	bucket* (p. 484)		

Actions	Description	Access level	Resource types (*required)	Condition keys	Dependent actions
PutBucketWebsite	Grants permission to set the configuration of the website that is specified in the website subresource	Write	bucket* (p. 484)		
		Write	bucket* (p. 484)	s3:authType (p. 485) s3:ResourceAccount (p. 485) s3:signatureAge (p. 486) s3:signatureversion (p. 486) s3:TlsVersion (p. 485) s3:x-amz-content-sha256 (p. 486)	
PutEncryptionConfiguration	Grants permission to set the encryption configuration for an Amazon S3 bucket	Write	bucket* (p. 484)	s3:authType (p. 485) s3:ResourceAccount (p. 485) s3:signatureAge (p. 486) s3:signatureversion (p. 486) s3:TlsVersion (p. 485) s3:x-amz-content-sha256 (p. 486)	
		Write	bucket* (p. 484)		
		Write	s3:authType (p. 485) s3:ResourceAccount (p. 485) s3:signatureAge (p. 486) s3:signatureversion (p. 486) s3:TlsVersion (p. 485) s3:x-amz-content-sha256 (p. 486)		

Actions	Description	Access level	Resource types (*required)	Condition keys	Dependent actions
PutInventoryConfiguration	Grants permission to add an inventory configuration to the bucket, identified by the inventory ID	Write	bucket* (p. 484)		
PutJobTagging	Grants permission to replace tags on an existing Amazon S3 Batch Operations job	Tagging	job* (p. 484)		
PutLifecycleConfiguration	Grants permission to create a new lifecycle configuration for the bucket or replace an existing lifecycle configuration	Write	bucket* (p. 484)		

Actions	Description	Access level	Resource types (*required)	Condition keys	Dependent actions
PutMetricsConfig <small>update</small>	Grants permission to set or update a metrics configuration for the CloudWatch request metrics from an Amazon S3 bucket	Write	bucket* (p. 484) <small>s3:authType (p. 485)</small> <small>s3:ResourceAccount (p. 485)</small> <small>s3:signatureAge (p. 486)</small> <small>s3:signatureversion (p. 486)</small> <small>s3:TlsVersion (p. 485)</small> <small>s3:x-amz-content-sha256 (p. 486)</small>		
PutObject	Grants permission to add an object to a bucket	Write	object* (p. 484)		

Actions	Description	Access level	Resource types (*required)	Condition keys	Dependent actions
					s3:DataAccessPointAccount (p. 485) s3:DataAccessPointArn (p. 485) s3:AccessPointNetworkOrigin (p. 485) s3:RequestObjectTag/<key> (p. 485) s3:RequestObjectTagKeys (p. 485) s3:authType (p. 485) s3:ResourceAccount (p. 485) s3:signatureAge (p. 486) s3:signatureversion (p. 486) s3:TlsVersion (p. 485) s3:x-amz-acl (p. 486) s3:x-amz-content-sha256 (p. 486) s3:x-amz-copy-source (p. 486) s3:x-amz-grant-full-control (p. 486) s3:x-amz-grant-read (p. 486) s3:x-amz-grant-read-acp (p. 486) s3:x-amz-grant-write (p. 486) s3:x-amz-grant-write-acp (p. 486)

Actions	Description	Access level	Resource types (*required)	Condition keys	Dependent actions
				s3:x-amz-metadata-directive (p. 486) s3:x-amz-server-side-encryption (p. 486) s3:x-amz-server-side-encryption-aws-kms-key-id (p. 487) s3:x-amz-server-side-encryption-customer-algorithm (p. 487) s3:x-amz-storage-class (p. 487) s3:x-amz-website-redirect-location (p. 487) s3:object-lock-mode (p. 486) s3:object-lock-retain-until-date (p. 486) s3:object-lock-remaining-retention-days (p. 486) s3:object-lock-legal-hold (p. 486)	

Actions	Description	Access level	Resource types (*required)	Condition keys	Dependent actions
PutObjectAcl	Grants permission to set the access control list (ACL) permissions for new or existing objects in an S3 bucket.	Permissions management	object* (p. 484)	s3:DataAccessPointAccount (p. 485) s3:DataAccessPointArn (p. 485) s3:AccessPointNetworkOrigin (p. 485) s3:ExistingObjectTag/<key> (p. 485) s3:authType (p. 485) s3:ResourceAccount (p. 485) s3:signatureAge (p. 486) s3:signatureversion (p. 486) s3:TlsVersion (p. 485) s3:x-amz-acl (p. 486) s3:x-amz-content-sha256 (p. 486) s3:x-amz-grant-full-control (p. 486) s3:x-amz-grant-read (p. 486) s3:x-amz-grant-read-acp (p. 486) s3:x-amz-grant-write (p. 486) s3:x-amz-grant-write-acp (p. 486) s3:x-amz-storage-class (p. 487)	
PutObjectLegalHold		Write	object* (p. 484)		

Actions	Description	Access level	Resource types (*required)	Condition keys	Dependent actions
	Grants permission to apply a Legal Hold configuration to the specified object			s3:DataAccessPointAccount (p. 485) s3:DataAccessPointArn (p. 485) s3:AccessPointNetworkOrigin (p. 485) s3:authType (p. 485) s3:ResourceAccount (p. 485) s3:signatureAge (p. 486) s3:signatureversion (p. 486) s3:TlsVersion (p. 485) s3:x-amz-content-sha256 (p. 486) s3:object-lock-legal-hold (p. 486)	
PutObjectRetention		Write	object* (p. 484)		

Actions	Description	Access level	Resource types (*required)	Condition keys	Dependent actions
	Grants permission to place an Object Retention configuration on an object				s3:DataAccessPointAccount (p. 485) s3:DataAccessPointArn (p. 485) s3:AccessPointNetworkOrigin (p. 485) s3:authType (p. 485) s3:ResourceAccount (p. 485) s3:signatureAge (p. 486) s3:signatureversion (p. 486) s3:TlsVersion (p. 485) s3:x-amz-content-sha256 (p. 486) s3:object-lock-mode (p. 486) s3:object-lock-retain-until-date (p. 486) s3:object-lock-remaining-retention-days (p. 486)

Actions	Description	Access level	Resource types (*required)	Condition keys	Dependent actions
PutObjectTagging	Grants permission to set the supplied tag-set to an object that already exists in a bucket or when first uploading the object	Tagging	object* (p. 484)		s3:DataAccessPointAccount (p. 485) s3:DataAccessPointArn (p. 485) s3:AccessPointNetworkOrigin (p. 485) s3:ExistingObjectTag/<key> (p. 485) s3:RequestObjectTag/<key> (p. 485) s3:RequestObjectTagKeys (p. 485) s3:authType (p. 485) s3:ResourceAccount (p. 485) s3:signatureAge (p. 486) s3:signatureversion (p. 486) s3:TlsVersion (p. 485) s3:x-amz-content-sha256 (p. 486)

Actions	Description	Access level	Resource types (*required)	Condition keys	Dependent actions
PutObjectVersion	Grants permission to use the acl subresource to set the access control list (ACL) permissions for an object that already exists in a bucket	Permissions management	object* (p. 484)		s3:DataAccessPointAccount (p. 485) s3:DataAccessPointArn (p. 485) s3:AccessPointNetworkOrigin (p. 485) s3:ExistingObjectTag/<key> (p. 485) s3:authType (p. 485) s3:ResourceAccount (p. 485) s3:signatureAge (p. 486) s3:signatureversion (p. 486) s3:TlsVersion (p. 485) s3:versionid (p. 486) s3:x-amz-acl (p. 486) s3:x-amz-content-sha256 (p. 486) s3:x-amz-grant-full-control (p. 486) s3:x-amz-grant-read (p. 486) s3:x-amz-grant-read-acp (p. 486) s3:x-amz-grant-write (p. 486) s3:x-amz-grant-write-acp (p. 486) s3:x-amz-storage-class (p. 487)

Actions	Description	Access level	Resource types (*required)	Condition keys	Dependent actions
PutObjectVersionTagging	Grants permission to set the Supplied tag-set for a specific version of an object	Tagging	object* (p. 484)		s3:DataAccessPointAccount (p. 485) s3:DataAccessPointArn (p. 485) s3:AccessPointNetworkOrigin (p. 485) s3:ExistingObjectTag/<key> (p. 485) s3:RequestObjectTag/<key> (p. 485) s3:RequestObjectTagKeys (p. 485) s3:authType (p. 485) s3:ResourceAccount (p. 485) s3:signatureAge (p. 486) s3:signatureversion (p. 486) s3:TlsVersion (p. 485) s3:versionid (p. 486) s3:x-amz-content-sha256 (p. 486)
PutReplicationConfiguration	Grants permission to create a new replication configuration or replace an existing one	Write	bucket* (p. 484)	iam:PassRole	s3:authType (p. 485) s3:ResourceAccount (p. 485) s3:signatureAge (p. 486) s3:signatureversion (p. 486) s3:TlsVersion (p. 485) s3:x-amz-content-sha256 (p. 486)

Actions	Description	Access level	Resource types (*required)	Condition keys	Dependent actions
PutStorageLensConfiguration	Grants permission to create or update an Amazon S3 Storage Lens configuration	Write			s3:authType (p. 485) s3:ResourceAccount (p. 485) s3:signatureAge (p. 486) s3:signatureversion (p. 486) s3:TlsVersion (p. 485) s3:x-amz-content-sha256 (p. 486) aws:TagKeys (p. 485) aws:RequestTag/\${TagKey} (p. 485)
PutStorageLensConfigurationTags	Grants permission to put or replace tags on an existing Amazon S3 Storage Lens configuration	Tagging	storagelensconfiguration* (p. 484)		s3:authType (p. 485) s3:ResourceAccount (p. 485) s3:signatureAge (p. 486) s3:signatureversion (p. 486) s3:TlsVersion (p. 485) s3:x-amz-content-sha256 (p. 486) aws:TagKeys (p. 485) aws:RequestTag/\${TagKey} (p. 485)
ReplicateDelete	Grants permission to replicate delete markers to the destination bucket	Write	object* (p. 484)		s3:authType (p. 485) s3:ResourceAccount (p. 485) s3:signatureAge (p. 486) s3:signatureversion (p. 486) s3:TlsVersion (p. 485) s3:x-amz-content-sha256 (p. 486)

Actions	Description	Access level	Resource types (*required)	Condition keys	Dependent actions
ReplicateObject	Grants permission to replicate objects and object tags to the destination bucket	Write	object* (p. 484)	s3:authType (p. 485) s3:ResourceAccount (p. 485) s3:signatureAge (p. 486) s3:signatureversion (p. 486) s3:TlsVersion (p. 485) s3:x-amz-content-sha256 (p. 486) s3:x-amz-server-side-encryption (p. 486) s3:x-amz-server-side-encryption-aws-kms-key-id (p. 487) s3:x-amz-server-side-encryption-customer-algorithm (p. 487)	
ReplicateTags	Grants permission to replicate object tags to the destination bucket	Tagging	object* (p. 484)	s3:authType (p. 485) s3:ResourceAccount (p. 485) s3:signatureAge (p. 486) s3:signatureversion (p. 486) s3:TlsVersion (p. 485) s3:x-amz-content-sha256 (p. 486)	

Actions	Description	Access level	Resource types (*required)	Condition keys	Dependent actions
RestoreObject	Grants permission to restore an archived copy of an object back into Amazon S3	Write	object* (p. 484)		s3:DataAccessPointAccount (p. 485) s3:DataAccessPointArn (p. 485) s3:AccessPointNetworkOrigin (p. 485) s3:authType (p. 485) s3:ResourceAccount (p. 485) s3:signatureAge (p. 486) s3:signatureversion (p. 486) s3:TlsVersion (p. 485) s3:x-amz-content-sha256 (p. 486)
UpdateJobPriority	Grants permission to update the priority of an existing job	Write	job* (p. 484)		s3:authType (p. 485) s3:ResourceAccount (p. 485) s3:signatureAge (p. 486) s3:signatureversion (p. 486) s3:TlsVersion (p. 485) s3:x-amz-content-sha256 (p. 486) s3:RequestJobPriority (p. 485) s3:ExistingJobPriority (p. 485) s3:ExistingJobOperation (p. 485)
UpdateJobStatus	Grants permission to update the status for the specified job	Write	job* (p. 484)		

Actions	Description	Access level	Resource types (*required)	Condition keys	Dependent actions
				s3:authType (p. 485) s3:ResourceAccount (p. 485) s3:signatureAge (p. 486) s3:signatureversion (p. 486) s3:TlsVersion (p. 485) s3:x-amz-content-sha256 (p. 486) s3:ExistingJobPriority (p. 485) s3:ExistingJobOperation (p. 485) s3:JobSuspendedCause (p. 485)	

Resource types defined by Amazon S3

The following resource types are defined by this service and can be used in the `Resource` element of IAM permission policy statements. Each action in the [Actions table \(p. 431\)](#) identifies the resource types that can be specified with that action. A resource type can also define which condition keys you can include in a policy. These keys are displayed in the last column of the table.

Resource types	ARN	Condition keys
accesspoint	arn:\${Partition}:s3:\${Region}: \${Account}:accesspoint/\${AccessPointName}	
bucket	arn:\${Partition}:s3::::\${BucketName}	
object	arn:\${Partition}:s3::::\${BucketName}/ \${ObjectName}	
job	arn:\${Partition}:s3:\${Region}: \${Account}:job/\${JobId}	
storagelensconfig	arn:\${Partition}:s3:\${Region}: \${Account}:storage-lens/\${ConfigId}	aws:ResourceTag/\${TagKey} (p. 485)
objectlambdaaccesspoint	arn:\${Partition}:s3-object-lambda:\${Region}: \${Account}:accesspoint/\${AccessPointName}	

Condition keys for Amazon S3

Amazon S3 defines the following condition keys that can be used in the `Condition` element of an IAM policy. You can use these keys to further refine the conditions under which the policy statement applies.

To view the global condition keys that are available to all services, see [Available global condition keys](#).

Condition keys	Description	Type
aws:RequestTag/\${TagKey}	Filters actions based on the tags that are passed in the request	String
aws:ResourceTag/\${TagKey}	Filters actions based on the tags associated with the resource	String
aws:TagKeys	Filters actions based on the tag keys that are passed in the request	String
s3:AccessPointNetworkOrigin	Filters access by the network origin (Internet or VPC)	String
s3:DataAccessPointArn	Filters access by the AWS account ID that owns the access point	String
s3:DataAccessPointArn	Filters access by an access point Amazon Resource Name (ARN)	String
s3:ExistingJobOperation	Filters access to updating the job priority by operation	String
s3:ExistingJobPriority	Filters access to cancelling existing jobs by priority range	Numeric
s3:ExistingObjectTag/<key>	Filters access by existing object tag key and value	String
s3:JobSuspendedCause	Filters access to cancelling suspended jobs by a specific job suspended cause (for example, AWAITING_CONFIRMATION)	String
s3:LocationConstraint	Filters access by a specific Region	String
s3:RequestJobOperation	Filters access to creating jobs by operation	String
s3:RequestJobPriority	Filters access to creating new jobs by priority range	Numeric
s3:RequestObjectTagObjects	Filters access by the tag keys and values to be added to objects	String
s3:RequestObjectTagKeys	Filters access by the tag keys to be added to objects	String
s3:ResourceAccount	Filters access by the resource owner AWS account ID	String
s3:TlsVersion	Filters access by the TLS version used by the client	Numeric
s3:VersionId	Filters access by a specific object version	String
s3:authType	Filters access by authentication method	String
s3:delimiter	Filters access by delimiter parameter	String

Condition keys	Description	Type
s3:locationconstraint	Filters access by a specific Region	String
s3:max-keys	Filters access by maximum number of keys returned in a ListBucket request	Numeric
s3:object-lock-legal-hold	Filters access by object legal hold status	String
s3:object-lock-mode	Filters access by object retention mode (COMPLIANCE or GOVERNANCE)	String
s3:object-lock-remaining-retention-days	Filters access by remaining object retention days	String
s3:object-lock-retain-until-date	Filters access by object retain-until date	String
s3:prefix	Filters access by key name prefix	String
s3:signatureAge	Filters access by the age in milliseconds of the request signature	Numeric
s3:signatureversion	Filters access by the version of AWS Signature used on the request	String
s3:versionid	Filters access by a specific object version	String
s3:x-amz-acl	Filters access by canned ACL in the request's x-amz-acl header	String
s3:x-amz-content-sha256	Filters access to unsigned content in your bucket	String
s3:x-amz-copy-source	Filters access to requests with a specific bucket, prefix, or object as the copy source	String
s3:x-amz-grant-full-control	Filters access to requests with the x-amz-grant-full-control (full control) header	String
s3:x-amz-grant-read	Filters access to requests with the x-amz-grant-read (read access) header	String
s3:x-amz-grant-read-acp	Filters access to requests with the x-amz-grant-read-acp (read permissions for the ACL) header	String
s3:x-amz-grant-write	Filters access to requests with the x-amz-grant-write (write access) header	String
s3:x-amz-grant-write-acp	Filters access to requests with the x-amz-grant-write-acp (write permissions for the ACL) header	String
s3:x-amz-metadata-directive	Filters access by object metadata behavior (COPY or REPLACE) when objects are copied	String
s3:x-amz-server-side-encryption	Filters access by server-side encryption	String

Condition keys	Description	Type
s3:x-amz-server-side-encryption-aws-kms-key-id	Filters access by AWS KMS key for server-side encryption	String
s3:x-amz-server-side-encryption-customer-algorithm	Filters access by customer specified algorithm for server-side encryption	String
s3:x-amz-storage-class	Filters access by storage class	String
s3:x-amz-website-redirect-location	Filters access by a specific website redirect location for buckets that are configured as static websites	String

Using bucket policies

You can create and configure bucket policies to grant permission to your Amazon S3 resources.

A bucket policy is a resource-based policy that you can use to grant access permissions to your bucket and the objects in it. Only the bucket owner can associate a policy with a bucket. The permissions attached to the bucket apply to all of the objects in the bucket that are owned by the bucket owner. These permissions do not apply to objects owned by other AWS accounts.

By default, when another AWS account uploads an object to your S3 bucket, that account (the object writer) owns the object, has access to it, and can grant other users access to it through ACLs. You can use Object Ownership to change this default behavior so that ACLs are disabled and you, as the bucket owner, automatically own every object in your bucket. As a result, access control for your data is based on policies, such as IAM policies, S3 bucket policies, virtual private cloud (VPC) endpoint policies, and AWS Organizations service control policies (SCPs). For more information, see [Controlling ownership of objects and disabling ACLs for your bucket \(p. 601\)](#).

Bucket policies use JSON-based access policy language. You can use bucket policies to add or deny permissions for the objects in a bucket. Bucket policies allow or deny requests based on the elements in the policy, including the requester, S3 actions, resources, and aspects or conditions of the request (for example, the IP address used to make the request). For example, you can create a bucket policy that grants cross-account permissions to upload objects to an S3 bucket while ensuring that the bucket owner has full control of the uploaded objects. For more information, see [Bucket policy examples \(p. 490\)](#).

In your bucket policy, you can use wildcard characters on Amazon Resource Names (ARNs) and other values to grant permissions to a subset of objects. For example, you can control access to groups of objects that begin with a common [prefix](#) or end with a given extension, such as `.html`.

The topics in this section provide examples and show you how to add a bucket policy in the S3 console. For information about IAM user policies, see [Using IAM user policies \(p. 499\)](#). For information about bucket policy language, see [Policies and Permissions in Amazon S3 \(p. 411\)](#)

Important

Bucket policies are limited to 20 KB in size.

Topics

- [Adding a bucket policy using the Amazon S3 console \(p. 488\)](#)
- [Controlling access from VPC endpoints with bucket policies \(p. 489\)](#)
- [Bucket policy examples \(p. 490\)](#)

Adding a bucket policy using the Amazon S3 console

You can use the Amazon S3 console to add a new bucket policy or edit an existing bucket policy. A bucket policy is a resource-based AWS Identity and Access Management (IAM) policy. You add a bucket policy to a bucket to grant other AWS accounts or IAM users access permissions for the bucket and the objects in it. Object permissions apply only to the objects that the bucket owner creates. For more information about bucket policies, see [Overview of managing access \(p. 395\)](#).

By default, when another AWS account uploads an object to your S3 bucket, that account (the object writer) owns the object, has access to it, and can grant other users access to it through ACLs. You can use Object Ownership to change this default behavior so that ACLs are disabled and you, as the bucket owner, automatically own every object in your bucket. As a result, access control for your data is based on policies, such as IAM policies, S3 bucket policies, virtual private cloud (VPC) endpoint policies, and AWS Organizations service control policies (SCPs). For more information, see [Controlling ownership of objects and disabling ACLs for your bucket \(p. 601\)](#).

Make sure to resolve security warnings, errors, general warnings, and suggestions from AWS Identity and Access Management Access Analyzer before you save your policy. IAM Access Analyzer runs policy checks to validate your policy against IAM [policy grammar](#) and [best practices](#). These checks generate findings and provide actionable recommendations to help you author policies that are functional and conform to security best practices. To learn more about validating policies using IAM Access Analyzer, see [IAM Access Analyzer policy validation](#) in the *IAM User Guide*. To view a list of the warnings, errors, and suggestions that are returned by IAM Access Analyzer, see [IAM Access Analyzer policy check reference](#).

To create or edit a bucket policy

1. Sign in to the AWS Management Console and open the Amazon S3 console at <https://console.aws.amazon.com/s3/>.
2. In the **Buckets** list, choose the name of the bucket that you want to create a bucket policy for or whose bucket policy you want to edit.
3. Choose **Permissions**.
4. Under **Bucket policy**, choose **Edit**. This opens the Edit bucket policy page.
5. On the **Edit bucket policy** page, explore **Policy examples** in the *Amazon S3 User Guide*, choose **Policy generator** to generate a policy automatically, or edit the JSON in the **Policy** section.

If you choose **Policy generator**, the AWS Policy Generator opens in a new window:

- a. On the **AWS Policy Generator** page, in **Select Type of Policy**, choose **S3 Bucket Policy**.
- b. Add a statement by entering the information in the provided fields, and then choose **Add Statement**. Repeat for as many statements as you would like to add. For more information about these fields, see the [IAM JSON policy elements reference](#) in the *IAM User Guide*.

Note

For convenience, the **Edit bucket policy** page displays the **Bucket ARN** (Amazon Resource Name) of the current bucket above the **Policy** text field. You can copy this ARN for use in the statements on the **AWS Policy Generator** page.

- c. After you finish adding statements, choose **Generate Policy**.
- d. Copy the generated policy text, choose **Close**, and return to the **Edit bucket policy** page in the Amazon S3 console.
6. In the **Policy** box, edit the existing policy or paste the bucket policy from the Policy generator. Make sure to resolve security warnings, errors, general warnings, and suggestions before you save your policy.
7. (Optional) Preview how your new policy affects public and cross-account access to your resource. Before you save your policy, you can check whether it introduces new IAM Access Analyzer findings or resolves existing findings. If you don't see an active analyzer, [create an account analyzer](#) in IAM Access Analyzer. For more information, see [Preview access](#) in the *IAM User Guide*.

8. Choose **Save changes**, which returns you to the Bucket Permissions page.

Controlling access from VPC endpoints with bucket policies

You can use Amazon S3 bucket policies to control access to buckets from specific virtual private cloud (VPC) endpoints, or specific VPCs. This section contains example bucket policies that can be used to control Amazon S3 bucket access from VPC endpoints. To learn how to set up VPC endpoints, see [VPC Endpoints](#) in the [VPC User Guide](#).

VPC enables you to launch AWS resources into a virtual network that you define. A VPC endpoint enables you to create a private connection between your VPC and another AWS service without requiring access over the internet, through a VPN connection, through a NAT instance, or through AWS Direct Connect.

A VPC endpoint for Amazon S3 is a logical entity within a VPC that allows connectivity only to Amazon S3. The VPC endpoint routes requests to Amazon S3 and routes responses back to the VPC. VPC endpoints change only how requests are routed. Amazon S3 public endpoints and DNS names will continue to work with VPC endpoints. For important information about using VPC endpoints with Amazon S3, see [Gateway VPC Endpoints](#) and [Endpoints for Amazon S3](#) in the [VPC User Guide](#).

VPC endpoints for Amazon S3 provide two ways to control access to your Amazon S3 data:

- You can control the requests, users, or groups that are allowed through a specific VPC endpoint. For information about this type of access control, see [Controlling Access to Services with VPC Endpoints](#) in the [VPC User Guide](#).
- You can control which VPCs or VPC endpoints have access to your buckets by using Amazon S3 bucket policies. For examples of this type of bucket policy access control, see the following topics on restricting access.

Topics

- [Restricting access to a specific VPC endpoint \(p. 489\)](#)
- [Restricting access to a specific VPC \(p. 490\)](#)

Important

When applying the Amazon S3 bucket policies for VPC endpoints described in this section, you might block your access to the bucket without intending to do so. Bucket permissions that are intended to specifically limit bucket access to connections originating from your VPC endpoint can block all connections to the bucket. For information about how to fix this issue, see [My bucket policy has the wrong VPC or VPC endpoint ID. How can I fix the policy so that I can access the bucket?](#) in the [AWS Support Knowledge Center](#).

Restricting access to a specific VPC endpoint

The following is an example of an Amazon S3 bucket policy that restricts access to a specific bucket, `awsexamplebucket1`, only from the VPC endpoint with the ID `vpce-1a2b3c4d`. The policy denies all access to the bucket if the specified endpoint is not being used. The `aws:SourceVpce` condition is used to specify the endpoint. The `aws:SourceVpce` condition does not require an Amazon Resource Name (ARN) for the VPC endpoint resource, only the VPC endpoint ID. For more information about using conditions in a policy, see [Amazon S3 condition key examples \(p. 420\)](#).

Important

- Before using the following example policy, replace the VPC endpoint ID with an appropriate value for your use case. Otherwise, you won't be able to access your bucket.
- This policy disables console access to the specified bucket, because console requests don't originate from the specified VPC endpoint.

```
{
    "Version": "2012-10-17",
    "Id": "Policy1415115909152",
    "Statement": [
        {
            "Sid": "Access-to-specific-VPCE-only",
            "Principal": "*",
            "Action": "s3:*",
            "Effect": "Deny",
            "Resource": ["arn:aws:s3:::awsexamplebucket1",
                        "arn:aws:s3:::awsexamplebucket1/*"],
            "Condition": {
                "StringNotEquals": {
                    "aws:SourceVpce": "vpce-1a2b3c4d"
                }
            }
        }
    ]
}
```

Restricting access to a specific VPC

You can create a bucket policy that restricts access to a specific VPC by using the `aws:SourceVpc` condition. This is useful if you have multiple VPC endpoints configured in the same VPC, and you want to manage access to your Amazon S3 buckets for all of your endpoints. The following is an example of a policy that allows VPC `vpc-111bbb22` to access `awsexamplebucket1` and its objects. The policy denies all access to the bucket if the specified VPC is not being used. The `vpc-111bbb22` condition key does not require an ARN for the VPC resource, only the VPC ID.

Important

- Before using the following example policy, replace the VPC ID with an appropriate value for your use case. Otherwise, you won't be able to access your bucket.
- This policy disables console access to the specified bucket, because console requests don't originate from the specified VPC.

```
{
    "Version": "2012-10-17",
    "Id": "Policy1415115909153",
    "Statement": [
        {
            "Sid": "Access-to-specific-VPC-only",
            "Principal": "*",
            "Action": "s3:*",
            "Effect": "Deny",
            "Resource": ["arn:aws:s3:::awsexamplebucket1",
                        "arn:aws:s3:::awsexamplebucket1/*"],
            "Condition": {
                "StringNotEquals": {
                    "aws:SourceVpc": "vpc-111bbb22"
                }
            }
        }
    ]
}
```

Bucket policy examples

This section presents a few examples of typical use cases for bucket policies. The policies use `DOC-EXAMPLE-BUCKET` strings in the resource value. To test these policies, replace these strings with your

bucket name. For information about bucket policies, see [Using bucket policies \(p. 487\)](#). For more information about policy language, see [Policies and Permissions in Amazon S3 \(p. 411\)](#).

A bucket policy is a resource-based policy that you can use to grant access permissions to your bucket and the objects in it. Only the bucket owner can associate a policy with a bucket. The permissions attached to the bucket apply to all of the objects in the bucket that are owned by the bucket owner. These permissions do not apply to objects owned by other AWS accounts.

By default, when another AWS account uploads an object to your S3 bucket, that account (the object writer) owns the object, has access to it, and can grant other users access to it through ACLs. You can use Object Ownership to change this default behavior so that ACLs are disabled and you, as the bucket owner, automatically own every object in your bucket. As a result, access control for your data is based on policies, such as IAM policies, S3 bucket policies, virtual private cloud (VPC) endpoint policies, and AWS Organizations service control policies (SCPs). For more information, see [Controlling ownership of objects and disabling ACLs for your bucket \(p. 601\)](#).

For more information about bucket policies, see [Using bucket policies \(p. 487\)](#).

Note

Bucket policies are limited to 20 KB in size.

You can use the [AWS Policy Generator](#) to create a bucket policy for your Amazon S3 bucket. You can then use the generated document to set your bucket policy by using the [Amazon S3 console](#), through several third-party tools, or via your application.

Important

When testing permissions using the Amazon S3 console, you must grant additional permissions that the console requires—`s3>ListAllMyBuckets`, `s3:GetBucketLocation`, and `s3>ListBucket` permissions. For an example walkthrough that grants permissions to users and tests them using the console, see [Controlling access to a bucket with user policies \(p. 499\)](#).

Topics

- [Granting permissions to multiple accounts with added conditions \(p. 491\)](#)
- [Granting read-only permission to an anonymous user \(p. 492\)](#)
- [Limiting access to specific IP addresses \(p. 493\)](#)
- [Restricting access to a specific HTTP referer \(p. 494\)](#)
- [Granting permission to an Amazon CloudFront OAI \(p. 495\)](#)
- [Adding a bucket policy to require MFA \(p. 495\)](#)
- [Granting cross-account permissions to upload objects while ensuring the bucket owner has full control \(p. 497\)](#)
- [Granting permissions for Amazon S3 Inventory and Amazon S3 analytics \(p. 497\)](#)
- [Granting permissions for Amazon S3 Storage Lens \(p. 498\)](#)

[Granting permissions to multiple accounts with added conditions](#)

The following example policy grants the `s3:PutObject` and `s3:PutObjectAcl` permissions to multiple AWS accounts and requires that any request for these operations include the `public-read` canned access control list (ACL). For more information, see [Amazon S3 actions \(p. 415\)](#) and [Amazon S3 condition key examples \(p. 420\)](#).

Warning

Use caution when granting anonymous access to your Amazon S3 bucket or disabling block public access settings. When you grant anonymous access, anyone in the world can access your bucket. We recommend that you never grant anonymous access to your Amazon S3 bucket unless you specifically need to, such as with [static website hosting \(p. 1116\)](#).

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "AddCannedAcl",
            "Effect": "Allow",
            "Principal": {
                "AWS": [
                    "arn:aws:iam::111122223333:root",
                    "arn:aws:iam::44445556666:root"
                ]
            },
            "Action": [
                "s3:PutObject",
                "s3:PutObjectAcl"
            ],
            "Resource": "arn:aws:s3:::DOC-EXAMPLE-BUCKET/*",
            "Condition": {
                "StringEquals": {
                    "s3:x-amz-acl": [
                        "public-read"
                    ]
                }
            }
        }
    ]
}
```

Granting read-only permission to an anonymous user

The following example policy grants the s3:GetObject permission to any public anonymous users. (For a list of permissions and the operations that they allow, see [Amazon S3 actions \(p. 415\)](#).) This permission allows anyone to read the object data, which is useful for when you configure your bucket as a website and want everyone to be able to read objects in the bucket. Before you use a bucket policy to grant read-only permission to an anonymous user, you must disable block public access settings for your bucket. For more information, see [Setting permissions for website access \(p. 1126\)](#).

Warning

Use caution when granting anonymous access to your Amazon S3 bucket or disabling block public access settings. When you grant anonymous access, anyone in the world can access your bucket. We recommend that you never grant anonymous access to your Amazon S3 bucket unless you specifically need to, such as with [static website hosting \(p. 1116\)](#).

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "PublicRead",
            "Effect": "Allow",
            "Principal": "*",
            "Action": [
                "s3:GetObject",
                "s3:GetObjectVersion"
            ],
            "Resource": [
                "arn:aws:s3:::DOC-EXAMPLE-BUCKET/*"
            ]
        }
    ]
}
```

Limits access to specific IP addresses

The following example denies permissions to any user to perform any Amazon S3 operations on objects in the specified S3 bucket unless the request originates from the range of IP addresses specified in the condition.

This statement identifies [54.240.143.0/24](#) as the range of allowed Internet Protocol version 4 (IPv4) IP addresses.

The Condition block uses the `NotIpAddress` condition and the `aws:SourceIp` condition key, which is an AWS-wide condition key. For more information about these condition keys, see [Amazon S3 condition key examples \(p. 420\)](#). The `aws:SourceIp` IPv4 values use the standard CIDR notation. For more information, see [IAM JSON Policy Elements Reference](#) in the *IAM User Guide*.

Warning

Before using this policy, replace the [54.240.143.0/24](#) IP address range in this example with an appropriate value for your use case. Otherwise, you will lose the ability to access your bucket.

```
{  
    "Version": "2012-10-17",  
    "Id": "S3PolicyId1",  
    "Statement": [  
        {  
            "Sid": "IPAllow",  
            "Effect": "Deny",  
            "Principal": "*",  
            "Action": "s3:*",  
            "Resource": [  
                "arn:aws:s3::::DOC-EXAMPLE-BUCKET",  
                "arn:aws:s3::::DOC-EXAMPLE-BUCKET/*"  
            ],  
            "Condition": {  
                "NotIpAddress": {  
                    "aws:SourceIp": "54.240.143.0/24"  
                }  
            }  
        }  
    ]  
}
```

Allows IPv4 and IPv6 addresses

When you start using IPv6 addresses, we recommend that you update all of your organization's policies with your IPv6 address ranges in addition to your existing IPv4 ranges to ensure that the policies continue to work as you make the transition to IPv6.

The following example bucket policy shows how to mix IPv4 and IPv6 address ranges to cover all of your organization's valid IP addresses. The example policy would allow access to the example IP addresses [54.240.143.1](#) and [2001:DB8:1234:5678::1](#) and would deny access to the addresses [54.240.143.129](#) and [2001:DB8:1234:5678:ABCD::1](#).

The IPv6 values for `aws:SourceIp` must be in standard CIDR format. For IPv6, we support using `::` to represent a range of 0s (for example, `2032001:DB8:1234:5678::/64`). For more information, see [IP Address Condition Operators](#) in the *IAM User Guide*.

Warning

Replace the IP address ranges in this example with appropriate values for your use case before using this policy. Otherwise, you might lose the ability to access your bucket.

```
{  
    "Id": "PolicyId2",  
    "Version": "2012-10-17",
```

```

"Statement": [
    {
        "Sid": "AllowIPmix",
        "Effect": "Allow",
        "Principal": "*",
        "Action": "s3:*",
        "Resource": [
            "arn:aws:s3:::DOC-EXAMPLE-BUCKET",
            "arn:aws:s3:::DOC-EXAMPLE-BUCKET/*"
        ],
        "Condition": {
            "IpAddress": {
                "aws:SourceIp": [
                    "54.240.143.0/24",
                    "2001:DB8:1234:5678::/64"
                ]
            },
            "NotIpAddress": {
                "aws:SourceIp": [
                    "54.240.143.128/30",
                    "2001:DB8:1234:5678:ABCD::/80"
                ]
            }
        }
    }
]
}

```

Restricting access to a specific HTTP referer

Suppose that you have a website with a domain name (www.example.com or example.com) with links to photos and videos stored in your Amazon S3 bucket, **DOC-EXAMPLE-BUCKET**. By default, all the Amazon S3 resources are private, so only the AWS account that created the resources can access them. To allow read access to these objects from your website, you can add a bucket policy that allows s3:GetObject permission with a condition, using the `aws:Referer` key, that the get request must originate from specific webpages. The following policy specifies the `StringLike` condition with the `aws:Referer` condition key.

```

{
    "Version": "2012-10-17",
    "Id": "http referer policy example",
    "Statement": [
        {
            "Sid": "Allow get requests originating from www.example.com and example.com.",
            "Effect": "Allow",
            "Principal": "*",
            "Action": ["s3:GetObject", "s3:GetObjectVersion"],
            "Resource": "arn:aws:s3:::DOC-EXAMPLE-BUCKET/*",
            "Condition": {
                "StringLike": {"aws:Referer": ["http://www.example.com/*", "http://example.com/*"]}
            }
        }
    ]
}

```

Make sure the browsers that you use include the `HTTP referer` header in the request.

Warning

We recommend that you use caution when using the `aws:Referer` condition key. It is dangerous to include a publicly known referer header value. Unauthorized parties can use modified or custom browsers to provide any `aws:Referer` value that they choose. Therefore, do not use `aws:Referer` to prevent unauthorized parties from making direct AWS requests.

The `aws:Referer` condition key is offered only to allow customers to protect their digital content, such as content stored in Amazon S3, from being referenced on unauthorized third-party sites. For more information, see `aws:Referer` in the *IAM User Guide*.

Granting permission to an Amazon CloudFront OAI

The following example bucket policy grants a CloudFront origin access identity (OAI) permission to get (read) all objects in your Amazon S3 bucket. You can use a CloudFront OAI to allow users to access objects in your bucket through CloudFront but not directly through Amazon S3. For more information, see [Restricting Access to Amazon S3 Content by Using an Origin Access Identity](#) in the *Amazon CloudFront Developer Guide*.

The following policy uses the OAI's ID as the policy's `Principal`. For more information about using S3 bucket policies to grant access to a CloudFront OAI, see [Using Amazon S3 Bucket Policies](#) in the *Amazon CloudFront Developer Guide*.

To use this example:

- Replace `EH1HDMB1FH2TC` with the OAI's ID. To find the OAI's ID, see the [Origin Access Identity page](#) on the CloudFront console, or use `ListCloudFrontOriginAccessIdentities` in the CloudFront API.
- Replace `DOC-EXAMPLE-BUCKET` with the name of your Amazon S3 bucket.

```
{  
    "Version": "2012-10-17",  
    "Id": "PolicyForCloudFrontPrivateContent",  
    "Statement": [  
        {  
            "Effect": "Allow",  
            "Principal": {  
                "AWS": "arn:aws:iam::cloudfront:user/CloudFront Origin Access  
Identity EH1HDMB1FH2TC"  
            },  
            "Action": "s3:GetObject",  
            "Resource": "arn:aws:s3:::DOC-EXAMPLE-BUCKET/*"  
        }  
    ]  
}
```

Adding a bucket policy to require MFA

Amazon S3 supports MFA-protected API access, a feature that can enforce multi-factor authentication (MFA) for access to your Amazon S3 resources. Multi-factor authentication provides an extra level of security that you can apply to your AWS environment. It is a security feature that requires users to prove physical possession of an MFA device by providing a valid MFA code. For more information, see [AWS Multi-Factor Authentication](#). You can require MFA for any requests to access your Amazon S3 resources.

You can enforce the MFA requirement using the `aws:MultiFactorAuthAge` key in a bucket policy. AWS Identity and Access Management (IAM) users can access Amazon S3 resources by using temporary credentials issued by the AWS Security Token Service (AWS STS). You provide the MFA code at the time of the AWS STS request.

When Amazon S3 receives a request with multi-factor authentication, the `aws:MultiFactorAuthAge` key provides a numeric value indicating how long ago (in seconds) the temporary credential was created. If the temporary credential provided in the request was not created using an MFA device, this key value is null (absent). In a bucket policy, you can add a condition to check this value, as shown in the following example bucket policy. This example policy denies any Amazon S3 operation on the `/taxdocuments` folder in the `DOC-EXAMPLE-BUCKET` bucket if the request is not authenticated using MFA. To learn more about MFA, see [Using Multi-Factor Authentication \(MFA\) in AWS](#) in the *IAM User Guide*.

```
{
    "Version": "2012-10-17",
    "Id": "123",
    "Statement": [
        {
            "Sid": "",
            "Effect": "Deny",
            "Principal": "*",
            "Action": "s3:*",
            "Resource": "arn:aws:s3:::DOC-EXAMPLE-BUCKET/taxdocuments/\*",
            "Condition": { "Null": { "aws:MultiFactorAuthAge": true } }
        }
    \]
}
```

The Null condition in the Condition block evaluates to true if the aws:MultiFactorAuthAge key value is null, indicating that the temporary security credentials in the request were created without the MFA key.

The following bucket policy is an extension of the preceding bucket policy. It includes two policy statements. One statement allows the s3:GetObject permission on a bucket ([DOC-EXAMPLE-BUCKET](#)) to everyone. Another statement further restricts access to the [DOC-EXAMPLE-BUCKET/taxdocuments](#) folder in the bucket by requiring MFA.

```
{
    "Version": "2012-10-17",
    "Id": "123",
    "Statement": [
        {
            "Sid": "",
            "Effect": "Deny",
            "Principal": "*",
            "Action": "s3:*",
            "Resource": "arn:aws:s3:::DOC-EXAMPLE-BUCKET/taxdocuments/\*",
            "Condition": { "Null": { "aws:MultiFactorAuthAge": true } }
        },
        {
            "Sid": "",
            "Effect": "Allow",
            "Principal": "\*",
            "Action": \["s3:GetObject"\],
            "Resource": "arn:aws:s3:::DOC-EXAMPLE-BUCKET/\\*"
        }
    \\]
}
```

You can optionally use a numeric condition to limit the duration for which the aws:MultiFactorAuthAge key is valid, independent of the lifetime of the temporary security credential used in authenticating the request. For example, the following bucket policy, in addition to requiring MFA authentication, also checks how long ago the temporary session was created. The policy denies any operation if the aws:MultiFactorAuthAge key value indicates that the temporary session was created more than an hour ago (3,600 seconds).

```
{
    "Version": "2012-10-17",
    "Id": "123",
    "Statement": [
        {
            "Sid": "",
            "Effect": "Deny",
            "Principal": "*",
            "Action": "s3:*",
            "Resource": "arn:aws:s3:::DOC-EXAMPLE-BUCKET/taxdocuments/\*",
            "Condition": { "LessThan": { "aws:MultiFactorAuthAge": 3600 } }
        }
    \]
}
```

```

        "Action": "s3:*",
        "Resource": "arn:aws:s3:::DOC-EXAMPLE-BUCKET/taxdocuments/*",
        "Condition": {"Null": {"aws:MultiFactorAuthAge": true}}
    },
    {
        "Sid": "",
        "Effect": "Deny",
        "Principal": "*",
        "Action": "s3:*",
        "Resource": "arn:aws:s3:::DOC-EXAMPLE-BUCKET/taxdocuments/*",
        "Condition": {"NumericGreater Than": {"aws:MultiFactorAuthAge": 3600}}
    },
    {
        "Sid": "",
        "Effect": "Allow",
        "Principal": "*",
        "Action": ["s3:GetObject"],
        "Resource": "arn:aws:s3:::DOC-EXAMPLE-BUCKET/*"
    }
]
}

```

Granting cross-account permissions to upload objects while ensuring the bucket owner has full control

The following example shows how to allow another AWS account to upload objects to your bucket while taking full control of the uploaded objects. This policy enforces that a specific AWS account ([111122223333](#)) be granted the ability to upload objects only if that account includes the bucket-owner-full-control canned ACL on upload. The `StringEquals` condition in the policy specifies the `s3:x-amz-acl` condition key to express the requirement (see [Amazon S3 condition key examples \(p. 420\)](#)).

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "PolicyForAllowUploadWithACL",
            "Effect": "Allow",
            "Principal": {"AWS": "111122223333"},
            "Action": "s3:PutObject",
            "Resource": "arn:aws:s3:::DOC-EXAMPLE-BUCKET/*",
            "Condition": {
                "StringEquals": {"s3:x-amz-acl": "bucket-owner-full-control"}
            }
        }
    ]
}
```

Granting permissions for Amazon S3 Inventory and Amazon S3 analytics

Amazon S3 Inventory creates lists of the objects in an Amazon S3 bucket, and Amazon S3 analytics export creates output files of the data used in the analysis. The bucket that the inventory lists the objects for is called the *source bucket*. The bucket where the inventory file is written and the bucket where the analytics export file is written is called a *destination bucket*. You must create a bucket policy for the destination bucket when setting up inventory for an Amazon S3 bucket and when setting up the analytics export. For more information, see [Amazon S3 Inventory \(p. 739\)](#) and [Amazon S3 analytics – Storage Class Analysis \(p. 1048\)](#).

The following example bucket policy grants Amazon S3 permission to write objects (PUTs) from the account for the source bucket to the destination bucket. You use a bucket policy like this on the destination bucket when setting up Amazon S3 Inventory and Amazon S3 analytics export.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "InventoryAndAnalyticsExamplePolicy",
            "Effect": "Allow",
            "Principal": {
                "Service": "s3.amazonaws.com"
            },
            "Action": "s3:PutObject",
            "Resource": [
                "arn:aws:s3:::destinationbucket/*"
            ],
            "Condition": {
                "ArnLike": {
                    "aws:SourceArn": "arn:aws:s3:::sourcebucket"
                },
                "StringEquals": {
                    "aws:SourceAccount": "111122223333",
                    "s3:x-amz-acl": "bucket-owner-full-control"
                }
            }
        }
    ]
}
```

Granting permissions for Amazon S3 Storage Lens

Amazon S3 Storage Lens aggregates your usage and activity metrics and displays the information in the account snapshot on the Amazon S3 console home (**Buckets**) page, interactive dashboards, or through a metrics export that you can download in CSV or Parquet format. You can use the dashboard to visualize insights and trends, flag outliers, and receive recommendations for optimizing storage costs and applying data protection best practices. You can use S3 Storage Lens through the AWS Management Console, AWS CLI, AWS SDKs, or REST API.

S3 Storage Lens can aggregate your storage usage to metrics exports in an Amazon S3 bucket for further analysis. The bucket that S3 Storage Lens places its metrics exports is known as the *destination bucket*. You must have a bucket policy for the *destination bucket* when setting up your S3 Storage Lens metrics export. For more information, see [Assessing your storage activity and usage with Amazon S3 Storage Lens \(p. 1053\)](#).

The following example bucket policy grants Amazon S3 permission to write objects (PUTs) to a *destination bucket*. You use a bucket policy like this on the *destination bucket* when setting up an S3 Storage Lens metrics export.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "S3StorageLensExamplePolicy",
            "Effect": "Allow",
            "Principal": {
                "Service": "storage-lens.s3.amazonaws.com"
            },
            "Action": "s3:PutObject",
            "Resource": [
                "arn:aws:s3:::destination-bucket/destination-prefix/"
StorageLens/111122223333/*"
            ],
            "Condition": {
                "StringEquals": {
                    "s3:x-amz-acl": "bucket-owner-full-control",
                    "s3:x-amz-storage-lens": "true"
                }
            }
        }
    ]
}
```

```
        "aws:SourceAccount": "111122223333",
        "aws:SourceArn": "arn:aws:s3:AWS Region:111122223333:storage-
lens/storage-lens-dashboard-configuration-id"
    }
}
]
```

Use the following modification to the previous bucket policy Resource when setting up an S3 Storage Lens organization-level metrics export.

```
"Resource": "arn:aws:s3:::destination-bucket/destination-prefix/
StorageLens/your-organization-id/*",
```

Using IAM user policies

You can create and configure IAM user policies for controlling user access to Amazon S3. User policies use JSON-based access policy language.

This section shows several IAM user policies for controlling user access to Amazon S3. For example *bucket policies*, see [Using bucket policies \(p. 487\)](#). For information about access policy language, see [Policies and Permissions in Amazon S3 \(p. 411\)](#).

Topics

- [Controlling access to a bucket with user policies \(p. 499\)](#)
- [User policy examples \(p. 516\)](#)

Controlling access to a bucket with user policies

This walkthrough explains how user permissions work with Amazon S3. In this example, you create a bucket with folders. You then create AWS Identity and Access Management (IAM) users in your AWS account and grant those users incremental permissions on your Amazon S3 bucket and the folders in it.

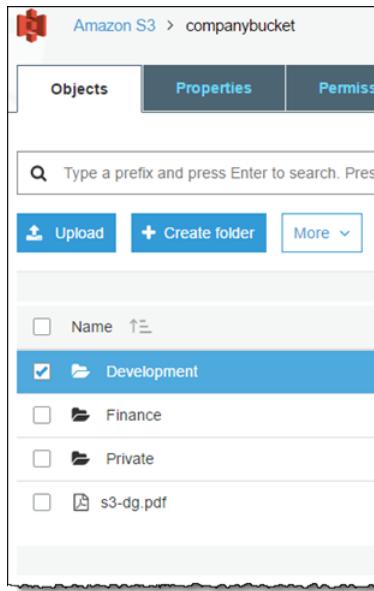
Topics

- [Basics of buckets and folders \(p. 499\)](#)
- [Walkthrough summary \(p. 501\)](#)
- [Preparing for the walkthrough \(p. 502\)](#)
- [Step 1: Create a bucket \(p. 502\)](#)
- [Step 2: Create IAM users and a group \(p. 503\)](#)
- [Step 3: Verify that IAM users have no permissions \(p. 503\)](#)
- [Step 4: Grant group-level permissions \(p. 504\)](#)
- [Step 5: Grant IAM user Alice specific permissions \(p. 510\)](#)
- [Step 6: Grant IAM user Bob specific permissions \(p. 514\)](#)
- [Step 7: Secure the private folder \(p. 514\)](#)
- [Step 8: Clean up \(p. 516\)](#)
- [Related resources \(p. 516\)](#)

Basics of buckets and folders

The Amazon S3 data model is a flat structure: You create a bucket, and the bucket stores objects. There is no hierarchy of subbuckets or subfolders, but you can emulate a folder hierarchy. Tools like the Amazon

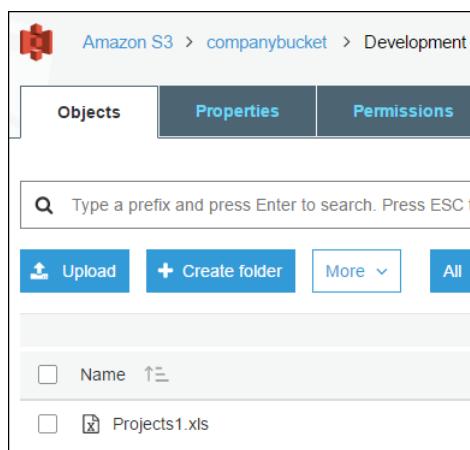
S3 console can present a view of these logical folders and subfolders in your bucket, as shown in the following image.



The console shows that a bucket named `companybucket` has three folders, `Private`, `Development`, and `Finance`, and an object, `s3-dg.pdf`. The console uses the object names (keys) to create a logical hierarchy with folders and subfolders. Consider the following examples:

- When you create the `Development` folder, the console creates an object with the key `Development/`. Note the trailing slash (/) delimiter.
- When you upload an object named `Projects1.xls` in the `Development` folder, the console uploads the object and gives it the key `Development/Projects1.xls`.

In the key, `Development` is the **prefix** and / is the delimiter. The Amazon S3 API supports prefixes and delimiters in its operations. For example, you can get a list of all objects from a bucket with a specific prefix and delimiter. On the console, when you open the `Development` folder, the console lists the objects in that folder. In the following example, the `Development` folder contains one object.



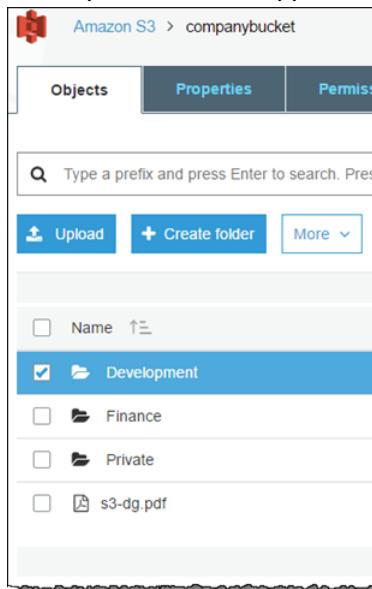
When the console lists the Development folder in the companybucket bucket, it sends a request to Amazon S3 in which it specifies a prefix of Development and a delimiter of / in the request. The console's response looks just like a folder list in your computer's file system. The preceding example shows that the bucket companybucket has an object with the key Development/Projects1.xls.

The console is using object keys to infer a logical hierarchy. Amazon S3 has no physical hierarchy; it only has buckets that contain objects in a flat file structure. When you create objects using the Amazon S3 API, you can use object keys that imply a logical hierarchy. When you create a logical hierarchy of objects, you can manage access to individual folders, as this walkthrough demonstrates.

Before you start, be sure that you are familiar with the concept of the *root-level* bucket content. Suppose that your companybucket bucket has the following objects:

- Private/privDoc1.txt
- Private/privDoc2.zip
- Development/project1.xls
- Development/project2.xls
- Finance/Tax2011/document1.pdf
- Finance/Tax2011/document2.pdf
- s3-dg.pdf

These object keys create a logical hierarchy with Private, Development, and the Finance as root-level folders and s3-dg.pdf as a root-level object. When you choose the bucket name on the Amazon S3 console, the root-level items appear as shown in the following image. The console shows the top-level prefixes (Private/, Development/, and Finance/) as root-level folders. The object key s3-dg.pdf has no prefix, and so it appears as a root-level item.



Walkthrough summary

In this walkthrough, you create a bucket with three folders (Private, Development, and Finance) in it.

You have two users, Alice and Bob. You want Alice to access only the Development folder, and you want Bob to access only the Finance folder. You want to keep the Private folder content private. In the

walkthrough, you manage access by creating IAM users (the example uses the user names Alice and Bob) and granting them the necessary permissions.

IAM also supports creating user groups and granting group-level permissions that apply to all users in the group. This helps you better manage permissions. For this exercise, both Alice and Bob need some common permissions. So you also create a group named Consultants and then add both Alice and Bob to the group. You first grant permissions by attaching a group policy to the group. Then you add user-specific permissions by attaching policies to specific users.

Note

The walkthrough uses companybucket as the bucket name, Alice and Bob as the IAM users, and Consultants as the group name. Because Amazon S3 requires that bucket names be globally unique, you must replace the bucket name with a name that you create.

Preparing for the walkthrough

In this example, you use your AWS account credentials to create IAM users. Initially, these users have no permissions. You incrementally grant these users permissions to perform specific Amazon S3 actions. To test these permissions, you sign in to the console with each user's credentials. As you incrementally grant permissions as an AWS account owner and test permissions as an IAM user, you need to sign in and out, each time using different credentials. You can do this testing with one browser, but the process will go faster if you can use two different browsers. Use one browser to connect to the AWS Management Console with your AWS account credentials and another to connect with the IAM user credentials.

To sign in to the AWS Management Console with your AWS account credentials, go to <https://console.aws.amazon.com/>. An IAM user cannot sign in using the same link. An IAM user must use an IAM-enabled sign-in page. As the account owner, you can provide this link to your users.

For more information about IAM, see [The AWS Management Console Sign-in Page in the IAM User Guide](#).

To provide a sign-in link for IAM users

1. Sign in to the AWS Management Console and open the IAM console at <https://console.aws.amazon.com/iam/>.
2. In the **Navigation** pane, choose **IAM Dashboard**.
3. Note the URL under **IAM users sign in link**. You will give this link to IAM users to sign in to the console with their IAM user name and password.

Step 1: Create a bucket

In this step, you sign in to the Amazon S3 console with your AWS account credentials, create a bucket, add folders (Development, Finance, and Private) to the bucket, and upload one or two sample documents in each folder.

1. Sign in to the AWS Management Console and open the Amazon S3 console at <https://console.aws.amazon.com/s3/>.
2. Create a bucket.

For step-by-step instructions, see [Creating a bucket \(p. 119\)](#).

3. Upload one document to the bucket.

This exercise assumes that you have the s3-dg.pdf document at the root level of this bucket. If you upload a different document, substitute its file name for s3-dg.pdf.

4. Add three folders named Private, Finance, and Development to the bucket.

For step-by-step instructions to create a folder, see [Organizing objects in the Amazon S3 console using folders \(p. 254\)](#) in the *Amazon Simple Storage Service User Guide*.

5. Upload one or two documents to each folder.

For this exercise, assume that you have uploaded a couple of documents in each folder, resulting in the bucket having objects with the following keys:

- Private/privDoc1.txt
- Private/privDoc2.zip
- Development/project1.xls
- Development/project2.xls
- Finance/Tax2011/document1.pdf
- Finance/Tax2011/document2.pdf
- s3-dg.pdf

For step-by-step instructions, see [Uploading objects \(p. 158\)](#).

Step 2: Create IAM users and a group

Now use the IAM console to add two IAM users, Alice and Bob, to your AWS account. Also create an administrative group named `Consultants`, and then add both users to the group.

Warning

When you add users and a group, do not attach any policies that grant permissions to these users. At first, these users don't have any permissions. In the following sections, you grant permissions incrementally. First you must ensure that you have assigned passwords to these IAM users. You use these user credentials to test Amazon S3 actions and verify that the permissions work as expected.

For step-by-step instructions for creating a new IAM user, see [Creating an IAM User in Your AWS account](#) in the *IAM User Guide*. When you create the users for this walkthrough, select **AWS Management Console access** and clear **Programmatic access**.

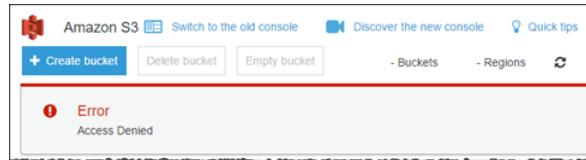
For step-by-step instructions for creating an administrative group, see [Creating Your First IAM Admin User and Group](#) in the *IAM User Guide*.

Step 3: Verify that IAM users have no permissions

If you are using two browsers, you can now use the second browser to sign in to the console using one of the IAM user credentials.

1. Using the IAM user sign-in link (see [To provide a sign-in link for IAM users \(p. 502\)](#)), sign in to the AWS Management Console using either of the IAM user credentials.
2. Open the Amazon S3 console at <https://console.aws.amazon.com/s3/>.

Verify the following console message telling you that access is denied.



Now, you can begin granting incremental permissions to the users. First, you attach a group policy that grants permissions that both users must have.

Step 4: Grant group-level permissions

You want the users to be able to do the following:

- List all buckets owned by the parent account. To do so, Bob and Alice must have permission for the `s3>ListAllMyBuckets` action.
- List root-level items, folders, and objects in the `companybucket` bucket. To do so, Bob and Alice must have permission for the `s3>ListBucket` action on the `companybucket` bucket.

First, you create a policy that grants these permissions, and then you attach it to the `Consultants` group.

Step 4.1: Grant permission to list all buckets

In this step, you create a managed policy that grants the users minimum permissions to enable them to list all buckets owned by the parent account. Then you attach the policy to the `Consultants` group. When you attach the managed policy to a user or a group, you grant the user or group permission to obtain a list of buckets owned by the parent AWS account.

1. Sign in to the AWS Management Console and open the IAM console at <https://console.aws.amazon.com/iam/>.

Note

Because you are granting user permissions, sign in using your AWS account credentials, not as an IAM user.

2. Create the managed policy.
 - a. In the navigation pane on the left, choose **Policies**, and then choose **Create Policy**.
 - b. Choose the **JSON** tab.
 - c. Copy the following access policy and paste it into the policy text field.

```
{  
    "Version": "2012-10-17",  
    "Statement": [  
        {  
            "Sid": "AllowGroupToSeeBucketListInTheConsole",  
            "Action": ["s3>ListAllMyBuckets"],  
            "Effect": "Allow",  
            "Resource": ["arn:aws:s3:::*"]  
        }  
    ]  
}
```

A policy is a JSON document. In the document, a `Statement` is an array of objects, each describing a permission using a collection of name-value pairs. The preceding policy describes one specific permission. The `Action` specifies the type of access. In the policy, the `s3>ListAllMyBuckets` is a predefined Amazon S3 action. This action covers the Amazon S3 GET Service operation, which returns list of all buckets owned by the authenticated sender. The `Effect` element value determines whether specific permission is allowed or denied.

- d. Choose **Review Policy**. On the next page, enter `AllowGroupToSeeBucketListInTheConsole` in the **Name** field, and then choose **Create policy**.

Note

The **Summary** entry displays a message stating that the policy does not grant any permissions. For this walkthrough, you can safely ignore this message.

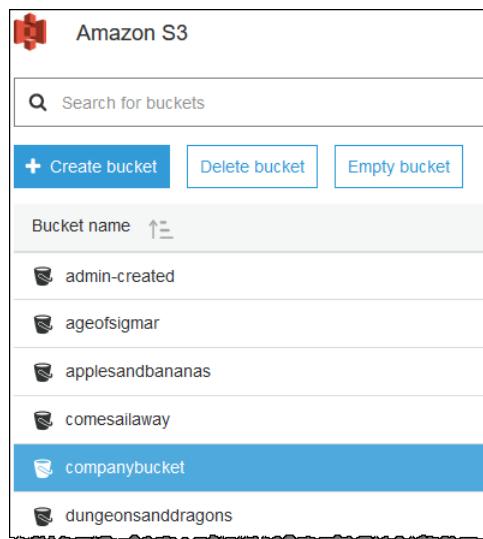
3. Attach the `AllowGroupToSeeBucketListInTheConsole` managed policy that you created to the `Consultants` group.

For step-by-step instructions for attaching a managed policy, see [Adding and Removing IAM Identity Permissions](#) in the *IAM User Guide*.

You attach policy documents to IAM users and groups in the IAM console. Because you want both users to be able to list the buckets, you attach the policy to the group.

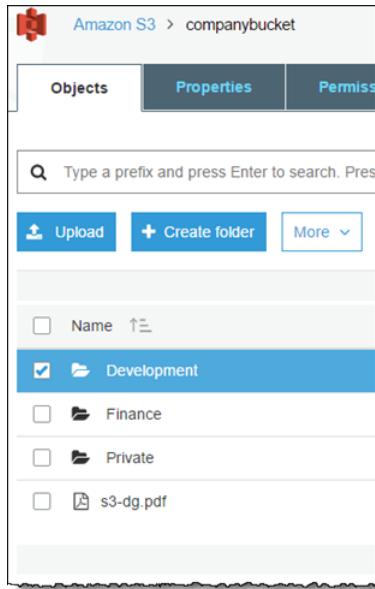
4. Test the permission.
 - a. Using the IAM user sign-in link (see [To provide a sign-in link for IAM users \(p. 502\)](#)), sign in to the console using any one of IAM user credentials.
 - b. Open the Amazon S3 console at <https://console.aws.amazon.com/s3/>.

The console should now list all the buckets but not the objects in any of the buckets.



Step 4.2: Enable users to list root-level content of a bucket

Next, you allow all users in the `Consultants` group to list the root-level `companybucket` bucket items. When a user chooses the company bucket on the Amazon S3 console, the user can see the root-level items in the bucket.



Note

This example uses `companybucket` for illustration. You must use the name of the bucket that you created.

To understand the request that the console sends to Amazon S3 when you choose a bucket name, the response that Amazon S3 returns, and how the console interprets the response, it is necessary to examine it a little more closely.

When you choose a bucket name, the console sends the [GET Bucket \(List Objects\)](#) request to Amazon S3. This request includes the following parameters:

- The `prefix` parameter with an empty string as its value.
- The `delimiter` parameter with `/` as its value.

The following is an example request.

```
GET ?prefix=&delimiter=/ HTTP/1.1
Host: companybucket.s3.amazonaws.com
Date: Wed, 01 Aug 2012 12:00:00 GMT
Authorization: AWS AKIAIOSFODNN7EXAMPLE:xQE0diMbLRepdf3YB+FIEXAMPLE=
```

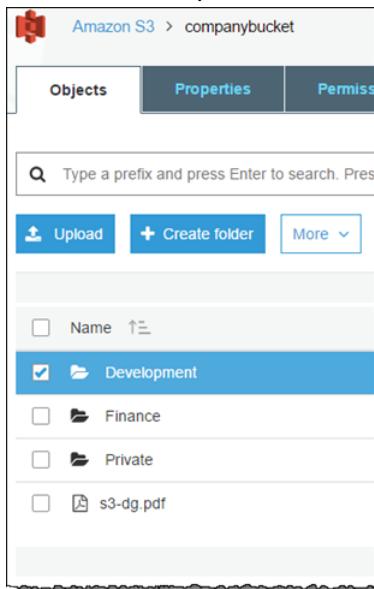
Amazon S3 returns a response that includes the following `<ListBucketResult>` element.

```
<ListBucketResult xmlns="http://s3.amazonaws.com/doc/2006-03-01/">
<Name>companybucket</Name>
<Prefix></Prefix>
<Delimiter>/<Delimiter>
...
<Contents>
<Key>s3-dg.pdf</Key>
...
</Contents>
<CommonPrefixes>
<Prefix>Development/<Prefix>
</CommonPrefixes>
<CommonPrefixes>
<Prefix>Finance/<Prefix>
```

```
</CommonPrefixes>
<CommonPrefixes>
  <Prefix>Private/</Prefix>
</CommonPrefixes>
</ListBucketResult>
```

The key `s3-dg.pdf` object does not contain the slash (/) delimiter, and Amazon S3 returns the key in the `<Contents>` element. However, all other keys in the example bucket contain the / delimiter. Amazon S3 groups these keys and returns a `<CommonPrefixes>` element for each of the distinct prefix values `Development/`, `Finance/`, and `Private/` that is a substring from the beginning of these keys to the first occurrence of the specified / delimiter.

The console interprets this result and displays the root-level items as three folders and one object key.



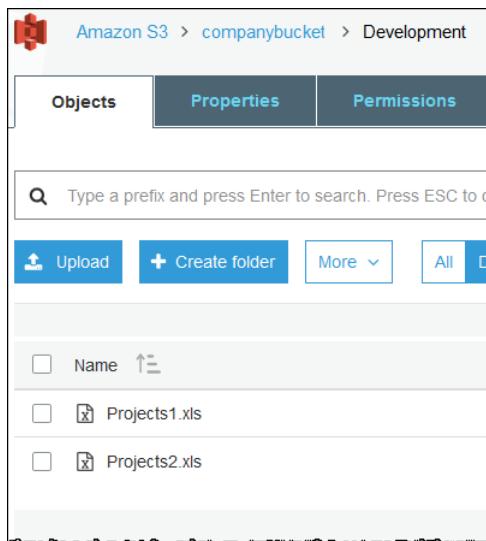
If Bob or Alice opens the **Development** folder, the console sends the [GET Bucket \(List Objects\)](#) request to Amazon S3 with the `prefix` and the `delimiter` parameters set to the following values:

- The `prefix` parameter with the value `Development/`.
- The `delimiter` parameter with the "/" value.

In response, Amazon S3 returns the object keys that start with the specified prefix.

```
<ListBucketResult xmlns="http://s3.amazonaws.com/doc/2006-03-01/">
  <Name>companybucket</Name>
  <Prefix>Development/<Prefix>
  <Delimiter>/<Delimiter>
  ...
  <Contents>
    <Key>Project1.xls</Key>
    ...
  </Contents>
  <Contents>
    <Key>Project2.xls</Key>
    ...
  </Contents>
</ListBucketResult>
```

The console shows the object keys.



Now, return to granting users permission to list the root-level bucket items. To list bucket content, users need permission to call the `s3>ListBucket` action, as shown in the following policy statement. To ensure that they see only the root-level content, you add a condition that users must specify an empty prefix in the request—that is, they are not allowed to double-click any of the root-level folders. Finally, you add a condition to require folder-style access by requiring user requests to include the `delimiter` parameter with the value `"/"`.

```
{
    "Sid": "AllowRootLevelListingOfCompanyBucket",
    "Action": ["s3>ListBucket"],
    "Effect": "Allow",
    "Resource": ["arn:aws:s3:::companybucket"],
    "Condition": {
        "StringEquals": {
            "s3:prefix": [""], "s3:delimiter": ["/"]
        }
    }
}
```

When you choose a bucket on the Amazon S3 console, the console first sends the [GET Bucket location](#) request to find the AWS Region where the bucket is deployed. Then the console uses the Region-specific endpoint for the bucket to send the [GET Bucket \(List Objects\)](#) request. As a result, if users are going to use the console, you must grant permission for the `s3:GetBucketLocation` action as shown in the following policy statement.

```
{
    "Sid": "RequiredByS3Console",
    "Action": ["s3:GetBucketLocation"],
    "Effect": "Allow",
    "Resource": ["arn:aws:s3:::*"]
}
```

To enable users to list root-level bucket content

1. Sign in to the AWS Management Console and open the IAM console at <https://console.aws.amazon.com/iam/>.

Use your AWS account credentials, not the credentials of an IAM user, to sign in to the console.

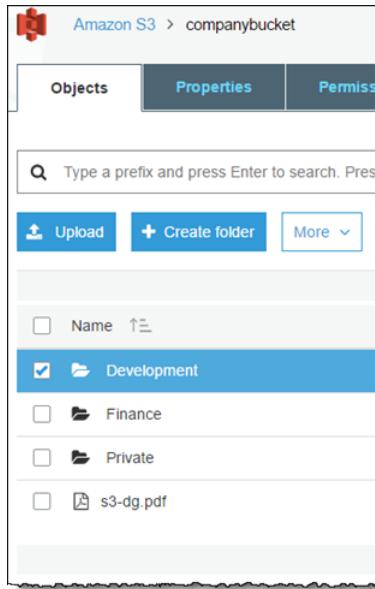
2. Replace the existing `AllowGroupToSeeBucketListInTheConsole` managed policy that is attached to the `Consultants` group with the following policy, which also allows the `s3>ListBucket` action. Remember to replace `companybucket` in the policy `Resource` with the name of your bucket.

For step-by-step instructions, see [Editing IAM Policies](#) in the *IAM User Guide*. When following the step-by-step instructions, be sure to follow the steps for applying your changes to all principal entities that the policy is attached to.

```
{  
    "Version": "2012-10-17",  
    "Statement": [  
        {  
            "Sid": "AllowGroupToSeeBucketListAndAlsoAllowGetBucketLocationRequiredForListBucket",  
            "Action": [ "s3>ListAllMyBuckets", "s3:GetBucketLocation" ],  
            "Effect": "Allow",  
            "Resource": [ "arn:aws:s3:::*" ]  
        },  
        {  
            "Sid": "AllowRootLevelListingOfCompanyBucket",  
            "Action": [ "s3>ListBucket" ],  
            "Effect": "Allow",  
            "Resource": [ "arn:aws:s3:::companybucket" ],  
            "Condition":{  
                "StringEquals":{  
                    "s3:prefix":[""], "s3:delimiter":["/"]  
                }  
            }  
        }  
    ]  
}
```

3. Test the updated permissions.
 - a. Using the IAM user sign-in link (see [To provide a sign-in link for IAM users \(p. 502\)](#)), sign in to the AWS Management Console.

Open the Amazon S3 console at <https://console.aws.amazon.com/s3/>.
 - b. Choose the bucket that you created, and the console shows the root-level bucket items. If you choose any folders in the bucket, you won't be able to see the folder content because you haven't yet granted those permissions.



This test succeeds when users use the Amazon S3 console. When you choose a bucket on the console, the console implementation sends a request that includes the `prefix` parameter with an empty string as its value and the `delimiter` parameter with "/" as its value.

Step 4.3: Summary of the group policy

The net effect of the group policy that you added is to grant the IAM users Alice and Bob the following minimum permissions:

- List all buckets owned by the parent account.
- See root-level items in the `companybucket` bucket.

However, the users still can't do much. Next, you grant user-specific permissions, as follows:

- Allow Alice to get and put objects in the `Development` folder.
- Allow Bob to get and put objects in the `Finance` folder.

For user-specific permissions, you attach a policy to the specific user, not to the group. In the following section, you grant Alice permission to work in the `Development` folder. You can repeat the steps to grant similar permission to Bob to work in the `Finance` folder.

Step 5: Grant IAM user Alice specific permissions

Now you grant additional permissions to Alice so that she can see the content of the `Development` folder and get and put objects in that folder.

Step 5.1: Grant IAM user Alice permission to list the development folder content

For Alice to list the `Development` folder content, you must apply a policy to the Alice user that grants permission for the `s3:ListBucket` action on the `companybucket` bucket, provided the request includes the prefix `Development/`. You want this policy to be applied only to the user Alice, so you use an inline policy. For more information about inline policies, see [Managed Policies and Inline Policies](#) in the *IAM User Guide*.

1. Sign in to the AWS Management Console and open the IAM console at <https://console.aws.amazon.com/iam/>.

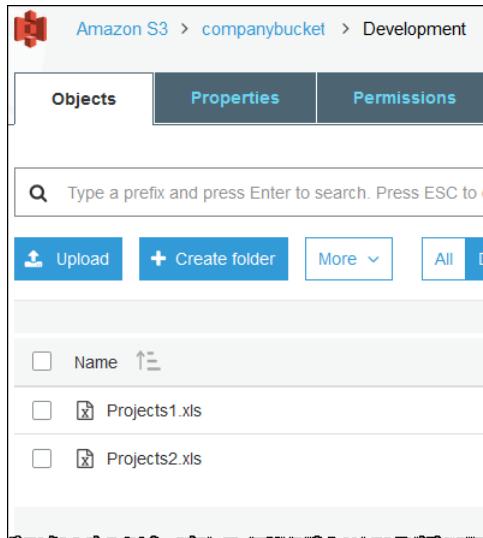
Use your AWS account credentials, not the credentials of an IAM user, to sign in to the console.

2. Create an inline policy to grant the user Alice permission to list the Development folder content.
 - a. In the navigation pane on the left, choose **Users**.
 - b. Choose the user name **Alice**.
 - c. On the user details page, choose the **Permissions** tab and then choose **Add inline policy**.
 - d. Choose the **JSON** tab.
 - e. Copy the following policy and paste it into the policy text field.

```
{  
    "Version": "2012-10-17",  
    "Statement": [  
        {  
            "Sid": "AllowListBucketIfSpecificPrefixIsIncludedInRequest",  
            "Action": ["s3>ListBucket"],  
            "Effect": "Allow",  
            "Resource": ["arn:aws:s3:::companybucket"],  
            "Condition": {"StringLike": {"s3:prefix": ["Development/*"]}}  
        }  
    ]  
}
```

- f. Choose **Review Policy**. On the next page, enter a name in the **Name** field, and then choose **Create policy**.
3. Test the change to Alice's permissions:
 - a. Using the IAM user sign-in link (see [To provide a sign-in link for IAM users \(p. 502\)](#)), sign in to the AWS Management Console.
 - b. Open the Amazon S3 console at <https://console.aws.amazon.com/s3/>.
 - c. On the Amazon S3 console, verify that Alice can see the list of objects in the `Development/` folder in the bucket.

When the user chooses the `/Development` folder to see the list of objects in it, the Amazon S3 console sends the `ListObjects` request to Amazon S3 with the prefix `/Development`. Because the user is granted permission to see the object list with the prefix `Development` and delimiter `/`, Amazon S3 returns the list of objects with the key prefix `Development/`, and the console displays the list.



Step 5.2: Grant IAM user Alice permissions to get and put objects in the development folder

For Alice to get and put objects in the Development folder, she needs permission to call the `s3:GetObject` and `s3:PutObject` actions. The following policy statements grant these permissions, provided that the request includes the `prefix` parameter with a value of `Development/`.

```
{
  "Sid": "AllowUserToReadWriteObjectData",
  "Action": ["s3:GetObject", "s3:PutObject"],
  "Effect": "Allow",
  "Resource": ["arn:aws:s3:::companybucket/Development/*"]
}
```

1. Sign in to the AWS Management Console and open the Amazon S3 console at <https://console.aws.amazon.com/s3/>.

Use your AWS account credentials, not the credentials of an IAM user, to sign in to the console.

2. Edit the inline policy that you created in the previous step.
 - a. In the navigation pane on the left, choose **Users**.
 - b. Choose the user name Alice.
 - c. On the user details page, choose the **Permissions** tab and expand the **Inline Policies** section.
 - d. Next to the name of the policy that you created in the previous step, choose **Edit Policy**.
 - e. Copy the following policy and paste it into the policy text field, replacing the existing policy.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "AllowListBucketIfSpecificPrefixIsIncludedInRequest",
      "Action": ["s3>ListBucket"],
      "Effect": "Allow",
      "Resource": ["arn:aws:s3:::companybucket"],
      "Condition": {
        "StringLike": {"s3:prefix": ["Development/*"]}
      }
    }
  ]
}
```

```

        },
        {
            "Sid": "AllowUserToReadWriteObjectDataInDevelopmentFolder",
            "Action": ["s3:GetObject", "s3:PutObject"],
            "Effect": "Allow",
            "Resource": ["arn:aws:s3:::companybucket/Development/*"]
        }
    ]
}

```

3. Test the updated policy:

- Using the IAM user sign-in link (see [To provide a sign-in link for IAM users \(p. 502\)](#)), sign into the AWS Management Console.
- Open the Amazon S3 console at <https://console.aws.amazon.com/s3/>.
- On the Amazon S3 console, verify that Alice can now add an object and download an object in the Development folder.

Step 5.3: Explicitly deny IAM user Alice permissions to any other folders in the bucket

User Alice can now list the root-level content in the companybucket bucket. She can also get and put objects in the Development folder. If you really want to tighten the access permissions, you could explicitly deny Alice access to any other folders in the bucket. If there is any other policy (bucket policy or ACL) that grants Alice access to any other folders in the bucket, this explicit deny overrides those permissions.

You can add the following statement to the user Alice policy that requires all requests that Alice sends to Amazon S3 to include the `prefix` parameter, whose value can be either `Development/*` or an empty string.

```
{
    "Sid": "ExplicitlyDenyAnyRequestsForAllOtherFoldersExceptDevelopment",
    "Action": ["s3>ListBucket"],
    "Effect": "Deny",
    "Resource": ["arn:aws:s3:::companybucket"],
    "Condition": {
        "StringNotLike": {"s3:prefix": ["Development/*", ""]},
        "Null": {"s3:prefix": false}
    }
}
```

There are two conditional expressions in the `Condition` block. The result of these conditional expressions is combined by using the logical AND. If both conditions are true, the result of the combined condition is true. Because the `Effect` in this policy is `Deny`, when the `Condition` evaluates to true, users can't perform the specified `Action`.

- The `Null` conditional expression ensures that requests from Alice include the `prefix` parameter.

The `prefix` parameter requires folder-like access. If you send a request without the `prefix` parameter, Amazon S3 returns all the object keys.

If the request includes the `prefix` parameter with a null value, the expression evaluates to true, and so the entire `Condition` evaluates to true. You must allow an empty string as value of the `prefix` parameter. From the preceding discussion, recall that allowing the null string allows Alice to retrieve root-level bucket items as the console does in the preceding discussion. For more information, see [Step 4.2: Enable users to list root-level content of a bucket \(p. 505\)](#).

- The `StringNotLike` conditional expression ensures that if the value of the `prefix` parameter is specified and is not `Development/*`, the request fails.

Follow the steps in the preceding section and again update the inline policy that you created for user Alice.

Copy the following policy and paste it into the policy text field, replacing the existing policy.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "AllowListBucketIfSpecificPrefixIsIncludedInRequest",
            "Action": ["s3>ListBucket"],
            "Effect": "Allow",
            "Resource": ["arn:aws:s3:::companybucket"],
            "Condition": {
                "StringLike": {"s3:prefix": ["Development/*"]}
            }
        },
        {
            "Sid": "AllowUserToReadWriteObjectDataInDevelopmentFolder",
            "Action": ["s3.GetObject", "s3:PutObject"],
            "Effect": "Allow",
            "Resource": ["arn:aws:s3:::companybucket/Development/*"]
        },
        {
            "Sid": "ExplicitlyDenyAnyRequestsForAllOtherFoldersExceptDevelopment",
            "Action": ["s3>ListBucket"],
            "Effect": "Deny",
            "Resource": ["arn:aws:s3:::companybucket"],
            "Condition": {
                "StringNotLike": {"s3:prefix": ["Development/*", ""]},
                "Null": {"s3:prefix": false}
            }
        }
    ]
}
```

Step 6: Grant IAM user Bob specific permissions

Now you want to grant Bob permission to the `Finance` folder. Follow the steps that you used earlier to grant permissions to Alice, but replace the `Development` folder with the `Finance` folder. For step-by-step instructions, see [Step 5: Grant IAM user Alice specific permissions \(p. 510\)](#).

Step 7: Secure the private folder

In this example, you have only two users. You granted all the minimum required permissions at the group level and granted user-level permissions only when you really need to permissions at the individual user level. This approach helps minimize the effort of managing permissions. As the number of users increases, managing permissions can become cumbersome. For example, you don't want any of the users in this example to access the content of the `Private` folder. How do you ensure that you don't accidentally grant a user permission to it? You add a policy that explicitly denies access to the folder. An explicit deny overrides any other permissions.

To ensure that the `Private` folder remains private, you can add the following two deny statements to the group policy:

- Add the following statement to explicitly deny any action on resources in the `Private` folder (`companybucket/Private/*`).

```
{
    "Sid": "ExplicitDenyAccessToPrivateFolderToEveryoneInTheGroup",
    "Action": ["s3:*"],
    "Effect": "Deny",
```

```
        "Resource": ["arn:aws:s3:::companybucket/Private/*"]
    }
```

- You also deny permission for the list objects action when the request specifies the `Private/` prefix. On the console, if Bob or Alice opens the `Private` folder, this policy causes Amazon S3 to return an error response.

```
{
    "Sid": "DenyListBucketOnPrivateFolder",
    "Action": ["s3>ListBucket"],
    "Effect": "Deny",
    "Resource": ["arn:aws:s3:::*"],
    "Condition": {
        "StringLike": {"s3:prefix": ["Private/"]}
    }
}
```

Replace the `Consultants` group policy with an updated policy that includes the preceding deny statements. After the updated policy is applied, none of the users in the group can access the `Private` folder in your bucket.

1. Sign in to the AWS Management Console and open the Amazon S3 console at <https://console.aws.amazon.com/s3/>.

Use your AWS account credentials, not the credentials of an IAM user, to sign in to the console.

2. Replace the existing `AllowGroupToSeeBucketListInTheConsole` managed policy that is attached to the `Consultants` group with the following policy. Remember to replace `companybucket` in the policy with the name of your bucket.

For instructions, see [Editing Customer Managed Policies](#) in the *IAM User Guide*. When following the instructions, make sure to follow the directions for applying your changes to all principal entities that the policy is attached to.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "AllowGroupToSeeBucketListAndAlsoAllowGetBucketLocationRequiredForListBucket",
            "Action": ["s3>ListAllMyBuckets", "s3:GetBucketLocation"],
            "Effect": "Allow",
            "Resource": ["arn:aws:s3:::*"]
        },
        {
            "Sid": "AllowRootLevelListingOfCompanyBucket",
            "Action": ["s3>ListBucket"],
            "Effect": "Allow",
            "Resource": ["arn:aws:s3:::companybucket"],
            "Condition": {
                "StringEquals": {"s3:prefix": [""]"}
            }
        },
        {
            "Sid": "RequireFolderPathStyleList",
            "Action": ["s3>ListBucket"],
            "Effect": "Deny",
            "Resource": ["arn:aws:s3:::*"],
            "Condition": {
                "StringNotEquals": {"s3:delimiter": "/"}
            }
        }
    ]
}
```

```
{  
    "Sid": "ExplicitDenyAccessToPrivateFolderToEveryoneInTheGroup",  
    "Action": ["s3:*"],  
    "Effect": "Deny",  
    "Resource": ["arn:aws:s3:::companybucket/Private/*"]  
},  
{  
    "Sid": "DenyListBucketOnPrivateFolder",  
    "Action": ["s3>ListBucket"],  
    "Effect": "Deny",  
    "Resource": ["arn:aws:s3:::*"],  
    "Condition":{  
        "StringLike": {"s3:prefix": ["Private/"]} }  
}  
]  
}
```

Step 8: Clean up

To clean up, open the IAM console and remove the users Alice and Bob. For step-by-step instructions, see [Deleting an IAM User in the IAM User Guide](#).

To ensure that you aren't charged further for storage, you should also delete the objects and the bucket that you created for this exercise.

Related resources

- [Managing IAM Policies](#) in the *IAM User Guide*.

User policy examples

This section shows several example AWS Identity and Access Management (IAM) user policies for controlling user access to Amazon S3. For example *bucket policies*, see [Using bucket policies \(p. 487\)](#). For information about IAM policy language, see [Bucket policies and user policies \(p. 411\)](#).

The following example policies will work if you use them programmatically. However, to use them with the Amazon S3 console, you must grant additional permissions that are required by the console. For information about using policies such as these with the Amazon S3 console, see [Controlling access to a bucket with user policies \(p. 499\)](#).

Topics

- [Allowing an IAM user access to one of your buckets \(p. 516\)](#)
- [Allowing each IAM user access to a folder in a bucket \(p. 517\)](#)
- [Allowing a group to have a shared folder in Amazon S3 \(p. 520\)](#)
- [Allowing all your users to read objects in a portion of a bucket \(p. 520\)](#)
- [Allowing a partner to drop files into a specific portion of a bucket \(p. 520\)](#)
- [Restricting access to Amazon S3 buckets within a specific AWS account \(p. 521\)](#)
- [Restricting access to Amazon S3 buckets within your organizational unit \(OU\) \(p. 522\)](#)
- [Restricting access to Amazon S3 buckets within your organization \(p. 522\)](#)

Allowing an IAM user access to one of your buckets

In this example, you want to grant an IAM user in your AWS account access to one of your buckets, **DOC-EXAMPLE-BUCKET1**, and allow the user to add, update, and delete objects.

In addition to granting the `s3:PutObject`, `s3:GetObject`, and `s3:DeleteObject` permissions to the user, the policy also grants the `s3>ListAllMyBuckets`, `s3:GetBucketLocation`, and `s3>ListBucket` permissions. These are the additional permissions required by the console. Also, the `s3:PutObjectAcl` and the `s3:GetObjectAcl` actions are required to be able to copy, cut, and paste objects in the console. For an example walkthrough that grants permissions to users and tests them using the console, see [Controlling access to a bucket with user policies \(p. 499\)](#).

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": "s3>ListAllMyBuckets",
            "Resource": "*"
        },
        {
            "Effect": "Allow",
            "Action": ["s3>ListBucket", "s3:GetBucketLocation"],
            "Resource": "arn:aws:s3:::DOC-EXAMPLE-BUCKET1"
        },
        {
            "Effect": "Allow",
            "Action": [
                "s3:PutObject",
                "s3:PutObjectAcl",
                "s3:GetObject",
                "s3:GetObjectAcl",
                "s3>DeleteObject"
            ],
            "Resource": "arn:aws:s3:::DOC-EXAMPLE-BUCKET1/*"
        }
    ]
}
```

Allowing each IAM user access to a folder in a bucket

In this example, you want two IAM users, Mary and Carlos, to have access to your bucket, **DOC-EXAMPLE-BUCKET1**, so that they can add, update, and delete objects. However, you want to restrict each user's access to a single prefix (folder) in the bucket. You might create folders with names that match their user names.

```
DOC-EXAMPLE-BUCKET1
Mary/
Carlos/
```

To grant each user access only to their folder, you can write a policy for each user and attach it individually. For example, you can attach the following policy to the user Mary to allow her specific Amazon S3 permissions on the **DOC-EXAMPLE-BUCKET1/Mary** folder.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "s3:PutObject",
                "s3:GetObject",
                "s3:GetObjectVersion",
                "s3>DeleteObject",
                "s3:DeleteObjectVersion"
            ],
            "Resource": "arn:aws:s3:::DOC-EXAMPLE-BUCKET1/Mary/*"
        }
    ]
}
```

```

        "Resource": "arn:aws:s3:::DOC-EXAMPLE-BUCKET1/Mary/*"
    ]
}

```

You can then attach a similar policy to the user Carlos, specifying the folder *Carlos* in the Resource value.

Instead of attaching policies to individual users, you can write a single policy that uses a policy variable and then attach the policy to a group. First, you must create a group and add both Mary and Carlos to the group. The following example policy allows a set of Amazon S3 permissions in the *DOC-EXAMPLE-BUCKET1*/\${aws:username} folder. When the policy is evaluated, the policy variable \${aws:username} is replaced by the requester's user name. For example, if Mary sends a request to put an object, the operation is allowed only if Mary is uploading the object to the *DOC-EXAMPLE-BUCKET1*/Mary folder.

```

{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "s3:PutObject",
                "s3:GetObject",
                "s3:GetObjectVersion",
                "s3:DeleteObject",
                "s3:DeleteObjectVersion"
            ],
            "Resource": "arn:aws:s3:::DOC-EXAMPLE-BUCKET1/${aws:username}/*"
        }
    ]
}

```

Note

When using policy variables, you must explicitly specify version 2012-10-17 in the policy. The default version of the IAM policy language, 2008-10-17, does not support policy variables.

If you want to test the preceding policy on the Amazon S3 console, the console requires additional permissions, as shown in the following policy. For information about how the console uses these permissions, see [Controlling access to a bucket with user policies \(p. 499\)](#).

```

{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "AllowGroupToSeeBucketListInTheConsole",
            "Action": [
                "s3>ListAllMyBuckets",
                "s3:GetBucketLocation"
            ],
            "Effect": "Allow",
            "Resource": "arn:aws:s3:::/*"
        },
        {
            "Sid": "AllowRootLevelListingOfTheBucket",
            "Action": "s3>ListBucket",
            "Effect": "Allow",
            "Resource": "arn:aws:s3:::DOC-EXAMPLE-BUCKET1",
            "Condition": {
                "StringEquals": {
                    "s3:prefix": [""], "s3:delimiter": ["/"]
                }
            }
        }
    ]
}

```

```

        }
    },
    {
        "Sid": "AllowListBucketOfASpecificUserPrefix",
        "Action": "s3>ListBucket",
        "Effect": "Allow",
        "Resource": "arn:aws:s3:::DOC-EXAMPLE-BUCKET1",
        "Condition":{ "StringLike":{ "s3:prefix":["${aws:username}/*"] } }
    },
    {
        "Sid": "AllowUserSpecificActionsOnlyInTheSpecificUserPrefix",
        "Effect": "Allow",
        "Action": [
            "s3:PutObject",
            "s3:GetObject",
            "s3:GetObjectVersion",
            "s3>DeleteObject",
            "s3>DeleteObjectVersion"
        ],
        "Resource": "arn:aws:s3:::DOC-EXAMPLE-BUCKET1/${aws:username}/*"
    }
]
}

```

Note

In the 2012-10-17 version of the policy, policy variables start with \$. This change in syntax can potentially create a conflict if your object key (object name) includes a \$.

To avoid this conflict, specify the \$ character by using \${\$}. For example, to include the object key my\$file in a policy, specify it as my\${\$}file.

Although IAM user names are friendly, human-readable identifiers, they aren't required to be globally unique. For example, if the user Carlos leaves the organization and another Carlos joins, then the new Carlos could access the old Carlos's information.

Instead of using user names, you could create folders based on IAM user IDs. Each IAM user ID is unique. In this case, you must modify the preceding policy to use the \${aws:userid} policy variable. For more information about user identifiers, see [IAM Identifiers](#) in the *IAM User Guide*.

```

{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "s3:PutObject",
                "s3:GetObject",
                "s3:GetObjectVersion",
                "s3>DeleteObject",
                "s3>DeleteObjectVersion"
            ],
            "Resource": "arn:aws:s3:::DOC-EXAMPLE-BUCKET1/home/${aws:userid}/*"
        }
    ]
}

```

[Allowing non-IAM users \(mobile app users\) access to folders in a bucket](#)

Suppose that you want to develop a mobile app, a game that stores users' data in an S3 bucket. For each app user, you want to create a folder in your bucket. You also want to limit each user's access to their own folder. But you cannot create folders before someone downloads your app and starts playing the game, because you don't have their user ID.

In this case, you can require users to sign in to your app by using public identity providers such as Login with Amazon, Facebook, or Google. After users have signed in to your app through one of these providers, they have a user ID that you can use to create user-specific folders at runtime.

You can then use web identity federation in AWS Security Token Service to integrate information from the identity provider with your app and to get temporary security credentials for each user. You can then create IAM policies that allow the app to access your bucket and perform such operations as creating user-specific folders and uploading data. For more information about web identity federation, see [About web identity Federation](#) in the *IAM User Guide*.

Allowing a group to have a shared folder in Amazon S3

Attaching the following policy to the group grants everybody in the group access to the following folder in Amazon S3: *DOC-EXAMPLE-BUCKET1/share/marketing*. Group members are allowed to access only the specific Amazon S3 permissions shown in the policy and only for objects in the specified folder.

```
{  
    "Version": "2012-10-17",  
    "Statement": [  
        {  
            "Effect": "Allow",  
            "Action": [  
                "s3:PutObject",  
                "s3:GetObject",  
                "s3:GetObjectVersion",  
                "s3:DeleteObject",  
                "s3:DeleteObjectVersion"  
            ],  
            "Resource": "arn:aws:s3:::DOC-EXAMPLE-BUCKET1/share/marketing/*"  
        }  
    ]  
}
```

Allowing all your users to read objects in a portion of a bucket

In this example, you create a group named *AllUsers*, which contains all the IAM users that are owned by the AWS account. You then attach a policy that gives the group access to *GetObject* and *GetObjectVersion*, but only for objects in the *DOC-EXAMPLE-BUCKET1/readonly* folder.

```
{  
    "Version": "2012-10-17",  
    "Statement": [  
        {  
            "Effect": "Allow",  
            "Action": [  
                "s3:GetObject",  
                "s3:GetObjectVersion"  
            ],  
            "Resource": "arn:aws:s3:::DOC-EXAMPLE-BUCKET1/readonly/*"  
        }  
    ]  
}
```

Allowing a partner to drop files into a specific portion of a bucket

In this example, you create a group called *AnyCompany* that represents a partner company. You create an IAM user for the specific person or application at the partner company that needs access, and then you put the user in the group.

You then attach a policy that gives the group *PutObject* access to the following folder in a bucket:

DOC-EXAMPLE-BUCKET1/uploads/anycompany

You want to prevent the *AnyCompany* group from doing anything else with the bucket, so you add a statement that explicitly denies permission to any Amazon S3 actions except PutObject on any Amazon S3 resource in the AWS account.

```
{  
    "Version": "2012-10-17",  
    "Statement": [  
        {  
            "Effect": "Allow",  
            "Action": "s3:PutObject",  
            "Resource": "arn:aws:s3:::DOC-EXAMPLE-BUCKET1/uploads/anycompany/*"  
        },  
        {  
            "Effect": "Deny",  
            "Action": "s3:*",  
            "NotResource": "arn:aws:s3:::DOC-EXAMPLE-BUCKET1/uploads/anycompany/*"  
        }  
    ]  
}
```

Restricting access to Amazon S3 buckets within a specific AWS account

If you want to ensure that your Amazon S3 principals are accessing only the resources that are inside of a trusted AWS account, you can restrict access. For example, this [identity-based IAM policy](#) uses a Deny effect to block access to Amazon S3 actions, unless the Amazon S3 resource that's being accessed is in account *222222222222*. To prevent an IAM principal in an AWS account from accessing Amazon S3 objects outside of the account, attach the following IAM policy:

```
{  
    "Version": "2012-10-17",  
    "Statement": [  
        {  
            "Sid": "DenyS3AccessOutsideMyBoundary",  
            "Effect": "Deny",  
            "Action": [  
                "s3:*"  
            ],  
            "Resource": "*",  
            "Condition": {  
                "StringNotEquals": {  
                    "aws:ResourceAccount": [  
                        "222222222222"  
                    ]  
                }  
            }  
        }  
    ]  
}
```

Note

This policy does not replace your existing IAM access controls, because it doesn't grant any access. Instead, this policy acts as an additional guardrail for your other IAM permissions, regardless of the permissions granted through other IAM policies.

Make sure to replace account ID *222222222222* in the policy with your own AWS account. To apply a policy to multiple accounts while still maintaining this restriction, replace the account ID with the `aws:PrincipalAccount` condition key. This condition requires that the principal and the resource must be in the same account.

Restricting access to Amazon S3 buckets within your organizational unit (OU)

If you have an [organizational unit \(OU\)](#) set up in AWS Organizations, you might want to restrict Amazon S3 bucket access to a specific part of your organization. In this example, we'll use the `aws:ResourceOrgPaths` key to restrict Amazon S3 bucket access to an OU in your organization. For this example, the [OU ID](#) is `ou-acroot-exampleou`. Make sure to replace this value in your own policy with your own OU IDs.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "AllowS3AccessOutsideMyBoundary",
            "Effect": "Allow",
            "Action": [
                "s3:*"
            ],
            "Resource": "*",
            "Condition": {
                "ForAllValues:StringNotLike": {
                    "aws:ResourceOrgPaths": [
                        "o-acorg/r-acroot/ou-acroot-exampleou/"
                    ]
                }
            }
        }
    ]
}
```

Note

This policy does not grant any access. Instead, this policy acts as a backstop for your other IAM permissions, preventing your principals from accessing Amazon S3 objects outside of an OU-defined boundary.

The policy denies access to Amazon S3 actions unless the Amazon S3 object that's being accessed is in the `ou-acroot-exampleou` OU in your organization. The [IAM policy condition](#) requires `aws:ResourceOrgPaths`, a multivalued condition key, to contain any of the listed OU paths. The policy uses the `ForAllValues:StringNotLike` operator to compare the values of `aws:ResourceOrgPaths` to the listed OUs without case-sensitive matching.

Restricting access to Amazon S3 buckets within your organization

To restrict access to Amazon S3 objects within your organization, attach an IAM policy to the root of the organization, applying it to all accounts in your organization. To require your IAM principals to follow this rule, use a [service-control policy \(SCP\)](#). If you choose to use an SCP, make sure to thoroughly [test the SCP](#) before attaching the policy to the root of the organization.

In the following example policy, access is denied to Amazon S3 actions unless the Amazon S3 object that's being accessed is in the same organization as the IAM principal that is accessing it:

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "DenyS3AccessOutsideMyBoundary",
            "Effect": "Deny",
            "Action": [
                "s3:*"
            ],
            "Resource": "arn:aws:s3:::/*",
            "Condition": {
                "StringNotLike": {
                    "aws:ResourceOrgPaths": [
                        "o-acorg/r-acroot/ou-acroot-exampleou/"
                    ]
                }
            }
        }
    ]
}
```

```
        "StringNotEquals": {
            "aws:ResourceOrgID": "${aws:PrincipalOrgID}"
        }
    }
}
```

Note

This policy does not grant any access. Instead, this policy acts as a backstop for your other IAM permissions, preventing your principals from accessing any Amazon S3 objects outside of your organization. This policy also applies to Amazon S3 resources that are created after the policy is put into effect.

The [IAM policy condition](#) in this example requires `aws:ResourceOrgID` and `aws:PrincipalOrgID` to be equal to each other. With this requirement, the principal making the request and the resource being accessed must be in the same organization.

Example walkthroughs: Managing access to your Amazon S3 resources

This topic provides the following introductory walkthrough examples for granting access to Amazon S3 resources. These examples use the AWS Management Console to create resources (buckets, objects, users) and grant them permissions. The examples then show you how to verify permissions using the command line tools, so you don't have to write any code. We provide commands using both the AWS Command Line Interface (CLI) and the AWS Tools for Windows PowerShell.

- [Example 1: Bucket owner granting its users bucket permissions \(p. 527\)](#)

The IAM users you create in your account have no permissions by default. In this exercise, you grant a user permission to perform bucket and object operations.

- [Example 2: Bucket owner granting cross-account bucket permissions \(p. 531\)](#)

In this exercise, a bucket owner, Account A, grants cross-account permissions to another AWS account, Account B. Account B then delegates those permissions to users in its account.

- **Managing object permissions when the object and bucket owners are not the same**

The example scenarios in this case are about a bucket owner granting object permissions to others, but not all objects in the bucket are owned by the bucket owner. What permissions does the bucket owner need, and how can it delegate those permissions?

The AWS account that creates a bucket is called the bucket owner. The owner can grant other AWS accounts permission to upload objects, and the AWS accounts that create objects own them. The bucket owner has no permissions on those objects created by other AWS accounts. If the bucket owner writes a bucket policy granting access to objects, the policy does not apply to objects that are owned by other accounts.

In this case, the object owner must first grant permissions to the bucket owner using an object ACL. The bucket owner can then delegate those object permissions to others, to users in its own account, or to another AWS account, as illustrated by the following examples.

- [Example 3: Bucket owner granting permissions to objects it does not own \(p. 536\)](#)

In this exercise, the bucket owner first gets permissions from the object owner. The bucket owner then delegates those permissions to users in its own account.

- [Example 4: Bucket owner granting cross-account permission to objects it does not own \(p. 541\)](#)

After receiving permissions from the object owner, the bucket owner cannot delegate permission to other AWS accounts because cross-account delegation is not supported (see [Permission delegation \(p. 403\)](#)). Instead, the bucket owner can create an IAM role with permissions to perform specific operations (such as get object) and allow another AWS account to assume that role. Anyone who assumes the role can then access objects. This example shows how a bucket owner can use an IAM role to enable this cross-account delegation.

Before you try the example walkthroughs

These examples use the AWS Management Console to create resources and grant permissions. And to test permissions, the examples use the command line tools, AWS Command Line Interface (CLI) and AWS Tools for Windows PowerShell, so you don't need to write any code. To test permissions you will need to set up one of these tools. For more information, see [Setting up the tools for the example walkthroughs \(p. 525\)](#).

In addition, when creating resources these examples don't use root credentials of an AWS account. Instead, you create an administrator user in these accounts to perform these tasks.

About using an administrator user to create resources and grant permissions

AWS Identity and Access Management (IAM) recommends not using the root credentials of your AWS account to make requests. Instead, create an IAM user, grant that user full access, and then use that user's credentials to make requests. We refer to this user as an administrator user. For more information, go to [Root Account Credentials vs. IAM User Credentials](#) in the *AWS General Reference* and [IAM Best Practices](#) in the *IAM User Guide*.

All example walkthroughs in this section use the administrator user credentials. If you have not created an administrator user for your AWS account, the topics show you how.

Note that to sign in to the AWS Management Console using the user credentials, you will need to use the IAM User Sign-In URL. The IAM console provides this URL for your AWS account. The topics show you how to get the URL.

Setting up the tools for the example walkthroughs

The introductory examples (see [Example walkthroughs: Managing access to your Amazon S3 resources \(p. 524\)](#)) use the AWS Management Console to create resources and grant permissions. And to test permissions, the examples use the command line tools, AWS Command Line Interface (CLI) and AWS Tools for Windows PowerShell, so you don't need to write any code. To test permissions, you must set up one of these tools.

To set up the AWS CLI

1. Download and configure the AWS CLI. For instructions, see the following topics in the *AWS Command Line Interface User Guide*.

[Getting Set Up with the AWS Command Line Interface](#)

[Installing the AWS Command Line Interface](#)

[Configuring the AWS Command Line Interface](#)

2. Set the default profile.

You will store user credentials in the AWS CLI config file. Create a default profile in the config file using your AWS account credentials. See [Configuration and Credential Files](#) for instructions on finding and editing your AWS CLI config file.

```
[default]
aws_access_key_id = access key ID
aws_secret_access_key = secret access key
region = us-west-2
```

3. Verify the setup by entering the following command at the command prompt. Both these commands don't provide credentials explicitly, so the credentials of the default profile are used.

- Try the help command

```
aws help
```

- Use aws s3 ls to get a list of buckets on the configured account.

```
aws s3 ls
```

As you go through the example walkthroughs, you will create users, and you will save user credentials in the config files by creating profiles, as the following example shows. Note that these profiles have names (AccountAdmin and AccountBadmin):

```
[profile AccountAadmin]
aws_access_key_id = User AccountAadmin access key ID
aws_secret_access_key = User AccountAadmin secret access key
region = us-west-2

[profile AccountBadmin]
aws_access_key_id = Account B access key ID
aws_secret_access_key = Account B secret access key
region = us-east-1
```

To run a command using these user credentials, you add the `--profile` parameter specifying the profile name. The following AWS CLI command retrieves a listing of objects in `examplebucket` and specifies the `AccountBadmin` profile.

```
aws s3 ls s3://examplebucket --profile AccountBadmin
```

Alternatively, you can configure one set of user credentials as the default profile by changing the `AWS_DEFAULT_PROFILE` environment variable from the command prompt. Once you've done this, whenever you perform AWS CLI commands without the `--profile` parameter, the AWS CLI will use the profile you set in the environment variable as the default profile.

```
$ export AWS_DEFAULT_PROFILE=AccountAadmin
```

To set up AWS Tools for Windows PowerShell

1. Download and configure the AWS Tools for Windows PowerShell. For instructions, go to [Download and Install the AWS Tools for Windows PowerShell](#) in the *AWS Tools for Windows PowerShell User Guide*.

Note

In order to load the AWS Tools for Windows PowerShell module, you need to enable PowerShell script execution. For more information, go to [Enable Script Execution](#) in the *AWS Tools for Windows PowerShell User Guide*.

2. For these exercises, you will specify AWS credentials per session using the `Set-AWSCredentials` command. The command saves the credentials to a persistent store (`-StoreAs` parameter).

```
Set-AWSCredentials -AccessKey AccessKeyId -SecretKey SecretAccessKey -storeas string
```

3. Verify the setup.

- Run the `Get-Command` to retrieve a list of available commands you can use for Amazon S3 operations.

```
Get-Command -module awspowershell -noun s3* -StoredCredentials string
```

- Run the `Get-S3Object` command to retrieve a list of objects in a bucket.

```
Get-S3Object -BucketName bucketname -StoredCredentials string
```

For a list of commands, go to [Amazon Simple Storage Service Cmdlets](#).

Now you are ready to try the exercises. Follow the links provided at the beginning of the section.

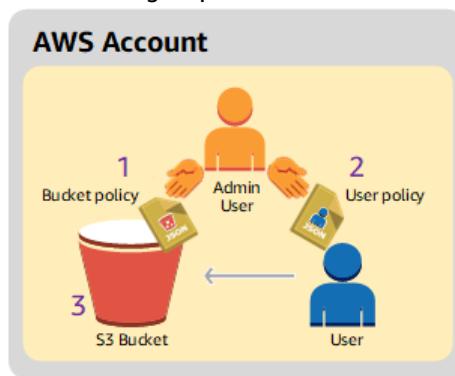
Example 1: Bucket owner granting its users bucket permissions

Topics

- Step 0: Preparing for the walkthrough (p. 527)
- Step 1: Create resources (a bucket and an IAM user) in account a and grant permissions (p. 528)
- Step 2: Test permissions (p. 530)

In this exercise, an AWS account owns a bucket, and it has an IAM user in the account. By default, the user has no permissions. For the user to perform any tasks, the parent account must grant them permissions. The bucket owner and parent account are the same. Therefore, to grant the user permissions on the bucket, the AWS account can use a bucket policy, a user policy, or both. The account owner will grant some permissions using a bucket policy and other permissions using a user policy.

The following steps summarize the walkthrough:



1. Account administrator creates a bucket policy granting a set of permissions to the user.
2. Account administrator attaches a user policy to the user granting additional permissions.
3. User then tries permissions granted via both the bucket policy and the user policy.

For this example, you will need an AWS account. Instead of using the root credentials of the account, you will create an administrator user (see [About using an administrator user to create resources and grant permissions \(p. 525\)](#)). We refer to the AWS account and the administrator user as follows:

Account ID	Account referred to as	Administrator user in the account
1111-1111-1111	Account A	AccountAAdmin

Note

The administrator user in this example is **AccountAAdmin**, which refers to Account A, and not **AccountAdmin**.

All the tasks of creating users and granting permissions are done in the AWS Management Console. To verify permissions, the walkthrough uses the command line tools, AWS Command Line Interface (CLI) and AWS Tools for Windows PowerShell, to verify the permissions, so you don't need to write any code.

Step 0: Preparing for the walkthrough

1. Make sure you have an AWS account and that it has a user with administrator privileges.
 - a. Sign up for an account, if needed. We refer to this account as Account A.

- i. Go to <https://aws.amazon.com/s3> and click **Sign Up**.
 - ii. Follow the on-screen instructions.
- AWS will notify you by email when your account is active and available for you to use.
- b. In Account A, create an administrator user AccountAadmin. Using Account A credentials, sign in to the [IAM console](#) and do the following:
- i. Create user AccountAadmin and note down the user security credentials.
For instructions, see [Creating an IAM User in Your AWS Account](#) in the *IAM User Guide*.
 - ii. Grant AccountAadmin administrator privileges by attaching a user policy giving full access.
For instructions, see [Working with Policies](#) in the *IAM User Guide*.
 - iii. Note down the **IAM User Sign-In URL** for AccountAadmin. You will need to use this URL when signing in to the AWS Management Console. For more information about where to find it, see [How Users Sign in to Your Account](#) in *IAM User Guide*. Note down the URL for each of the accounts.
2. Set up either the AWS Command Line Interface (CLI) or the AWS Tools for Windows PowerShell. Make sure you save administrator user credentials as follows:
 - If using the AWS CLI, create a profile, AccountAadmin, in the config file.
 - If using the AWS Tools for Windows PowerShell, make sure you store credentials for the session as AccountAadmin.

For instructions, see [Setting up the tools for the example walkthroughs \(p. 525\)](#).

Step 1: Create resources (a bucket and an IAM user) in account a and grant permissions

Using the credentials of user AccountAadmin in Account A, and the special IAM user sign-in URL, sign in to the AWS Management Console and do the following:

1. Create Resources (a bucket and an IAM user)
 - a. In the Amazon S3 console create a bucket. Note down the AWS Region in which you created it. For instructions, see [Creating a bucket \(p. 119\)](#).
 - b. In the IAM console, do the following:
 - i. Create a user, Dave.
For instructions, see [Creating IAM Users \(AWS Management Console\)](#) in the *IAM User Guide*.
 - ii. Note down the UserDave credentials.
 - iii. Note down the Amazon Resource Name (ARN) for user Dave. In the IAM console, select the user, and the **Summary** tab provides the user ARN.

2. Grant Permissions.

Because the bucket owner and the parent account to which the user belongs are the same, the AWS account can grant user permissions using a bucket policy, a user policy, or both. In this example, you do both. If the object is also owned by the same account, the bucket owner can grant object permissions in the bucket policy (or an IAM policy).

- a. In the Amazon S3 console, attach the following bucket policy to `awsexamplebucket1`.

The policy has two statements.

- The first statement grants Dave the bucket operation permissions `s3:GetBucketLocation` and `s3>ListBucket`.
- The second statement grants the `s3:GetObject` permission. Because Account A also owns the object, the account administrator is able to grant the `s3:GetObject` permission.

In the Principal statement, Dave is identified by his user ARN. For more information about policy elements, see [Bucket policies and user policies \(p. 411\)](#).

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "statement1",
            "Effect": "Allow",
            "Principal": {
                "AWS": "arn:aws:iam::AccountA-ID:user/Dave"
            },
            "Action": [
                "s3:GetBucketLocation",
                "s3>ListBucket"
            ],
            "Resource": [
                "arn:aws:s3:::awsexamplebucket1"
            ]
        },
        {
            "Sid": "statement2",
            "Effect": "Allow",
            "Principal": {
                "AWS": "arn:aws:iam::AccountA-ID:user/Dave"
            },
            "Action": [
                "s3:GetObject"
            ],
            "Resource": [
                "arn:aws:s3:::awsexamplebucket1/*"
            ]
        }
    ]
}
```

- Create an inline policy for the user Dave by using the following policy. The policy grants Dave the `s3:PutObject` permission. You need to update the policy by providing your bucket name.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "PermissionForObjectOperations",
            "Effect": "Allow",
            "Action": [
                "s3:PutObject"
            ],
            "Resource": [
                "arn:aws:s3:::awsexamplebucket1/*"
            ]
        }
    ]
}
```

For instructions, see [Working with Inline Policies](#) in the *IAM User Guide*. Note you need to sign in to the console using Account A credentials.

Step 2: Test permissions

Using Dave's credentials, verify that the permissions work. You can use either of the following two procedures.

Test using the AWS CLI

1. Update the AWS CLI config file by adding the following UserDaveAccountA profile. For more information, see [Setting up the tools for the example walkthroughs \(p. 525\)](#).

```
[profile UserDaveAccountA]
aws_access_key_id = access-key
aws_secret_access_key = secret-access-key
region = us-east-1
```

2. Verify that Dave can perform the operations as granted in the user policy. Upload a sample object using the following AWS CLI put-object command.

The --body parameter in the command identifies the source file to upload. For example, if the file is in the root of the C: drive on a Windows machine, you specify c:\HappyFace.jpg. The --key parameter provides the key name for the object.

```
aws s3api put-object --bucket awsexamplebucket1 --key HappyFace.jpg --
body HappyFace.jpg --profile UserDaveAccountA
```

Run the following AWS CLI command to get the object.

```
aws s3api get-object --bucket awsexamplebucket1 --key HappyFace.jpg OutputFile.jpg --
profile UserDaveAccountA
```

Test using the AWS Tools for Windows PowerShell

1. Store Dave's credentials as AccountADave. You then use these credentials to PUT and GET an object.

```
set-awscredentials -AccessKey AccessKeyID -SecretKey SecretAccessKey -storeas
AccountADave
```

2. Upload a sample object using the AWS Tools for Windows PowerShell Write-S3Object command using user Dave's stored credentials.

```
Write-S3Object -bucketname awsexamplebucket1 -key HappyFace.jpg -file HappyFace.jpg -
StoredCredentials AccountADave
```

Download the previously uploaded object.

```
Read-S3Object -bucketname awsexamplebucket1 -key HappyFace.jpg -file Output.jpg -
StoredCredentials AccountADave
```

Example 2: Bucket owner granting cross-account bucket permissions

Topics

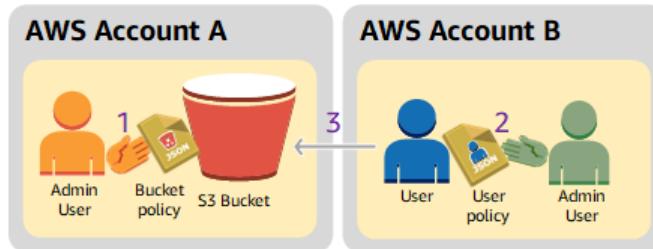
- [Step 0: Preparing for the walkthrough \(p. 532\)](#)
- [Step 1: Do the Account A tasks \(p. 533\)](#)
- [Step 2: Do the Account B tasks \(p. 534\)](#)
- [Step 3: \(Optional\) Try explicit deny \(p. 535\)](#)
- [Step 4: Clean up \(p. 536\)](#)

An AWS account—for example, Account A—can grant another AWS account, Account B, permission to access its resources such as buckets and objects. Account B can then delegate those permissions to users in its account. In this example scenario, a bucket owner grants cross-account permission to another account to perform specific bucket operations.

Note

Account A can also directly grant a user in Account B permissions using a bucket policy. But the user will still need permission from the parent account, Account B, to which the user belongs, even if Account B does not have permissions from Account A. As long as the user has permission from both the resource owner and the parent account, the user will be able to access the resource.

The following is a summary of the walkthrough steps:



1. Account A administrator user attaches a bucket policy granting cross-account permissions to Account B to perform specific bucket operations.

Note that administrator user in Account B will automatically inherit the permissions.

2. Account B administrator user attaches user policy to the user delegating the permissions it received from Account A.
3. User in Account B then verifies permissions by accessing an object in the bucket owned by Account A.

For this example, you need two accounts. The following table shows how we refer to these accounts and the administrator users in them. Per IAM guidelines (see [About using an administrator user to create resources and grant permissions \(p. 525\)](#)) we do not use the account root credentials in this walkthrough. Instead, you create an administrator user in each account and use those credentials in creating resources and granting them permissions.

AWS account ID	Account referred to as	Administrator user in the account
1111-1111-1111	Account A	AccountAadmin
2222-2222-2222	Account B	AccountBadmin

All the tasks of creating users and granting permissions are done in the AWS Management Console. To verify permissions, the walkthrough uses the command line tools, AWS Command Line Interface (CLI) and AWS Tools for Windows PowerShell, so you don't need to write any code.

Step 0: Preparing for the walkthrough

1. Make sure you have two AWS accounts and that each account has one administrator user as shown in the table in the preceding section.
 - a. Sign up for an AWS account, if needed.
 - i. Go to <https://aws.amazon.com/s3/> and click **Create an AWS Account**.
 - ii. Follow the on-screen instructions.AWS will notify you by email when your account is active and available for you to use.
 - b. Using Account A credentials, sign in to the **IAM console** to create the administrator user:
 - i. Create user AccountAadmin and note down the security credentials. For instructions, see [Creating an IAM User in Your AWS account](#) in the *IAM User Guide*.
 - ii. Grant AccountAadmin administrator privileges by attaching a user policy giving full access. For instructions, see [Working with Policies](#) in the *IAM User Guide*.
 - c. While you are in the IAM console, note down the **IAM User Sign-In URL** on the **Dashboard**. All users in the account must use this URL when signing in to the AWS Management Console.

For more information, see [How Users Sign in to Your Account](#) in *IAM User Guide*.
 - d. Repeat the preceding step using Account B credentials and create administrator user AccountBadmin.
2. Set up either the AWS Command Line Interface (CLI) or the AWS Tools for Windows PowerShell. Make sure you save administrator user credentials as follows:
 - If using the AWS CLI, create two profiles, AccountAadmin and AccountBadmin, in the config file.
 - If using the AWS Tools for Windows PowerShell, make sure you store credentials for the session as AccountAadmin and AccountBadmin.

For instructions, see [Setting up the tools for the example walkthroughs \(p. 525\)](#).

3. Save the administrator user credentials, also referred to as profiles. You can use the profile name instead of specifying credentials for each command you enter. For more information, see [Setting up the tools for the example walkthroughs \(p. 525\)](#).
 - a. Add profiles in the AWS CLI credentials file for each of the administrator users in the two accounts.

```
[AccountAadmin]
aws_access_key_id = access-key-ID
aws_secret_access_key = secret-access-key
region = us-east-1

[AccountBadmin]
aws_access_key_id = access-key-ID
aws_secret_access_key = secret-access-key
region = us-east-1
```

3. If you are using the AWS Tools for Windows PowerShell

```
set-awscredentials -AccessKey AcctA-access-key-ID -SecretKey AcctA-secret-access-
key -storeas AccountAadmin
```

```
set-awscredentials -AccessKey AcctB-access-key-ID -SecretKey AcctB-secret-access-key -storeas AccountBadmin
```

Step 1: Do the Account A tasks

Step 1.1: Sign in to the AWS Management Console

Using the IAM user sign-in URL for Account A first sign in to the AWS Management Console as AccountAadmin user. This user will create a bucket and attach a policy to it.

Step 1.2: Create a bucket

1. In the Amazon S3 console, create a bucket. This exercise assumes the bucket is created in the US East (N. Virginia) region and is named **DOC-EXAMPLE-BUCKET**.

For instructions, see [Creating a bucket \(p. 119\)](#).

2. Upload a sample object to the bucket.

For instructions, go to [Step 2: Upload an object to your bucket \(p. 15\)](#).

Step 1.3: Attach a bucket policy to grant cross-account permissions to Account B

The bucket policy grants the `s3:GetLifecycleConfiguration` and `s3>ListBucket` permissions to Account B. It is assumed you are still signed into the console using AccountAadmin user credentials.

1. Attach the following bucket policy to **DOC-EXAMPLE-BUCKET**. The policy grants Account B permission for the `s3:GetLifecycleConfiguration` and `s3>ListBucket` actions.

For instructions, see [Adding a bucket policy using the Amazon S3 console \(p. 488\)](#).

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "Example permissions",
            "Effect": "Allow",
            "Principal": {
                "AWS": "arn:aws:iam::AccountB-ID:root"
            },
            "Action": [
                "s3:GetLifecycleConfiguration",
                "s3>ListBucket"
            ],
            "Resource": [
                "arn:aws:s3:::DOC-EXAMPLE-BUCKET"
            ]
        }
    ]
}
```

2. Verify Account B (and thus its administrator user) can perform the operations.

- Using the AWS CLI

```
aws s3 ls s3://DOC-EXAMPLE-BUCKET --profile AccountBadmin
aws s3api get-bucket-lifecycle-configuration --bucket DOC-EXAMPLE-BUCKET --profile AccountBadmin
```

- Using the AWS Tools for Windows PowerShell

```
get-s3object -BucketName DOC-EXAMPLE-BUCKET -StoredCredentials AccountBadmin
get-s3bucketlifecycleconfiguration -BucketName DOC-EXAMPLE-BUCKET -StoredCredentials
AccountBadmin
```

Step 2: Do the Account B tasks

Now the Account B administrator creates a user, Dave, and delegates the permissions received from Account A.

Step 2.1: Sign in to the AWS Management Console

Using the IAM user sign-in URL for Account B, first sign in to the AWS Management Console as AccountBadmin user.

Step 2.2: Create user Dave in Account B

In the IAM console, create a user, Dave.

For instructions, see [Creating IAM Users \(AWS Management Console\)](#) in the *IAM User Guide*.

Step 2.3: Delegate permissions to user Dave

Create an inline policy for the user Dave by using the following policy. You will need to update the policy by providing your bucket name.

It is assumed you are signed in to the console using AccountBadmin user credentials.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "Example",
            "Effect": "Allow",
            "Action": [
                "s3>ListBucket"
            ],
            "Resource": [
                "arn:aws:s3:::DOC-EXAMPLE-BUCKET"
            ]
        }
    ]
}
```

For instructions, see [Working with Inline Policies](#) in the *IAM User Guide*.

Step 2.4: Test permissions

Now Dave in Account B can list the contents of DOC-EXAMPLE-BUCKET owned by Account A. You can verify the permissions using either of the following procedures.

Test using the AWS CLI

1. Add the UserDave profile to the AWS CLI config file. For more information about the config file, see [Setting up the tools for the example walkthroughs \(p. 525\)](#).

```
[profile UserDave]
aws_access_key_id = access-key
aws_secret_access_key = secret-access-key
```

```
region = us-east-1
```

- At the command prompt, enter the following AWS CLI command to verify Dave can now get an object list from the **DOC-EXAMPLE-BUCKET** owned by Account A. Note the command specifies the UserDave profile.

```
aws s3 ls s3://DOC-EXAMPLE-BUCKET --profile UserDave
```

Dave does not have any other permissions. So if he tries any other operation—for example, the following get bucket lifecycle configuration—Amazon S3 returns permission denied.

```
aws s3api get-bucket-lifecycle-configuration --bucket DOC-EXAMPLE-BUCKET --profile UserDave
```

Test using AWS Tools for Windows PowerShell

- Store Dave's credentials as AccountBDave.

```
set-awscredentials -AccessKey AccessKeyID -SecretKey SecretAccessKey -storeas AccountBDave
```

- Try the List Bucket command.

```
get-s3object -BucketName DOC-EXAMPLE-BUCKET -StoredCredentials AccountBDave
```

Dave does not have any other permissions. So if he tries any other operation—for example, the following get bucket lifecycle configuration—Amazon S3 returns permission denied.

```
get-s3bucketlifecycleconfiguration -BucketName DOC-EXAMPLE-BUCKET -StoredCredentials AccountBDave
```

Step 3: (Optional) Try explicit deny

You can have permissions granted via an ACL, a bucket policy, and a user policy. But if there is an explicit deny set via either a bucket policy or a user policy, the explicit deny takes precedence over any other permissions. For testing, let's update the bucket policy and explicitly deny Account B the s3:ListBucket permission. The policy also grants s3>ListBucket permission, but explicit deny takes precedence, and Account B or users in Account B will not be able to list objects in **DOC-EXAMPLE-BUCKET**.

- Using credentials of user AccountAdmin in Account A, replace the bucket policy by the following.

```
{  
    "Version": "2012-10-17",  
    "Statement": [  
        {  
            "Sid": "Example permissions",  
            "Effect": "Allow",  
            "Principal": {  
                "AWS": "arn:aws:iam::AccountB-ID:root"  
            },  
            "Action": [  
                "s3:GetLifecycleConfiguration",  
                "s3>ListBucket"  
            ],  
            "Resource": "arn:aws:s3:::DOC-EXAMPLE-BUCKET"  
        }  
    ]  
}
```

```
        "Resource": [
            "arn:aws:s3:::DOC-EXAMPLE-BUCKET"
        ],
    },
{
    "Sid": "Deny permission",
    "Effect": "Deny",
    "Principal": {
        "AWS": "arn:aws:iam::AccountB-ID:root"
    },
    "Action": [
        "s3>ListBucket"
    ],
    "Resource": [
        "arn:aws:s3:::DOC-EXAMPLE-BUCKET"
    ]
}
]
```

2. Now if you try to get a bucket list using AccountBadmin credentials, you will get access denied.

- Using the AWS CLI:

```
aws s3 ls s3://DOC-EXAMPLE-BUCKET --profile AccountBadmin
```

- Using the AWS Tools for Windows PowerShell:

```
get-s3object -BucketName DOC-EXAMPLE-BUCKET -StoredCredentials AccountBDave
```

Step 4: Clean up

1. After you are done testing, you can do the following to clean up.
 - Sign in to the AWS Management Console ([AWS Management Console](#)) using Account A credentials, and do the following:
 - In the Amazon S3 console, remove the bucket policy attached to **DOC-EXAMPLE-BUCKET**. In the bucket **Properties**, delete the policy in the **Permissions** section.
 - If the bucket is created for this exercise, in the Amazon S3 console, delete the objects and then delete the bucket.
 - In the IAM console, remove the AccountAadmin user.
2. Sign in to the AWS Management Console ([AWS Management Console](#)) using Account B credentials. In the IAM console, delete user AccountBadmin.

Example 3: Bucket owner granting permissions to objects it does not own

Topics

- [Step 0: Preparing for the walkthrough \(p. 538\)](#)
- [Step 1: Do the Account A tasks \(p. 539\)](#)
- [Step 2: Do the Account B tasks \(p. 540\)](#)
- [Step 3: Test permissions \(p. 540\)](#)
- [Step 4: Clean up \(p. 541\)](#)

The scenario for this example is that a bucket owner wants to grant permission to access objects, but not all objects in the bucket are owned by the bucket owner. For this example, the bucket owner is trying to grant permission to users in its own account.

A bucket owner can enable other AWS accounts to upload objects. By default, the bucket owner doesn't own objects written to a bucket by another AWS account. Objects are owned by the accounts that write them to an S3 bucket. If the bucket owner doesn't own objects in the bucket, the object owner must first grant permission to the bucket owner using an object ACL. Then, the bucket owner can grant permissions to an object that they do not own. For more information, see [Amazon S3 bucket and object ownership \(p. 396\)](#).

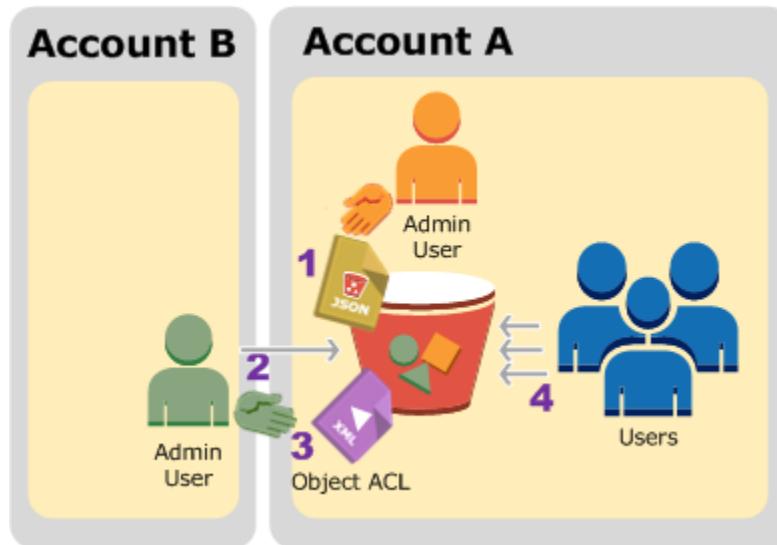
If the bucket owner applies the bucket owner enforced setting for S3 Object Ownership for the bucket, the bucket owner will own all objects in the bucket, including objects written by another AWS account. This will resolve the issue that objects are not owned by the bucket owner. Then, you can delegate permission to users in your own account or to other AWS accounts.

Note

By default, when another AWS account uploads an object to your S3 bucket, that account (the object writer) owns the object, has access to it, and can grant other users access to it through ACLs. You can use Object Ownership to change this default behavior so that ACLs are disabled and you, as the bucket owner, automatically own every object in your bucket. As a result, access control for your data is based on policies, such as IAM policies, S3 bucket policies, virtual private cloud (VPC) endpoint policies, and AWS Organizations service control policies (SCPs).

A majority of modern use cases in Amazon S3 no longer require the use of ACLs, and we recommend that you disable ACLs except in unusual circumstances where you need to control access for each object individually. With Object Ownership, you can disable ACLs and rely on policies for access control. When you disable ACLs, you can easily maintain a bucket with objects uploaded by different AWS accounts. You, as the bucket owner, own all the objects in the bucket and can manage access to them using policies. For more information, see [Controlling ownership of objects and disabling ACLs for your bucket \(p. 601\)](#).

In this example, we assume the bucket owner has not applied the bucket owner enforced setting for Object Ownership. The bucket owner delegates permission to users in its own account. The following is a summary of the walkthrough steps:



1. Account A administrator user attaches a bucket policy with two statements.
 - Allow cross-account permission to Account B to upload objects.
 - Allow a user in its own account to access objects in the bucket.
2. Account B administrator user uploads objects to the bucket owned by Account A.

3. Account B administrator updates the object ACL adding grant that gives the bucket owner full-control permission on the object.
4. User in Account A verifies by accessing objects in the bucket, regardless of who owns them.

For this example, you need two accounts. The following table shows how we refer to these accounts and the administrator users in these accounts. In this walkthrough, you don't use the account root credentials, according to the recommended IAM guidelines. For more information, see [About using an administrator user to create resources and grant permissions \(p. 525\)](#). Instead, you create an administrator in each account and use those credentials in creating resources and granting them permissions.

AWS account ID	Account referred to as	Administrator in the account
1111-1111-1111	Account A	AccountAadmin
2222-2222-2222	Account B	AccountBadmin

All the tasks of creating users and granting permissions are done in the AWS Management Console. To verify permissions, the walkthrough uses the command line tools, AWS Command Line Interface (AWS CLI) and AWS Tools for Windows PowerShell, so you don't need to write any code.

Step 0: Preparing for the walkthrough

1. Make sure that you have two AWS accounts and each account has one administrator as shown in the table in the preceding section.
 - a. Sign up for an AWS account, if needed.
 - i. Open the [Amazon S3 page](#) and choose **Create an AWS Account**.
 - ii. Follow the on-screen instructions. AWS will notify you by email when your account is active and available for you to use.
 - b. Using Account A credentials, sign in to the [IAM console](#) and do the following to create an administrator user:
 - Create user *AccountAadmin* and note down security credentials. For more information about adding users, see [Creating an IAM user in your AWS account](#) in the *IAM User Guide*.
 - Grant AccountAadmin administrator permissions by attaching a user policy giving full access. For instructions, see [Managing IAM policies](#) in the *IAM User Guide*.
 - In the IAM console **Dashboard**, note down the **IAM User Sign-In URL**. Users in this account must use this URL when signing in to the AWS Management Console. For more information, see [How users sign in to your account](#) in *IAM User Guide*.
 - c. Repeat the preceding step using Account B credentials and create administrator user *AccountBadmin*.
2. Set up either the AWS CLI or the Tools for Windows PowerShell. Make sure that you save the administrator credentials as follows:
 - If using the AWS CLI, create two profiles, *AccountAadmin* and *AccountBadmin*, in the config file.
 - If using the Tools for Windows PowerShell, make sure that you store credentials for the session as *AccountAadmin* and *AccountBadmin*.

For instructions, see [Setting up the tools for the example walkthroughs \(p. 525\)](#).

Step 1: Do the Account A tasks

Perform the following steps for Account A:

Step 1.1: Sign in to the console

Using the IAM user sign-in URL for Account A, sign in to the AWS Management Console as AccountAadmin user. This user will create a bucket and attach a policy to it.

Step 1.2: Create a bucket and user, and add a bucket policy to grant user permissions

1. In the Amazon S3 console, create a bucket. This exercise assumes that the bucket is created in the US East (N. Virginia) Region, and the name is **DOC-EXAMPLE-BUCKET1**.

For instructions, see [Creating a bucket \(p. 119\)](#).

2. In the IAM console, create a user *Dave*.

For instructions, see [Creating IAM users \(console\)](#) in the *IAM User Guide*.

3. Note down the *Dave* credentials.
4. In the Amazon S3 console, attach the following bucket policy to **DOC-EXAMPLE-BUCKET1** bucket. For instructions, see [Adding a bucket policy using the Amazon S3 console \(p. 488\)](#). Follow the steps to add a bucket policy. For information about how to find account IDs, see [Finding your AWS account ID](#).

The policy grants Account B the `s3:PutObject` and `s3>ListBucket` permissions. The policy also grants user *Dave* the `s3:GetObject` permission.

```
{  
    "Version": "2012-10-17",  
    "Statement": [  
        {  
            "Sid": "Statement1",  
            "Effect": "Allow",  
            "Principal": {  
                "AWS": "arn:aws:iam::AccountB-ID:root"  
            },  
            "Action": [  
                "s3:PutObject",  
                "s3>ListBucket"  
            ],  
            "Resource": [  
                "arn:aws:s3::::DOC-EXAMPLE-BUCKET1/*",  
                "arn:aws:s3::::DOC-EXAMPLE-BUCKET1"  
            ]  
        },  
        {  
            "Sid": "Statement3",  
            "Effect": "Allow",  
            "Principal": {  
                "AWS": "arn:aws:iam::AccountA-ID:user/Dave"  
            },  
            "Action": [  
                "s3:GetObject"  
            ],  
            "Resource": [  
                "arn:aws:s3::::DOC-EXAMPLE-BUCKET1/*"  
            ]  
        }  
    ]  
}
```

Step 2: Do the Account B tasks

Now that Account B has permissions to perform operations on Account A's bucket, the Account B administrator will do the following:

- Upload an object to Account A's bucket.
- Add a grant in the object ACL to allow Account A, the bucket owner, full control.

Using the AWS CLI

1. Using the `put-object` CLI command, upload an object. The `--body` parameter in the command identifies the source file to upload. For example, if the file is on C: drive of a Windows machine, you would specify `c:\HappyFace.jpg`. The `--key` parameter provides the key name for the object.

```
aws s3api put-object --bucket DOC-EXAMPLE-BUCKET1 --key HappyFace.jpg --body HappyFace.jpg --profile AccountBadmin
```

2. Add a grant to the object ACL to allow the bucket owner full control of the object. For information about how to find a canonical user ID, see [Finding the canonical user ID for your AWS account \(p. 561\)](#).

```
aws s3api put-object-acl --bucket DOC-EXAMPLE-BUCKET1 --key HappyFace.jpg --grant-full-control id="AccountA-CanonicalUserID" --profile AccountBadmin
```

Using the Tools for Windows PowerShell

1. Using the `Write-S3Object` Tools for Windows PowerShell command, upload an object.

```
Write-S3Object -BucketName DOC-EXAMPLE-BUCKET1 -key HappyFace.jpg -file HappyFace.jpg -StoredCredentials AccountBadmin
```

2. Add a grant to the object ACL to allow the bucket owner full control of the object.

```
Set-S3ACL -BucketName DOC-EXAMPLE-BUCKET1 -Key HappyFace.jpg -CannedACLName "bucket-owner-full-control" -StoredCreden
```

Step 3: Test permissions

Now verify that user Dave in Account A can access the object owned by Account B.

Using the AWS CLI

1. Add user Dave credentials to the AWS CLI config file and create a new profile, `UserDaveAccountA`. For more information, see [Setting up the tools for the example walkthroughs \(p. 525\)](#).

```
[profile UserDaveAccountA]
aws_access_key_id = access-key
aws_secret_access_key = secret-access-key
region = us-east-1
```

2. Run the `get-object` CLI command to download `HappyFace.jpg` and save it locally. You provide user Dave credentials by adding the `--profile` parameter.

```
aws s3api get-object --bucket DOC-EXAMPLE-BUCKET1 --key HappyFace.jpg Outputfile.jpg --  
profile UserDaveAccountA
```

Using the Tools for Windows PowerShell

1. Store user Dave AWS credentials, as UserDaveAccountA, to persistent store.

```
Set-AWSCredentials -AccessKey UserDave-AccessKey -SecretKey UserDave-SecretAccessKey -  
storeas UserDaveAccountA
```

2. Run the Read-S3Object command to download the HappyFace.jpg object and save it locally. You provide user Dave credentials by adding the -StoredCredentials parameter.

```
Read-S3Object -BucketName DOC-EXAMPLE-BUCKET1 -Key HappyFace.jpg -file HappyFace.jpg -  
StoredCredentials UserDaveAccountA
```

Step 4: Clean up

1. After you are done testing, you can do the following to clean up.
 - Sign in to the [AWS Management Console](#) using Account A credentials, and do the following:
 - In the Amazon S3 console, remove the bucket policy attached to **DOC-EXAMPLE-BUCKET1**. In the bucket **Properties**, delete the policy in the **Permissions** section.
 - If the bucket is created for this exercise, in the Amazon S3 console, delete the objects and then delete the bucket.
 - In the IAM console, remove the **AccountAadmin** user.
2. Sign in to the [AWS Management Console](#) using Account B credentials. In the IAM console, delete user **AccountBadmin**.

Example 4: Bucket owner granting cross-account permission to objects it does not own

Topics

- [Background: Cross-account permissions and using IAM roles \(p. 542\)](#)
- [Step 0: Preparing for the walkthrough \(p. 543\)](#)
- [Step 1: Do the account a tasks \(p. 545\)](#)
- [Step 2: Do the account b tasks \(p. 547\)](#)
- [Step 3: Do the account C tasks \(p. 547\)](#)
- [Step 4: Clean up \(p. 549\)](#)
- [Related resources \(p. 549\)](#)

In this example scenario, you own a bucket and you have enabled other AWS accounts to upload objects. If you have applied the bucket owner enforced setting for S3 Object Ownership for the bucket, you will own all objects in the bucket, including objects written by another AWS account. This will resolve the issue that objects are not owned by you, the bucket owner. Then, you can delegate permission to users in your own account or to other AWS accounts. Suppose the bucket owner enforced setting for S3 Object Ownership is not enabled. That is, your bucket can have objects that other AWS accounts own.

Now, suppose as a bucket owner, you need to grant cross-account permission on objects, regardless of who the owner is, to a user in another account. For example, that user could be a billing application that needs to access object metadata. There are two core issues:

- The bucket owner has no permissions on those objects created by other AWS accounts. So for the bucket owner to grant permissions on objects it does not own, the object owner, the AWS account that created the objects, must first grant permission to the bucket owner. The bucket owner can then delegate those permissions.
- Bucket owner account can delegate permissions to users in its own account (see [Example 3: Bucket owner granting permissions to objects it does not own \(p. 536\)](#)), but it cannot delegate permissions to other AWS accounts, because cross-account delegation is not supported.

In this scenario, the bucket owner can create an AWS Identity and Access Management (IAM) role with permission to access objects, and grant another AWS account permission to assume the role temporarily enabling it to access objects in the bucket.

Note

By default, when another AWS account uploads an object to your S3 bucket, that account (the object writer) owns the object, has access to it, and can grant other users access to it through ACLs. You can use Object Ownership to change this default behavior so that ACLs are disabled and you, as the bucket owner, automatically own every object in your bucket. As a result, access control for your data is based on policies, such as IAM policies, S3 bucket policies, virtual private cloud (VPC) endpoint policies, and AWS Organizations service control policies (SCPs).

A majority of modern use cases in Amazon S3 no longer require the use of ACLs, and we recommend that you disable ACLs except in unusual circumstances where you need to control access for each object individually. With Object Ownership, you can disable ACLs and rely on policies for access control. When you disable ACLs, you can easily maintain a bucket with objects uploaded by different AWS accounts. You, as the bucket owner, own all the objects in the bucket and can manage access to them using policies. For more information, see [Controlling ownership of objects and disabling ACLs for your bucket \(p. 601\)](#).

[Background: Cross-account permissions and using IAM roles](#)

IAM roles enable several scenarios to delegate access to your resources, and cross-account access is one of the key scenarios. In this example, the bucket owner, Account A, uses an IAM role to temporarily delegate object access cross-account to users in another AWS account, Account C. Each IAM role you create has two policies attached to it:

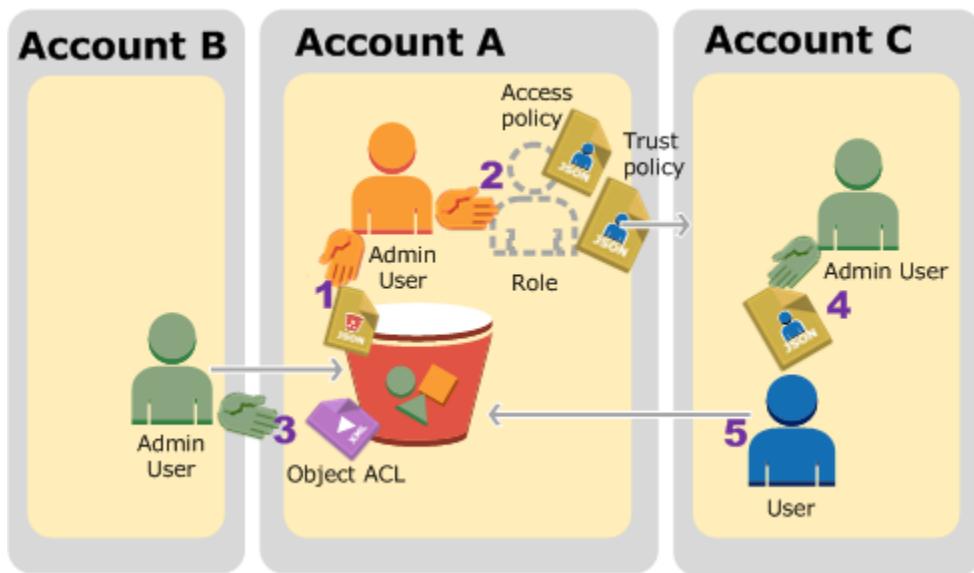
- A trust policy identifying another AWS account that can assume the role.
- An access policy defining what permissions—for example, `s3:GetObject`—are allowed when someone assumes the role. For a list of permissions you can specify in a policy, see [Amazon S3 actions \(p. 415\)](#).

The AWS account identified in the trust policy then grants its user permission to assume the role. The user can then do the following to access objects:

- Assume the role and, in response, get temporary security credentials.
- Using the temporary security credentials, access the objects in the bucket.

For more information about IAM roles, go to [IAM Roles](#) in *IAM User Guide*.

The following is a summary of the walkthrough steps:



1. Account A administrator user attaches a bucket policy granting Account B conditional permission to upload objects.
2. Account A administrator creates an IAM role, establishing trust with Account C, so users in that account can access Account A. The access policy attached to the role limits what user in Account C can do when the user accesses Account A.
3. Account B administrator uploads an object to the bucket owned by Account A, granting full-control permission to the bucket owner.
4. Account C administrator creates a user and attaches a user policy that allows the user to assume the role.
5. User in Account C first assumes the role, which returns the user temporary security credentials. Using those temporary credentials, the user then accesses objects in the bucket.

For this example, you need three accounts. The following table shows how we refer to these accounts and the administrator users in these accounts. Per IAM guidelines (see [About using an administrator user to create resources and grant permissions \(p. 525\)](#)) we do not use the account root credentials in this walkthrough. Instead, you create an administrator user in each account and use those credentials in creating resources and granting them permissions

AWS account ID	Account referred to as	Administrator user in the account
1111-1111-1111	Account A	AccountAadmin
2222-2222-2222	Account B	AccountBadmin
3333-3333-3333	Account C	AccountCadmin

Step 0: Preparing for the walkthrough

Note

You may want to open a text editor and write down some of the information as you walk through the steps. In particular, you will need account IDs, canonical user IDs, IAM User Sign-in URLs for each account to connect to the console, and Amazon Resource Names (ARNs) of the IAM users, and roles.

1. Make sure you have three AWS accounts and each account has one administrator user as shown in the table in the preceding section.
 - a. Sign up for AWS accounts, as needed. We refer to these accounts as Account A, Account B, and Account C.
 - i. Go to <https://aws.amazon.com/s3/> and click **Create an AWS Account**.
 - ii. Follow the on-screen instructions.AWS will notify you by email when your account is active and available for you to use.
 - b. Using Account A credentials, sign in to the **IAM console** and do the following to create an administrator user:
 - Create user AccountAadmin and note down security credentials. For more information about adding users, see [Creating an IAM User in Your AWS Account](#) in the *IAM User Guide*.
 - Grant AccountAadmin administrator privileges by attaching a user policy giving full access. For instructions, see [Working with Policies](#) in the *IAM User Guide*.
 - In the IAM Console **Dashboard**, note down the **IAM User Sign-In URL**. Users in this account must use this URL when signing in to the AWS Management Console. For more information, go to [How Users Sign In to Your Account](#) in *IAM User Guide*.
 - c. Repeat the preceding step to create administrator users in Account B and Account C.
2. For Account C, note down the canonical user ID.

When you create an IAM role in Account A, the trust policy grants Account C permission to assume the role by specifying the account ID. You can find account information as follows:

- a. Use your AWS account ID or account alias, your IAM user name, and your password to sign in to the [Amazon S3 Console](#).
 - b. Choose the name of an Amazon S3 bucket to view the details about that bucket.
 - c. Choose the **Permissions** tab and then choose **Access Control List**.
 - d. In the **Access for your AWS account** section, in the **Account** column is a long identifier, such as c1daexampleaf850ea79cf0430f33d72579fd1611c97f7ded193374c0b163b6. This is your canonical user ID.
3. When creating a bucket policy, you will need the following information. Note down these values:
 - **Canonical user ID of Account A** – When the Account A administrator grants conditional upload object permission to the Account B administrator, the condition specifies the canonical user ID of the Account A user that must get full-control of the objects.

Note

The canonical user ID is the Amazon S3-only concept. It is a 64-character obfuscated version of the account ID.

- **User ARN for Account B administrator** – You can find the user ARN in the IAM console. You will need to select the user and find the user's ARN in the **Summary** tab.

In the bucket policy, you grant AccountBadmin permission to upload objects and you specify the user using the ARN. Here's an example ARN value:

```
arn:aws:iam::AccountB-ID:user/AccountBadmin
```

4. Set up either the AWS Command Line Interface (CLI) or the AWS Tools for Windows PowerShell. Make sure you save administrator user credentials as follows:
 - If using the AWS CLI, create profiles, AccountAadmin and AccountBadmin, in the config file.
 - If using the AWS Tools for Windows PowerShell, make sure you store credentials for the session as AccountAadmin and AccountBadmin.

For instructions, see [Setting up the tools for the example walkthroughs \(p. 525\)](#).

Step 1: Do the account a tasks

In this example, Account A is the bucket owner. So user AccountAadmin in Account A will create a bucket, attach a bucket policy granting the Account B administrator permission to upload objects, create an IAM role granting Account C permission to assume the role so it can access objects in the bucket.

Step 1.1: Sign in to the AWS Management Console

Using the IAM User Sign-in URL for Account A, first sign in to the AWS Management Console as AccountAadmin user. This user will create a bucket and attach a policy to it.

Step 1.2: Create a bucket and attach a bucket policy

In the Amazon S3 console, do the following:

1. Create a bucket. This exercise assumes the bucket name is `examplebucket`.

For instructions, see [Creating a bucket \(p. 119\)](#).

2. Attach the following bucket policy granting conditional permission to the Account B administrator permission to upload objects.

You need to update the policy by providing your own values for `examplebucket`, `AccountB-ID`, and the `CanonicalUserId-of-AWSaccountA-BucketOwner`.

```
{  
    "Version": "2012-10-17",  
    "Statement": [  
        {  
            "Sid": "111",  
            "Effect": "Allow",  
            "Principal": {  
                "AWS": "arn:aws:iam::AccountB-ID:user/AccountBadmin"  
            },  
            "Action": "s3:PutObject",  
            "Resource": "arn:aws:s3:::awsexamplebucket1/*"  
        },  
        {  
            "Sid": "112",  
            "Effect": "Deny",  
            "Principal": {  
                "AWS": "arn:aws:iam::AccountB-ID:user/AccountBadmin"  
            },  
            "Action": "s3:PutObject",  
            "Resource": "arn:aws:s3:::awsexamplebucket1/*",  
            "Condition": {  
                "StringNotEquals": {  
                    "s3:x-amz-grant-full-control": "id=CanonicalUserId-of-AWSaccountA-  
BucketOwner"  
                }  
            }  
        }  
    ]  
}
```

Step 1.3: Create an IAM role to allow account C cross-account access in account a

In the IAM console, create an IAM role ("examplerole") that grants Account C permission to assume the role. Make sure you are still signed in as the Account A administrator because the role must be created in Account A.

1. Before creating the role, prepare the managed policy that defines the permissions that the role requires. You attach this policy to the role in a later step.
 - a. In the navigation pane on the left, click **Policies** and then click **Create Policy**.
 - b. Next to **Create Your Own Policy**, click **Select**.
 - c. Enter `access-accountA-bucket` in the **Policy Name** field.
 - d. Copy the following access policy and paste it into the **Policy Document** field. The access policy grants the role `s3:GetObject` permission so when Account C user assumes the role, it can only perform the `s3:GetObject` operation.

```
{  
    "Version": "2012-10-17",  
    "Statement": [  
        {  
            "Effect": "Allow",  
            "Action": "s3:GetObject",  
            "Resource": "arn:aws:s3:::awsexamplebucket1/*"  
        }  
    ]  
}
```

- e. Click **Create Policy**.

The new policy appears in the list of managed policies.

2. In the navigation pane on the left, click **Roles** and then click **Create New Role**.
3. Under **Select Role Type**, select **Role for Cross-Account Access**, and then click the **Select** button next to **Provide access between AWS accounts you own**.
4. Enter the Account C account ID.

For this walkthrough you do not need to require users to have multi-factor authentication (MFA) to assume the role, so leave that option unselected.

5. Click **Next Step** to set the permissions that will be associated with the role.
6. Select the box next to the `access-accountA-bucket` policy that you created and then click **Next Step**.

The Review page appears so you can confirm the settings for the role before it's created. One very important item to note on this page is the link that you can send to your users who need to use this role. Users who click the link go straight to the Switch Role page with the Account ID and Role Name fields already filled in. You can also see this link later on the Role Summary page for any cross-account role.

7. Enter `examplerole` for the role name, and then click **Next Step**.
8. After reviewing the role, click **Create Role**.

The `examplerole` role is displayed in the list of roles.

9. Click the role name `examplerole`.
10. Select the **Trust Relationships** tab.
11. Click **Show policy document** and verify the trust policy shown matches the following policy.

The following trust policy establishes trust with Account C, by allowing it the `sts:AssumeRole` action. For more information, go to [AssumeRole](#) in the *AWS Security Token Service API Reference*.

```
{  
    "Version": "2012-10-17",  
    "Statement": [  
        {  
            "Sid": "",  
            "Effect": "Allow",  
            "Principal": {  
                "AWS": "arn:aws:iam::AccountC-ID:root"  
            },  
            "Action": "sts:AssumeRole"  
        }  
    ]  
}
```

12. Note down the Amazon Resource Name (ARN) of the `examplerole` role you created.

Later in the following steps, you attach a user policy to allow an IAM user to assume this role, and you identify the role by the ARN value.

Step 2: Do the account b tasks

The `examplebucket` owned by Account A needs objects owned by other accounts. In this step, the Account B administrator uploads an object using the command line tools.

- Using the `put-object` AWS CLI command, upload an object to the `examplebucket`.

```
aws s3api put-object --bucket examplebucket --key HappyFace.jpg --body HappyFace.jpg --grant-full-control id="canonicalUserId-ofTheBucketOwner" --profile AccountBadmin
```

Note the following:

- The `--Profile` parameter specifies `AccountBadmin` profile, so the object is owned by Account B.
- The parameter `grant-full-control` grants the bucket owner full-control permission on the object as required by the bucket policy.
- The `--body` parameter identifies the source file to upload. For example, if the file is on the C: drive of a Windows computer, you specify `c:\HappyFace.jpg`.

Step 3: Do the account C tasks

In the preceding steps, Account A has already created a role, `examplerole`, establishing trust with Account C. This allows users in Account C to access Account A. In this step, Account C administrator creates a user (Dave) and delegates him the `sts:AssumeRole` permission it received from Account A. This will allow Dave to assume the `examplerole` and temporarily gain access to Account A. The access policy that Account A attached to the role will limit what Dave can do when he accesses Account A—specifically, get objects in `examplebucket`.

Step 3.1: Create a user in account C and delegate permission to assume `examplerole`

1. Using the IAM user sign-in URL for Account C, first sign in to the AWS Management Console as `AccountCadmin` user.
2. In the IAM console, create a user Dave.

For instructions, see [Creating IAM Users \(AWS Management Console\)](#) in the *IAM User Guide*.

3. Note down the Dave credentials. Dave will need these credentials to assume the `examplerole` role.
4. Create an inline policy for the Dave IAM user to delegate the `sts:AssumeRole` permission to Dave on the `examplerole` role in account A.
 - a. In the navigation pane on the left, click **Users**.
 - b. Click the user name Dave.
 - c. On the user details page, select the **Permissions** tab and then expand the **Inline Policies** section.
 - d. Choose **click here (or Create User Policy)**.
 - e. Click **Custom Policy**, and then click **Select**.
 - f. Enter a name for the policy in the **Policy Name** field.
 - g. Copy the following policy into the **Policy Document** field.

You will need to update the policy by providing the Account A ID.

```
{  
    "Version": "2012-10-17",  
    "Statement": [  
        {  
            "Effect": "Allow",  
            "Action": ["sts:AssumeRole"],  
            "Resource": "arn:aws:iam::AccountA-ID:role/examplerole"  
        }  
    ]  
}
```

h. Click **Apply Policy**

5. Save Dave's credentials to the config file of the AWS CLI by adding another profile, `AccountCDave`.

```
[profile AccountCDave]  
aws_access_key_id = UserDaveAccessKeyID  
aws_secret_access_key = UserDaveSecretAccessKey  
region = us-west-2
```

Step 3.2: Assume role (`examplerole`) and access objects

Now Dave can access objects in the bucket owned by Account A as follows:

- Dave first assumes the `examplerole` using his own credentials. This will return temporary credentials.
- Using the temporary credentials, Dave will then access objects in Account A's bucket.

1. At the command prompt, run the following AWS CLI `assume-role` command using the `AccountCDave` profile.

You will need to update the ARN value in the command by providing the Account A ID where `examplerole` is defined.

```
aws sts assume-role --role-arn arn:aws:iam::accountA-ID:role/examplerole --profile  
AccountCDave --role-session-name test
```

In response, AWS Security Token Service (STS) returns temporary security credentials (access key ID, secret access key, and a session token).

2. Save the temporary security credentials in the AWS CLI config file under the `TempCred` profile.

```
[profile TempCred]
aws_access_key_id = temp-access-key-ID
aws_secret_access_key = temp-secret-access-key
aws_session_token = session-token
region = us-west-2
```

3. At the command prompt, run the following AWS CLI command to access objects using the temporary credentials. For example, the command specifies the head-object API to retrieve object metadata for the HappyFace.jpg object.

```
aws s3api get-object --bucket examplebucket --key HappyFace.jpg SaveFileAs.jpg --
profile TempCred
```

Because the access policy attached to `examplerole` allows the actions, Amazon S3 processes the request. You can try any other action on any other object in the bucket.

If you try any other action—for example, `get-object-acl`—you will get permission denied because the role is not allowed that action.

```
aws s3api get-object-acl --bucket examplebucket --key HappyFace.jpg --profile TempCred
```

We used user Dave to assume the role and access the object using temporary credentials. It could also be an application in Account C that accesses objects in `examplebucket`. The application can obtain temporary security credentials, and Account C can delegate the application permission to assume `examplerole`.

Step 4: Clean up

1. After you are done testing, you can do the following to clean up.
 - Sign in to the AWS Management Console ([AWS Management Console](#)) using account A credentials, and do the following:
 - In the Amazon S3 console, remove the bucket policy attached to `examplebucket`. In the bucket **Properties**, delete the policy in the **Permissions** section.
 - If the bucket is created for this exercise, in the Amazon S3 console, delete the objects and then delete the bucket.
 - In the IAM console, remove the `examplerole` you created in Account A.
 - In the IAM console, remove the AccountAdmin user.
2. Sign in to the AWS Management Console ([AWS Management Console](#)) using Account B credentials. In the IAM console, delete user AccountBadmin.
3. Sign in to the AWS Management Console ([AWS Management Console](#)) using Account C credentials. In the IAM console, delete user AccountCadmin and user Dave.

Related resources

- [Creating a Role to Delegate Permissions to an IAM User](#) in the *IAM User Guide*.
- [Tutorial: Delegate Access Across AWS Accounts Using IAM Roles](#) in the *IAM User Guide*.
- [Working with Policies](#) in the *IAM User Guide*.

Using service-linked roles for Amazon S3 Storage Lens

To use Amazon S3 Storage Lens to collect and aggregate metrics across all your accounts in AWS Organizations, you must first ensure that S3 Storage Lens has trusted access enabled by the management account in your organization. S3 Storage Lens creates a service-linked role to allow it to get the list of AWS accounts belonging to your organization. This list of accounts is used by S3 Storage Lens to collect metrics for S3 resources in all the member accounts when the S3 Storage Lens dashboard or configurations are created or updated.

Amazon S3 Storage Lens uses AWS Identity and Access Management (IAM) [service-linked roles](#). A service-linked role is a unique type of IAM role that is linked directly to S3 Storage Lens. Service-linked roles are predefined by S3 Storage Lens and include all the permissions that the service requires to call other AWS services on your behalf.

A service-linked role makes setting up S3 Storage Lens easier because you don't have to add the necessary permissions manually. S3 Storage Lens defines the permissions of its service-linked roles, and unless defined otherwise, only S3 Storage Lens can assume its roles. The defined permissions include the trust policy and the permissions policy, and that permissions policy cannot be attached to any other IAM entity.

You can delete this service-linked role only after first deleting the related resources. This protects your S3 Storage Lens resources because you can't inadvertently remove permission to access the resources.

For information about other services that support service-linked roles, see [AWS services that work with IAM](#) and look for the services that have **Yes** in the **Service-Linked Role** column. Choose a **Yes** with a link to view the service-linked role documentation for that service.

Service-linked role permissions for Amazon S3 Storage Lens

S3 Storage Lens uses the service-linked role named **AWSServiceRoleForS3StorageLens** – This enables access to AWS services and resources used or managed by S3 Storage Lens. This allows S3 Storage Lens to access AWS Organizations resources on your behalf.

The S3 Storage Lens service-linked role trusts the following service on your organization's storage:

- storage-lens.s3.amazonaws.com

The role permissions policy allows S3 Storage Lens to complete the following actions:

- organizations:DescribeOrganization
- organizations>ListAccounts
- organizations>ListAWSServiceAccessForOrganization
- organizations>ListDelegatedAdministrators

You must configure permissions to allow an IAM entity (such as a user, group, or role) to create, edit, or delete a service-linked role. For more information, see [Service-linked role permissions](#) in the *IAM User Guide*.

Creating a service-linked role for S3 Storage Lens

You don't need to manually create a service-linked role. When you complete one of the following tasks while signed into the AWS Organizations management or the delegate administrator accounts, S3 Storage Lens creates the service-linked role for you:

- Create an S3 Storage Lens dashboard configuration for your organization in the Amazon S3 console.

- **PUT** an S3 Storage Lens configuration for your organization using the REST API, AWS CLI and SDKs.

Note

S3 Storage Lens will support a maximum of five delegated administrators per Organization.

If you delete this service-linked role, the preceding actions will re-create it as needed.

Example policy for S3 Storage Lens service-linked role

Example Permissions policy for the S3 Storage Lens service-linked role

```
{  
    "Version": "2012-10-17",  
    "Statement": [  
        {  
            "Sid": "AwsOrgsAccess",  
            "Effect": "Allow",  
            "Action": [  
                "organizations:DescribeOrganization",  
                "organizations>ListAccounts",  
                "organizations>ListAWSServiceAccessForOrganization",  
                "organizations>ListDelegatedAdministrators"  
            ],  
            "Resource": [  
                "*"  
            ]  
        }  
    ]  
}
```

Editing a service-linked role for Amazon S3 Storage Lens

S3 Storage Lens does not allow you to edit the AWSServiceRoleForS3StorageLens service-linked role. After you create a service-linked role, you cannot change the name of the role because various entities might reference the role. However, you can edit the description of the role using IAM. For more information, see [Editing a service-linked role](#) in the *IAM User Guide*.

Deleting a service-linked role for Amazon S3 Storage Lens

If you no longer need to use the service-linked role, we recommend that you delete that role. That way you don't have an unused entity that is not actively monitored or maintained. However, you must clean up the resources for your service-linked role before you can manually delete it.

Note

If the Amazon S3 Storage Lens service is using the role when you try to delete the resources, then the deletion might fail. If that happens, wait for a few minutes and try the operation again.

To delete the AWSServiceRoleForS3StorageLens you must delete all the organization level S3 Storage Lens configurations present in all Regions using the AWS Organizations Management or the delegate administrator accounts.

The resources are organization-level S3 Storage Lens configurations. Use S3 Storage Lens to clean up the resources and then use the IAM console, CLI, REST API or AWS SDK to delete the role.

In the REST API, AWS CLI and SDKs, S3 Storage Lens configurations can be discovered using `ListStorageLensConfigurations` in all the Regions where your Organization has created S3 Storage Lens configurations. Use the action `DeleteStorageLensConfiguration` to delete these configurations so that you can then delete the role.

Note

To delete the service-linked role, you must delete all the organization-level S3 Storage Lens configurations in all the Regions where they exist.

To delete Amazon S3 Storage Lens resources used by the AWSServiceRoleForS3StorageLens

1. You must use the `ListStorageLensConfigurations` in every Region that you have S3 Storage Lens configurations to get a list of your organization level configurations. This list may also be obtained from the Amazon S3 console.
2. These configurations then must be deleted from the appropriate regional endpoints by invoking the `DeleteStorageLensConfiguration` API call or via the Amazon S3 console.

To manually delete the service-linked role using IAM

After the configurations are deleted the `AWSServiceRoleForS3StorageLens` can be deleted from the IAM console or by invoking the IAM API `DeleteServiceLinkedRole`, the AWS CLI, or the AWS SDK. For more information, see [Deleting a service-linked role](#) in the *IAM User Guide*.

Supported Regions for S3 Storage Lens service-linked roles

S3 Storage Lens supports using service-linked roles in all of the AWS Regions where the service is available. For more information, see [Amazon S3 Regions and Endpoints](#).

AWS managed policies for Amazon S3

To add permissions to users, groups, and roles, it is easier to use AWS managed policies than to write policies yourself. It takes time and expertise to [create IAM customer managed policies](#) that provide your team with only the permissions they need. To get started quickly, you can use our AWS managed policies. These policies cover common use cases and are available in your AWS account. For more information about AWS managed policies, see [AWS managed policies](#) in the *IAM User Guide*.

AWS services maintain and update AWS managed policies. You can't change the permissions in AWS managed policies. Services occasionally add additional permissions to an AWS managed policy to support new features. This type of update affects all identities (users, groups, and roles) where the policy is attached. Services are most likely to update an AWS managed policy when a new feature is launched or when new operations become available. Services do not remove permissions from an AWS managed policy, so policy updates won't break your existing permissions.

Additionally, AWS supports managed policies for job functions that span multiple services. For example, the `ViewOnlyAccess` AWS managed policy provides read-only access to many AWS services and resources. When a service launches a new feature, AWS adds read-only permissions for new operations and resources. For a list and descriptions of job function policies, see [AWS managed policies for job functions](#) in the *IAM User Guide*.

AWS managed policy: AmazonS3FullAccess

You can attach the `AmazonS3FullAccess` policy to your IAM identities. This policy grants permissions that allow full access to Amazon S3.

To view the permissions for this policy, see [AmazonS3FullAccess](#) in the AWS Management Console.

AWS managed policy: AmazonS3ReadOnlyAccess

You can attach the `AmazonS3ReadOnlyAccess` policy to your IAM identities. This policy grants permissions that allow read-only access to Amazon S3.

To view the permissions for this policy, see [AmazonS3ReadOnlyAccess](#) in the AWS Management Console.

AWS managed policy: [AmazonS3ObjectLambdaExecutionRolePolicy](#)

Provides AWS Lambda functions the required permissions to send data to S3 Object Lambda when requests are made to an S3 Object Lambda access point. Also grants Lambda permissions to write to Amazon CloudWatch logs.

To view the permissions for this policy, see [AmazonS3ObjectLambdaExecutionRolePolicy](#) in the AWS Management Console.

Amazon S3 updates to AWS managed policies

View details about updates to AWS managed policies for Amazon S3 since this service began tracking these changes.

Change	Description	Date
Amazon S3 added S3 Object Lambda permissions to AmazonS3FullAccess and AmazonS3ReadOnlyAccess	Amazon S3 updated the AmazonS3FullAccess and AmazonS3ReadOnlyAccess policies to include permissions for S3 Object Lambda.	September 27, 2021
Amazon S3 added AmazonS3ObjectLambdaExecutionRolePolicy	Amazon S3 added a new AWS managed policy called AmazonS3ObjectLambdaExecutionRolePolicy that provides Lambda functions permissions to interact with S3 Object Lambda and write to CloudWatch logs.	August 18, 2021
Amazon S3 started tracking changes	Amazon S3 started tracking changes for its AWS managed policies.	August 18, 2021

Managing access with ACLs

Access control lists (ACLs) are one of the resource-based options (see [Overview of managing access \(p. 395\)](#)) that you can use to manage access to your buckets and objects. You can use ACLs to grant basic read/write permissions to other AWS accounts. There are limits to managing permissions using ACLs.

For example, you can grant permissions only to other AWS accounts; you cannot grant permissions to users in your account. You cannot grant conditional permissions, nor can you explicitly deny permissions. ACLs are suitable for specific scenarios. For example, if a bucket owner allows other AWS accounts to upload objects, permissions to these objects can only be managed using object ACL by the AWS account that owns the object.

By default, when another AWS account uploads an object to your S3 bucket, that account (the object writer) owns the object, has access to it, and can grant other users access to it through ACLs. You can use Object Ownership to change this default behavior so that ACLs are disabled and you, as the bucket owner, automatically own every object in your bucket. As a result, access control for your data is based on policies, such as IAM policies, S3 bucket policies, virtual private cloud (VPC) endpoint policies, and AWS Organizations service control policies (SCPs).

A majority of modern use cases in Amazon S3 no longer require the use of ACLs, and we recommend that you disable ACLs except in unusual circumstances where you need to control access for each object individually. With Object Ownership, you can disable ACLs and rely on policies for access control. When you disable ACLs, you can easily maintain a bucket with objects uploaded by different AWS accounts. You, as the bucket owner, own all the objects in the bucket and can manage access to them using policies. For more information, see [Controlling ownership of objects and disabling ACLs for your bucket \(p. 601\)](#).

Important

If your bucket uses the bucket owner enforced setting for S3 Object Ownership, you must use policies to grant access to your bucket and the objects in it. Requests to set ACLs or update ACLs fail and return the `AccessControlListNotSupported` error code. Requests to read ACLs are still supported.

For more information about ACLs, see the following topics.

Topics

- [Access control list \(ACL\) overview \(p. 554\)](#)
- [Finding the canonical user ID for your AWS account \(p. 561\)](#)
- [Configuring ACLs \(p. 562\)](#)

Access control list (ACL) overview

Amazon S3 access control lists (ACLs) enable you to manage access to buckets and objects. Each bucket and object has an ACL attached to it as a subresource. It defines which AWS accounts or groups are granted access and the type of access. When a request is received against a resource, Amazon S3 checks the corresponding ACL to verify that the requester has the necessary access permissions.

By default, when another AWS account uploads an object to your S3 bucket, that account (the object writer) owns the object, has access to it, and can grant other users access to it through ACLs. You can use Object Ownership to change this default behavior so that ACLs are disabled and you, as the bucket owner, automatically own every object in your bucket. As a result, access control for your data is based on policies, such as IAM policies, S3 bucket policies, virtual private cloud (VPC) endpoint policies, and AWS Organizations service control policies (SCPs).

A majority of modern use cases in Amazon S3 no longer require the use of ACLs, and we recommend that you disable ACLs except in unusual circumstances where you need to control access for each

object individually. With Object Ownership, you can disable ACLs and rely on policies for access control. When you disable ACLs, you can easily maintain a bucket with objects uploaded by different AWS accounts. You, as the bucket owner, own all the objects in the bucket and can manage access to them using policies. For more information, see [Controlling ownership of objects and disabling ACLs for your bucket \(p. 601\)](#).

Important

If your bucket uses the bucket owner enforced setting for S3 Object Ownership, you must use policies to grant access to your bucket and the objects in it. Requests to set ACLs or update ACLs fail and return the `AccessControlListNotSupported` error code. Requests to read ACLs are still supported.

When you create a bucket or an object, Amazon S3 creates a default ACL that grants the resource owner full control over the resource. This is shown in the following sample bucket ACL (the default object ACL has the same structure):

Example

```
<?xml version="1.0" encoding="UTF-8"?>
<AccessControlPolicy xmlns="http://s3.amazonaws.com/doc/2006-03-01/">
  <Owner>
    <ID>*** Owner-Canonical-User-ID ***</ID>
    <DisplayName>owner-display-name</DisplayName>
  </Owner>
  <AccessControlList>
    <Grant>
      <Grantee xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        xsi:type="Canonical User">
        <ID>*** Owner-Canonical-User-ID ***</ID>
        <DisplayName>display-name</DisplayName>
      </Grantee>
      <Permission>FULL_CONTROL</Permission>
    </Grant>
  </AccessControlList>
</AccessControlPolicy>
```

The sample ACL includes an `Owner` element that identifies the owner by the AWS account's canonical user ID. For instructions on finding your canonical user id, see [Finding an AWS account canonical user ID \(p. 556\)](#). The `Grant` element identifies the grantee (either an AWS account or a predefined group) and the permission granted. This default ACL has one `Grant` element for the owner. You grant permissions by adding `Grant` elements, with each grant identifying the grantee and the permission.

Note

An ACL can have up to 100 grants.

Topics

- [Who is a grantee? \(p. 555\)](#)
- [What permissions can I grant? \(p. 557\)](#)
- [Sample ACL \(p. 559\)](#)
- [Canned ACL \(p. 560\)](#)

Who is a grantee?

A grantee can be an AWS account or one of the predefined Amazon S3 groups. You grant permission to an AWS account using the email address or the canonical user ID. However, if you provide an email address in your grant request, Amazon S3 finds the canonical user ID for that account and adds it to the ACL. The resulting ACLs always contain the canonical user ID for the AWS account, not the AWS account's email address.

When you grant access rights, you specify each grantee as a type=value pair, where the type is one of the following:

- `id` – if the value specified is the canonical user ID of an AWS account
- `uri` – if you are granting permissions to a predefined group
- `emailAddress` – if the value specified is the email address of an AWS account

Important

Using email addresses to specify a grantee is only supported in the following AWS Regions:

- US East (N. Virginia)
- US West (N. California)
- US West (Oregon)
- Asia Pacific (Singapore)
- Asia Pacific (Sydney)
- Asia Pacific (Tokyo)
- Europe (Ireland)
- South America (São Paulo)

For a list of all the Amazon S3 supported regions and endpoints, see [Regions and Endpoints](#) in the [Amazon Web Services General Reference](#).

Example Example: Email Address

For example, the following `x-amz-grant-read` header grants the AWS accounts identified by email addresses permissions to read object data and its metadata:

```
x-amz-grant-read: emailAddress="xyz@amazon.com", emailAddress="abc@amazon.com"
```

Warning

When you grant other AWS accounts access to your resources, be aware that the AWS accounts can delegate their permissions to users under their accounts. This is known as *cross-account access*. For information about using cross-account access, see [Creating a Role to Delegate Permissions to an IAM User in the IAM User Guide](#).

[Finding an AWS account canonical user ID](#)

The canonical user ID is associated with your AWS account. This ID is a long string of characters, such as `79a59df900b949e55d96a1e698fbacedfd6e09d98eacf8f8d5218e7cd47ef2be`. For information about how to find the canonical user ID for your account, see [Finding the canonical user ID for your AWS account \(p. 561\)](#).

You can also look up the canonical user ID of an AWS account by reading the ACL of a bucket or an object to which the AWS account has access permissions. When an individual AWS account is granted permissions by a grant request, a grant entry is added to the ACL with the account's canonical user ID.

Note

If you make your bucket public (not recommended) any unauthenticated user can upload objects to the bucket. These anonymous users don't have an AWS account. When an anonymous user uploads an object to your bucket Amazon S3 adds a special canonical user ID (`65a011a29cdf8ec533ec3d1ccaae921c`) as the object owner in the ACL. For more information, see [Amazon S3 bucket and object ownership \(p. 396\)](#).

Amazon S3 predefined groups

Amazon S3 has a set of predefined groups. When granting account access to a group, you specify one of our URIs instead of a canonical user ID. We provide the following predefined groups:

- **Authenticated Users group** – Represented by `http://acs.amazonaws.com/groups/global/AuthenticatedUsers`.

This group represents all AWS accounts. **Access permission to this group allows any AWS account to access the resource.** However, all requests must be signed (authenticated).

Warning

When you grant access to the **Authenticated Users group** any AWS authenticated user in the world can access your resource.

- **All Users group** – Represented by `http://acs.amazonaws.com/groups/global/AllUsers`.

Access permission to this group allows anyone in the world access to the resource. The requests can be signed (authenticated) or unsigned (anonymous). Unsigned requests omit the Authentication header in the request.

Warning

We highly recommend that you never grant the **All Users group** `WRITE`, `WRITE_ACP`, or `FULL_CONTROL` permissions. For example, while `WRITE` permissions do not allow non-owners to overwrite or delete existing objects, `WRITE` permissions still allow anyone to store objects in your bucket, for which you are billed. For more details about these permissions, see the following section [What permissions can I grant? \(p. 557\)](#).

- **Log Delivery group** – Represented by `http://acs.amazonaws.com/groups/s3/LogDelivery`.

`WRITE` permission on a bucket enables this group to write server access logs (see [Logging requests using server access logging \(p. 978\)](#)) to the bucket.

Note

When using ACLs, a grantee can be an AWS account or one of the predefined Amazon S3 groups. However, the grantee cannot be an IAM user. For more information about AWS users and permissions within IAM, go to [Using AWS Identity and Access Management](#).

What permissions can I grant?

The following table lists the set of permissions that Amazon S3 supports in an ACL. The set of ACL permissions is the same for an object ACL and a bucket ACL. However, depending on the context (bucket ACL or object ACL), these ACL permissions grant permissions for specific buckets or object operations. The table lists the permissions and describes what they mean in the context of objects and buckets.

For more information about ACL permissions in the Amazon S3 console, see [Configuring ACLs \(p. 562\)](#).

ACL permissions

Permission	When granted on a bucket	When granted on an object
READ	Allows grantee to list the objects in the bucket.	Allows grantee to read the object data and its metadata
WRITE	Allows grantee to create new objects in the bucket. For the bucket and object owners of existing objects, also allows deletions and overwrites of those objects.	Not applicable
READ_ACP	Allows grantee to read the bucket ACL	Allows grantee to read the object ACL

Permission	When granted on a bucket	When granted on an object
WRITE_ACP	Allows grantee to write the ACL for the applicable bucket	Allows grantee to write the ACL for the applicable object
FULL_CONTROL	Allows grantee the READ, WRITE, READ_ACP, and WRITE_ACP permissions on the bucket	Allows grantee the READ, READ_ACP, and WRITE_ACP permissions on the object

Warning

Use caution when granting access permissions to your S3 buckets and objects. For example, granting `WRITE` access to a bucket allows the grantee to create objects in the bucket. We highly recommend that you read through this entire [Access control list \(ACL\) overview \(p. 554\)](#) section before granting permissions.

Mapping of ACL permissions and access policy permissions

As shown in the preceding table, an ACL allows only a finite set of permissions, compared to the number of permissions you can set in an access policy (see [Amazon S3 actions \(p. 415\)](#)). Each of these permissions allows one or more Amazon S3 operations.

The following table shows how each ACL permission maps to the corresponding access policy permissions. As you can see, access policy allows more permissions than an ACL does. You use ACLs primarily to grant basic read/write permissions, similar to file system permissions. For more information about when to use an ACL, see [Access policy guidelines \(p. 400\)](#).

For more information about ACL permissions in the Amazon S3 console, see [Configuring ACLs \(p. 562\)](#).

ACL permission	Corresponding access policy permissions when the ACL permission is granted on a bucket	Corresponding access policy permissions when the ACL permission is granted on an object
READ	<code>s3>ListBucket</code> , <code>s3>ListBucketVersions</code> , and <code>s3>ListBucketMultipartUploads</code>	<code>s3GetObject</code> and <code>s3GetObjectVersion</code>
WRITE	<code>s3PutObject</code> Bucket owner can create, overwrite, and delete any object in the bucket, and object owner has <code>FULL_CONTROL</code> over their object. In addition, when the grantee is the bucket owner, granting <code>WRITE</code> permission in a bucket ACL allows the <code>s3DeleteObjectVersion</code> action to be performed on any version in that bucket.	Not applicable
READ_ACP	<code>s3GetBucketAcl</code>	<code>s3GetObjectAcl</code> and <code>s3GetObjectVersionAcl</code>
WRITE_ACP	<code>s3PutBucketAcl</code>	<code>s3PutObjectAcl</code> and <code>s3PutObjectVersionAcl</code>
FULL_CONTROL	Equivalent to granting <code>READ</code> , <code>WRITE</code> , <code>READ_ACP</code> , and <code>WRITE_ACP</code> ACL	Equivalent to granting <code>READ</code> , <code>READ_ACP</code> , and <code>WRITE_ACP</code> ACL permissions.

ACL permission	Corresponding access policy permissions when the ACL permission is granted on a bucket	Corresponding access policy permissions when the ACL permission is granted on an object
	permissions. Accordingly, this ACL permission maps to a combination of corresponding access policy permissions.	Accordingly, this ACL permission maps to a combination of corresponding access policy permissions.

Condition keys

When you grant access policy permissions, you can use condition keys to constrain the value for the ACL on an object using a bucket policy. The context keys below correspond to ACLs. You can use these context keys to mandate the use of a specific ACL in a request:

- `s3:x-amz-grant-read` - Require read access.
- `s3:x-amz-grant-write` - Require write access.
- `s3:x-amz-grant-read-acp` - Require read access to the bucket ACL.
- `s3:x-amz-grant-write-acp` - Require write access to the bucket ACL.
- `s3:x-amz-grant-full-control` - Require full control.
- `s3:x-amz-acl` - Require a [Canned ACL \(p. 560\)](#).

For example policies that involves ACL-specific headers, see [Example 1: Granting s3:PutObject permission with a condition requiring the bucket owner to get full control \(p. 421\)](#). For a complete list of Amazon S3-specific condition keys, see [Actions, resources, and condition keys for Amazon S3 \(p. 431\)](#).

Sample ACL

The following sample ACL on a bucket identifies the resource owner and a set of grants. The format is the XML representation of an ACL in the Amazon S3 REST API. The bucket owner has `FULL_CONTROL` of the resource. In addition, the ACL shows how permissions are granted on a resource to two AWS accounts, identified by canonical user ID, and two of the predefined Amazon S3 groups discussed in the preceding section.

Example

```
<?xml version="1.0" encoding="UTF-8"?>
<AccessControlPolicy xmlns="http://s3.amazonaws.com/doc/2006-03-01/">
  <Owner>
    <ID>Owner-canonical-user-ID</ID>
    <DisplayName>display-name</DisplayName>
  </Owner>
  <AccessControlList>
    <Grant>
      <Grantee xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:type="CanonicalUser">
        <ID>Owner-canonical-user-ID</ID>
        <DisplayName>display-name</DisplayName>
      </Grantee>
      <Permission>FULL_CONTROL</Permission>
    </Grant>

    <Grant>
      <Grantee xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:type="CanonicalUser">
        <ID>user1-canonical-user-ID</ID>
        <DisplayName>display-name</DisplayName>
      </Grantee>
      <Permission>FULL_CONTROL</Permission>
    </Grant>
  </AccessControlList>
</AccessControlPolicy>
```

```

        </Grantee>
        <Permission>WRITE</Permission>
    </Grant>

    <Grant>
        <Grantee xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:type="CanonicalUser">
            <ID>user2-canonical-user-ID</ID>
            <DisplayName>display-name</DisplayName>
        </Grantee>
        <Permission>READ</Permission>
    </Grant>

    <Grant>
        <Grantee xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:type="Group">
            <URI>http://acs.amazonaws.com/groups/global/AllUsers</URI>
        </Grantee>
        <Permission>READ</Permission>
    </Grant>
    <Grant>
        <Grantee xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:type="Group">
            <URI>http://acs.amazonaws.com/groups/s3/LogDelivery</URI>
        </Grantee>
        <Permission>WRITE</Permission>
    </Grant>

</AccessControlList>
</AccessControlPolicy>

```

Canned ACL

Amazon S3 supports a set of predefined grants, known as *canned ACLs*. Each canned ACL has a predefined set of grantees and permissions. The following table lists the set of canned ACLs and the associated predefined grants.

Canned ACL	Applies to	Permissions added to ACL
private	Bucket and object	Owner gets FULL_CONTROL. No one else has access rights (default).
public-read	Bucket and object	Owner gets FULL_CONTROL. The AllUsers group (see Who is a grantee? (p. 555)) gets READ access.
public-read-write	Bucket and object	Owner gets FULL_CONTROL. The AllUsers group gets READ and WRITE access. Granting this on a bucket is generally not recommended.
aws-exec-read	Bucket and object	Owner gets FULL_CONTROL. Amazon EC2 gets READ access to GET an Amazon Machine Image (AMI) bundle from Amazon S3.
authenticated-read	Bucket and object	Owner gets FULL_CONTROL. The AuthenticatedUsers group gets READ access.
bucket-owner-read	Object	Object owner gets FULL_CONTROL. Bucket owner gets READ access. If you specify this canned ACL when creating a bucket, Amazon S3 ignores it.
bucket-owner-full-control	Object	Both the object owner and the bucket owner get FULL_CONTROL over the object. If you specify this canned ACL when creating a bucket, Amazon S3 ignores it.

Canned ACL	Applies to	Permissions added to ACL
log-delivery-write	Bucket	The LogDelivery group gets WRITE and READ_ACP permissions on the bucket. For more information about logs, see (Logging requests using server access logging (p. 978)).

Note

You can specify only one of these canned ACLs in your request.

You specify a canned ACL in your request using the `x-amz-acl` request header. When Amazon S3 receives a request with a canned ACL in the request, it adds the predefined grants to the ACL of the resource.

Finding the canonical user ID for your AWS account

The canonical user ID is an alpha-numeric identifier, such as `79a59df900b949e55d96a1e698fbacedfd6e09d98eacf8f8d5218e7cd47ef2be`, that is an obfuscated form of the AWS account ID. You can use this ID to identify an AWS account when granting cross-account access to buckets and objects using Amazon S3. You can retrieve the canonical user ID for your AWS account as either the root user or an IAM user.

You can find the canonical user ID for your AWS account using the AWS Management Console or the AWS CLI. The canonical user ID for an AWS account is specific to that account. You can retrieve the canonical user ID for your account as the root user, a federated user, or an IAM user.

Prerequisites

If you are a federated user or are accessing the information programmatically, such as through the AWS CLI, you must have permission to list and view an Amazon S3 bucket.

Using the S3 console (root user or an IAM user)

Follow these steps to find the canonical user ID for your AWS account when you are signed into the console as the root user or an IAM user. For more information about the root user and IAM users, see [Overview of AWS identity management: Users](#) in the *IAM User Guide*.

1. Sign in to the console as the root user or an IAM user.

For more information, see [Signing in to the AWS Management Console](#) in the *IAM User Guide*.

2. In the navigation bar on the upper right, choose your account name or number, and then choose **Security Credentials**.
3. Find the canonical ID for the account:
 - If you are the root user, expand **Account identifiers** and find **Canonical User ID**.
 - If you are an IAM user, under **Account details**, find **Account canonical user ID**.

Using the S3 console (federated user)

Follow these steps to find the canonical user ID for your account when you are signed in to the AWS Management Console as a federated user. For more information about federated users, see [Federating existing users](#) in the *IAM User Guide*.

Note

To verify that you've signed into the AWS Management Console as a federated user, on the AWS Management Console page, choose your account information and check the expanded account information. If **Federated user** is showing in the account information, you are signed in as a federated user.

1. Sign in to the console as a federated user.

For more information, see [Signing in to the AWS Management Console](#) in the *IAM User Guide*.

2. In the Amazon S3 console, choose a bucket name to view the bucket details.

3. Choose **Permissions**, and then scroll down to the **Access control list (ACL)** section.

Under **Bucket owner (your AWS account)**, the canonical user ID for the AWS account appears.

Using the AWS CLI

Use the `list-buckets` command as follows to find the canonical user ID using the AWS CLI.

```
aws s3api list-buckets --query Owner.ID --output text
```

Configuring ACLs

This section explains how to manage access permissions for S3 buckets and objects using access control lists (ACLs). You can add grants to your resource AC using the AWS Management Console, AWS Command Line Interface (CLI), REST API, or AWS SDKs.

Bucket and object permissions are independent of each other. An object does not inherit the permissions from its bucket. For example, if you create a bucket and grant write access to a user, you can't access that user's objects unless the user explicitly grants you access.

You can grant permissions to other AWS account users or to predefined groups. The user or group that you are granting permissions to is called the *grantee*. By default, the owner, which is the AWS account that created the bucket, has full permissions.

Each permission you grant for a user or group adds an entry in the ACL that is associated with the bucket. The ACL lists grants, which identify the grantee and the permission granted.

By default, when another AWS account uploads an object to your S3 bucket, that account (the object writer) owns the object, has access to it, and can grant other users access to it through ACLs. You can use Object Ownership to change this default behavior so that ACLs are disabled and you, as the bucket owner, automatically own every object in your bucket. As a result, access control for your data is based on policies, such as IAM policies, S3 bucket policies, virtual private cloud (VPC) endpoint policies, and AWS Organizations service control policies (SCPs).

A majority of modern use cases in Amazon S3 no longer require the use of ACLs, and we recommend that you disable ACLs except in unusual circumstances where you need to control access for each object individually. With Object Ownership, you can disable ACLs and rely on policies for access control. When you disable ACLs, you can easily maintain a bucket with objects uploaded by different AWS accounts. You, as the bucket owner, own all the objects in the bucket and can manage access to them using policies. For more information, see [Controlling ownership of objects and disabling ACLs for your bucket \(p. 601\)](#).

Important

If your bucket uses the bucket owner enforced setting for S3 Object Ownership, you must use policies to grant access to your bucket and the objects in it. Requests to set ACLs or update ACLs fail and return the `AccessControlListNotSupported` error code. Requests to read ACLs are still supported.

Warning

We highly recommend that you avoid granting write access to the **Everyone (public access)** or **Authenticated Users group (all AWS authenticated users)** groups. For more information about the effects of granting write access to these groups, see [Amazon S3 predefined groups \(p. 557\)](#).

Using the S3 console to set ACL permissions for a bucket

The console displays combined access grants for duplicate grantees. To see the full list of ACLs, use the Amazon S3 REST API, AWS CLI, or AWS SDKs.

The following table shows the ACL permissions that you can configure for buckets in the Amazon S3 console.

Amazon S3 console ACL permissions for buckets

Console permission	ACL permission	Access
Objects - List	READ	Allows grantee to list the objects in the bucket.
Objects - Write	WRITE	Allows grantee to create new objects in the bucket. For the bucket and object owners of existing objects, also allows deletions and overwrites of those objects.
Bucket ACL - Read	READ_ACP	Allows grantee to read the bucket ACL.
Bucket ACL - Write	WRITE_ACP	Allows grantee to write the ACL for the applicable bucket.
Everyone (public access): Objects - List	READ	Grants public read access for the objects in the bucket. When you grant list access to Everyone (public access) , anyone in the world can access the objects in the bucket.
Everyone (public access): Bucket ACL - Read	READ_ACP	Grants public read access for the bucket ACL. When you grant read access to Everyone (public access) , anyone in the world can access the bucket ACL.

For more information about ACL permissions, see [Access control list \(ACL\) overview \(p. 554\)](#).

Important

If your bucket uses the bucket owner enforced setting for S3 Object Ownership, you must use policies to grant access to your bucket and the objects in it. Requests to set ACLs or update ACLs fail and return the `AccessControlListNotSupported` error code. Requests to read ACLs are still supported.

To set ACL permissions for a bucket

1. Sign in to the AWS Management Console and open the Amazon S3 console at <https://console.aws.amazon.com/s3/>.
2. In the **Buckets** list, choose the name of the bucket that you want to set permissions for.
3. Choose **Permissions**.
4. Under **Access control list**, choose **Edit**.

You can edit the following ACL permissions for the bucket:

Objects

- **List** – Allows a grantee to list the objects in the bucket.
- **Write** – Allows grantee to create new objects in the bucket. For the bucket and object owners of existing objects, also allows deletions and overwrites of those objects.

In the S3 console, you can only grant write access to the S3 log delivery group and the bucket owner (your AWS account). We highly recommend that you do not grant write access for other grantees. However, if you need to grant write access, you can use the AWS CLI, AWS SDKs, or the REST API.

Bucket ACL

- **Read** – Allows grantee to read the bucket ACL.
 - **Write** – Allows grantee to write the ACL for the applicable bucket.
5. To change the bucket owner's permissions, beside **Bucket owner (your AWS account)**, clear or select from the following ACL permissions:
- **Objects – List or Write**
 - **Bucket ACL – Read or Write**

The *owner* refers to the AWS account root user, not an AWS Identity and Access Management (IAM) user. For more information about the root user, see [The AWS account root user](#) in the *IAM User Guide*.

6. To grant or undo permissions for the general public (everyone on the internet), beside **Everyone (public access)**, clear or select from the following ACL permissions:
- **Objects – List**
 - **Bucket ACL – Read**

Warning

Use caution when granting the **Everyone** group public access to your S3 bucket. When you grant access to this group, anyone in the world can access your bucket. We highly recommend that you never grant any kind of public write access to your S3 bucket.

7. To grant or undo permissions for anyone with an AWS account, beside **Authenticated Users group (anyone with an AWS account)**, clear or select from the following ACL permissions:
- **Objects – List**
 - **Bucket ACL – Read**
8. To grant or undo permissions for Amazon S3 to write server access logs to the bucket, under **S3 log delivery group**, clear or select from the following ACL permissions:
- **Objects – List or Write**
 - **Bucket ACL – Read or Write**

If a bucket is set up as the target bucket to receive access logs, the bucket permissions must allow the **Log Delivery** group write access to the bucket. When you enable server access logging on a bucket, the Amazon S3 console grants write access to the **Log Delivery** group for the target bucket that you choose to receive the logs. For more information about server access logging, see [Enabling Amazon S3 server access logging \(p. 980\)](#).

9. To grant access to another AWS account, do the following:
- a. Choose **Add grantee**.
 - b. In the **Grantee** box, enter the canonical ID of the other AWS account.
 - c. Select from the following ACL permissions:
- **Objects – List or Write**
 - **Bucket ACL – Read or Write**

Warning

When you grant other AWS accounts access to your resources, be aware that the AWS accounts can delegate their permissions to users under their accounts. This is known as *cross-account access*. For information about using cross-account access, see [Creating a Role to Delegate Permissions to an IAM User](#) in the *IAM User Guide*.

10. To remove access to another AWS account, under **Access for other AWS accounts**, choose **Remove**.
11. To save your changes, choose **Save changes**.

Using the S3 console to set ACL permissions for an object

The console displays combined access grants for duplicate grantees. To see the full list of ACLs, use the Amazon S3 REST API, AWS CLI, or AWS SDKs. The following table shows the ACL permissions that you can configure for objects in the Amazon S3 console.

Amazon S3 console ACL permissions for objects

Console permission	ACL permission	Access
Object - Read	READ	Allows grantee to read the object data and its metadata.
Object ACL - Read	READ_ACP	Allows grantee to read the object ACL.
Object ACL - Write	WRITE_ACP	Allows grantee to write the ACL for the applicable object

For more information about ACL permissions, see [Access control list \(ACL\) overview \(p. 554\)](#).

Important

If your bucket uses the bucket owner enforced setting for S3 Object Ownership, you must use policies to grant access to your bucket and the objects in it. Requests to set ACLs or update ACLs fail and return the `AccessControlListNotSupported` error code. Requests to read ACLs are still supported.

To set ACL permissions for 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 **Buckets** list, choose the name of the bucket that contains the object.
3. In the **objects** list, choose the name of the object for which you want to set permissions.
4. Choose **Permissions**.
5. Under Access control list (ACL), choose **Edit**.

You can edit the following ACL permissions for the object:

Object

- **Read** – Allows grantee to read the object data and its metadata.

Object ACL

- **Read** – Allows grantee to read the object ACL.

- **Write** – Allows grantee to write the ACL for the applicable object. In the S3 console, you can only grant write access to the bucket owner (your AWS account). We highly recommend that you do not grant write access for other grantees. However, if you need to grant write access, you can use the AWS CLI, AWS SDKs, or the REST API.
6. You can manage object access permissions for the following:

a. **Access for object owner**

The *owner* refers to the AWS account root user, and not an AWS Identity and Access Management (IAM) user. For more information about the root user, see [The AWS account root user](#) in the *IAM User Guide*.

To change the owner's object access permissions, under **Access for object owner**, choose **Your AWS Account (owner)**.

Select the check boxes for the permissions that you want to change, and then choose **Save**.

b. **Access for other AWS accounts**

To grant permissions to an AWS user from a different AWS account, under **Access for other AWS accounts**, choose **Add account**. In the **Enter an ID** field, enter the canonical ID of the AWS user that you want to grant object permissions to. For information about finding a canonical ID, see [Your AWS account identifiers](#) in the *Amazon Web Services General Reference*. You can add as many as 99 users.

Select the check boxes for the permissions that you want to grant to the user, and then choose **Save**. To display information about the permissions, choose the Help icons.

c. **Public access**

To grant access to your object to the general public (everyone in the world), under **Public access**, choose **Everyone**. Granting public access permissions means that anyone in the world can access the object.

Select the check boxes for the permissions that you want to grant, and then choose **Save**.

Warning

- Use caution when granting the **Everyone** group anonymous access to your Amazon S3 objects. When you grant access to this group, anyone in the world can access your object. If you need to grant access to everyone, we highly recommend that you only grant permissions to **Read objects**.
- We highly recommend that you *do not* grant the **Everyone** group write object permissions. Doing so allows anyone to overwrite the ACL permissions for the object.

Using the AWS SDKs

This section provides examples of how to configure access control list (ACL) grants on buckets and objects.

Important

If your bucket uses the bucket owner enforced setting for S3 Object Ownership, you must use policies to grant access to your bucket and the objects in it. Requests to set ACLs or update ACLs fail and return the `AccessControlListNotSupported` error code. Requests to read ACLs are still supported.

Java

This section provides examples of how to configure access control list (ACL) grants on buckets and objects. The first example creates a bucket with a canned ACL (see [Canned ACL \(p. 560\)](#)).

creates a list of custom permission grants, and then replaces the canned ACL with an ACL containing the custom grants. The second example shows how to modify an ACL using the `AccessControlList.grantPermission()` method.

Example Create a bucket and specify a canned ACL that grants permission to the S3 log delivery group

This example creates a bucket. In the request, the example specifies a canned ACL that grants the Log Delivery group permission to write logs to the bucket.

```
import com.amazonaws.AmazonServiceException;
import com.amazonaws.SdkClientException;
import com.amazonaws.regions.Regions;
import com.amazonaws.services.s3.AmazonS3;
import com.amazonaws.services.s3.AmazonS3ClientBuilder;
import com.amazonaws.services.s3.model.*;

import java.io.IOException;
import java.util.ArrayList;

public class CreateBucketWithACL {

    public static void main(String[] args) throws IOException {
        Regions clientRegion = Regions.DEFAULT_REGION;
        String bucketName = "*** Bucket name ***";
        String userEmailForReadPermission = "*** user@example.com ***";

        try {
            AmazonS3 s3Client = AmazonS3ClientBuilder.standard()
                .withRegion(clientRegion)
                .build();

            // Create a bucket with a canned ACL. This ACL will be replaced by the
            setBucketAcl()
            // calls below. It is included here for demonstration purposes.
            CreateBucketRequest createBucketRequest = new
CreateBucketRequest(bucketName, clientRegion.getName())
                .withCannedAcl(CannedAccessControlList.LogDeliveryWrite);
            s3Client.createBucket(createBucketRequest);

            // Create a collection of grants to add to the bucket.
            ArrayList<Grant> grantCollection = new ArrayList<Grant>();

            // Grant the account owner full control.
            Grant grant1 = new Grant(new
CanonicalGrantee(s3Client.getAWSAccountOwner().getId()), Permission.FullControl);
            grantCollection.add(grant1);

            // Grant the LogDelivery group permission to write to the bucket.
            Grant grant2 = new Grant(GroupGrantee.LogDelivery, Permission.Write);
            grantCollection.add(grant2);

            // Save grants by replacing all current ACL grants with the two we just
created.
            AccessControlList bucketAcl = new AccessControlList();
            bucketAcl.grantAllPermissions(grantCollection.toArray(new Grant[0]));
            s3Client.setBucketAcl(bucketName, bucketAcl);

            // Retrieve the bucket's ACL, add another grant, and then save the new ACL.
            AccessControlList newBucketAcl = s3Client.getBucketAcl(bucketName);
            Grant grant3 = new Grant(new
EmailAddressGrantee(userEmailForReadPermission), Permission.Read);
            newBucketAcl.grantAllPermissions(grant3);
        }
    }
}
```

```
        s3Client.setBucketAcl(bucketName, newBucketAcl);
    } catch (AmazonServiceException e) {
        // The call was transmitted successfully, but Amazon S3 couldn't process
        // it and returned an error response.
        e.printStackTrace();
    } catch (SdkClientException e) {
        // Amazon S3 couldn't be contacted for a response, or the client
        // couldn't parse the response from Amazon S3.
        e.printStackTrace();
    }
}
```

Example Update ACL on an existing object

This example updates the ACL on an object. The example performs the following tasks:

- Retrieves an object's ACL
- Clears the ACL by removing all existing permissions
- Adds two permissions: full access to the owner, and WRITE_ACP (see [What permissions can I grant? \(p. 557\)](#)) to a user identified by an email address
- Saves the ACL to the object

```
import com.amazonaws.AmazonServiceException;
import com.amazonaws.SdkClientException;
import com.amazonaws.auth.profile.ProfileCredentialsProvider;
import com.amazonaws.regions.Regions;
import com.amazonaws.services.s3.AmazonS3;
import com.amazonaws.services.s3.AmazonS3ClientBuilder;
import com.amazonaws.services.s3.model.AccessControlList;
import com.amazonaws.services.s3.model.CanonicalGrantee;
import com.amazonaws.services.s3.model.EmailAddressGrantee;
import com.amazonaws.services.s3.model.Permission;

import java.io.IOException;

public class ModifyACLExistingObject {

    public static void main(String[] args) throws IOException {
        Regions clientRegion = Regions.DEFAULT_REGION;
        String bucketName = "*** Bucket name ***";
        String keyName = "*** Key name ***";
        String emailGrantee = "*** user@example.com ***";

        try {
            AmazonS3 s3Client = AmazonS3ClientBuilder.standard()
                .withCredentials(new ProfileCredentialsProvider())
                .withRegion(clientRegion)
                .build();

            // Get the existing object ACL that we want to modify.
            AccessControlList acl = s3Client.getObjectAcl(bucketName, keyName);

            // Clear the existing list of grants.
            acl.getGrantsAsList().clear();

            // Grant a sample set of permissions, using the existing ACL owner for Full
            Control permissions.
            acl.grantPermission(new CanonicalGrantee(acl.getOwner().getId()),
Permission.FullControl);
        }
    }
}
```

```
        acl.grantPermission(new EmailAddressGrantee(emailGrantee),  
Permission.WriteAcp);  
  
        // Save the modified ACL back to the object.  
        s3Client.setObjectAcl(bucketName, keyName, acl);  
    } catch (AmazonServiceException e) {  
        // The call was transmitted successfully, but Amazon S3 couldn't process  
        // it, so it returned an error response.  
        e.printStackTrace();  
    } catch (SdkClientException e) {  
        // Amazon S3 couldn't be contacted for a response, or the client  
        // couldn't parse the response from Amazon S3.  
        e.printStackTrace();  
    }  
}
```

.NET

Example Create a bucket and specify a canned ACL that grants permission to the S3 log delivery group

This C# example creates a bucket. In the request, the code also specifies a canned ACL that grants the Log Delivery group permissions to write the logs to the bucket.

For instructions on creating and testing a working example, see [Running the Amazon S3 .NET Code Examples \(p. 1193\)](#).

```
using Amazon;  
using Amazon.S3;  
using Amazon.S3.Model;  
using System;  
using System.Threading.Tasks;  
  
namespace Amazon.DocSamples.S3  
{  
    class ManagingBucketACLTTest  
    {  
        private const string newBucketName = "**** bucket name ****";  
        // Specify your bucket region (an example region is shown).  
        private static readonly RegionEndpoint bucketRegion = RegionEndpoint.USWest2;  
        private static IAmazonS3 client;  
  
        public static void Main()  
        {  
            client = new AmazonS3Client(bucketRegion);  
            CreateBucketUseCannedACLAync().Wait();  
        }  
  
        private static async Task CreateBucketUseCannedACLAync()  
        {  
            try  
            {  
                // Add bucket (specify canned ACL).  
                PutBucketRequest putBucketRequest = new PutBucketRequest()  
                {  
                    BucketName = newBucketName,  
                    BucketRegion = S3Region.EUW1, // S3Region.US,  
                                         // Add canned ACL.  
                    CannedACL = S3CannedACL.LogDeliveryWrite  
                };  
                PutBucketResponse putBucketResponse = await  
client.PutBucketAsync(putBucketRequest);
```

```
// Retrieve bucket ACL.  
GetACLResponse getACLResponse = await client.GetACLAsync(new  
GetACLRequest  
{  
    BucketName = newBucketName  
});  
}  
catch (AmazonS3Exception amazonS3Exception)  
{  
    Console.WriteLine("S3 error occurred. Exception: " +  
amazonS3Exception.ToString());  
}  
catch (Exception e)  
{  
    Console.WriteLine("Exception: " + e.ToString());  
}  
}  
}  
}
```

Example Update ACL on an existing object

This C# example updates the ACL on an existing object. The example performs the following tasks:

- Retrieves an object's ACL.
 - Clears the ACL by removing all existing permissions.
 - Adds two permissions: full access to the owner, and WRITE_ACP to a user identified by email address.
 - Saves the ACL by sending a `PutAcl` request.

For instructions on creating and testing a working example, see [Running the Amazon S3 .NET Code Examples \(p. 1193\)](#).

```
using Amazon;
using Amazon.S3;
using Amazon.S3.Model;
using System;
using System.Collections.Generic;
using System.Threading.Tasks;

namespace Amazon.DocSamples.S3
{
    class ManagingObjectACLTTest
    {
        private const string bucketName = "*** bucket name ***";
        private const string keyName = "*** object key name ***";
        private const string emailAddress = "*** email address ***";
        // Specify your bucket region (an example region is shown).
        private static readonly RegionEndpoint bucketRegion = RegionEndpoint.USWest2;
        private static IAmazonS3 client;
        public static void Main()
        {
            client = new AmazonS3Client(bucketRegion);
            TestObjectACLTTestAsync().Wait();
        }
        private static async Task TestObjectACLTTestAsync()
        {
            try
            {
                // Retrieve the ACL for the object.
            }
        }
    }
}
```

Using the REST API

Amazon S3 APIs enable you to set an ACL when you create a bucket or an object. Amazon S3 also provides API to set an ACL on an existing bucket or an object. These APIs provide the following methods to set an ACL:

- **Set ACL using request headers**— When you send a request to create a resource (bucket or object), you set an ACL using the request headers. Using these headers, you can either specify a canned ACL or specify grants explicitly (identifying grantee and permissions explicitly).
- **Set ACL using request body**— When you send a request to set an ACL on an existing resource, you can set the ACL either in the request header or in the body.

For information on the REST API support for managing ACLs, see the following sections in the *Amazon Simple Storage Service API Reference*:

- [GET Bucket acl](#)
- [PUT Bucket acl](#)
- [GET Object acl](#)
- [PUT Object acl](#)
- [PUT Object](#)
- [PUT Bucket](#)
- [PUT Object - Copy](#)
- [Initiate Multipart Upload](#)

Important

If your bucket uses the bucket owner enforced setting for S3 Object Ownership, you must use policies to grant access to your bucket and the objects in it. Requests to set ACLs or update ACLs fail and return the `AccessControlListNotSupported` error code. Requests to read ACLs are still supported.

Access Control List (ACL)-Specific Request Headers

You can use headers to grant access control list (ACL)-based permissions. By default, all objects are private. Only the owner has full access control. When adding a new object, you can grant permissions to individual AWS accounts or to predefined groups defined by Amazon S3. These permissions are then added to the Access Control List (ACL) on the object. For more information, see [Access control list \(ACL\) overview \(p. 554\)](#).

With this operation, you can grant access permissions using one of these two methods:

- **Canned ACL (`x-amz-acl`)**— Amazon S3 supports a set of predefined ACLs, known as canned ACLs. Each canned ACL has a predefined set of grantees and permissions. For more information, see [Canned ACL \(p. 560\)](#).
- **Access Permissions**— To explicitly grant access permissions to specific AWS accounts or groups, use the following headers. Each header maps to specific permissions that Amazon S3 supports in an ACL. For more information, see [Access control list \(ACL\) overview \(p. 554\)](#). In the header, you specify a list of grantees who get the specific permission.
 - `x-amz-grant-read`
 - `x-amz-grant-write`
 - `x-amz-grant-read-acp`
 - `x-amz-grant-write-acp`
 - `x-amz-grant-full-control`

Using the AWS CLI

For more information about managing ACLs using the AWS CLI, see [put-bucket-acl](#) in the *AWS CLI Command Reference*.

Important

If your bucket uses the bucket owner enforced setting for S3 Object Ownership, you must use policies to grant access to your bucket and the objects in it. Requests to set ACLs or update ACLs fail and return the `AccessControlListNotSupported` error code. Requests to read ACLs are still supported.

Using cross-origin resource sharing (CORS)

Cross-origin resource sharing (CORS) defines a way for client web applications that are loaded in one domain to interact with resources in a different domain. With CORS support, you can build rich client-side web applications with Amazon S3 and selectively allow cross-origin access to your Amazon S3 resources.

This section provides an overview of CORS. The subtopics describe how you can enable CORS using the Amazon S3 console, or programmatically by using the Amazon S3 REST API and the AWS SDKs.

Cross-origin resource sharing: Use-case scenarios

The following are example scenarios for using CORS.

Scenario 1

Suppose that you are hosting a website in an Amazon S3 bucket named `website` as described in [Hosting a static website using Amazon S3 \(p. 1116\)](#). Your users load the website endpoint:

```
http://website.s3-website.us-east-1.amazonaws.com
```

Now you want to use JavaScript on the webpages that are stored in this bucket to be able to make authenticated GET and PUT requests against the same bucket by using the Amazon S3 API endpoint for the bucket, `website.s3.us-east-1.amazonaws.com`. A browser would normally block JavaScript from allowing those requests, but with CORS you can configure your bucket to explicitly enable cross-origin requests from `website.s3-website.us-east-1.amazonaws.com`.

Scenario 2

Suppose that you want to host a web font from your S3 bucket. Again, browsers require a CORS check (also called a preflight check) for loading web fonts. You would configure the bucket that is hosting the web font to allow any origin to make these requests.

How does Amazon S3 evaluate the CORS configuration on a bucket?

When Amazon S3 receives a preflight request from a browser, it evaluates the CORS configuration for the bucket and uses the first `CORSRule` rule that matches the incoming browser request to enable a cross-origin request. For a rule to match, the following conditions must be met:

- The request's `Origin` header must match an `AllowedOrigin` element.
- The request method (for example, GET or PUT) or the `Access-Control-Request-Method` header in case of a preflight `OPTIONS` request must be one of the `AllowedMethod` elements.
- Every header listed in the request's `Access-Control-Request-Headers` header on the preflight request must match an `AllowedHeader` element.

Note

The ACLs and policies continue to apply when you enable CORS on the bucket.

For more information about using CORS, see the following topics.

Topics

- [CORS configuration \(p. 574\)](#)
- [Configuring cross-origin resource sharing \(CORS\) \(p. 577\)](#)
- [Troubleshooting CORS \(p. 583\)](#)

CORS configuration

To configure your bucket to allow cross-origin requests, you create a CORS configuration. The CORS configuration is a document with rules that identify the origins that you will allow to access your bucket, the operations (HTTP methods) that you will support for each origin, and other operation-specific information. You can add up to 100 rules to the configuration. You can add the CORS configuration as the `cors` subresource to the bucket.

If you are configuring CORS in the S3 console, you must use JSON to create a CORS configuration. The new S3 console only supports JSON CORS configurations.

For more information about the CORS configuration and the elements in it, see the topics below. For instructions on how to add a CORS configuration, see [Configuring cross-origin resource sharing \(CORS\) \(p. 577\)](#).

Important

In the S3 console, the CORS configuration must be JSON.

Topics

- [Example 1 \(p. 574\)](#)
- [Example 2 \(p. 576\)](#)
- [AllowedMethod element \(p. 577\)](#)
- [AllowedOrigin element \(p. 577\)](#)
- [AllowedHeader element \(p. 577\)](#)
- [ExposeHeader element \(p. 577\)](#)
- [MaxAgeSeconds element \(p. 577\)](#)

Example 1

Instead of accessing a website by using an Amazon S3 website endpoint, you can use your own domain, such as `example1.com` to serve your content. For information about using your own domain, see [Configuring a static website using a custom domain registered with Route 53 \(p. 97\)](#).

The following example `cors` configuration has three rules, which are specified as `CORSRule` elements:

- The first rule allows cross-origin PUT, POST, and DELETE requests from the `http://www.example1.com` origin. The rule also allows all headers in a preflight OPTIONS request through the `Access-Control-Request-Headers` header. In response to preflight OPTIONS requests, Amazon S3 returns requested headers.
- The second rule allows the same cross-origin requests as the first rule, but the rule applies to another origin, `http://www.example2.com`.
- The third rule allows cross-origin GET requests from all origins. The `*` wildcard character refers to all origins.

JSON

```
[
```

```
{  
    "AllowedHeaders": [  
        "*"  
    ],  
    "AllowedMethods": [  
        "PUT",  
        "POST",  
        "DELETE"  
    ],  
    "AllowedOrigins": [  
        "http://www.example1.com"  
    ],  
    "ExposeHeaders": []  
},  
{  
    "AllowedHeaders": [  
        "*"  
    ],  

```

XML

```
<corsConfiguration>  
  <corsRule>  
    <allowedOrigin>http://www.example1.com</allowedOrigin>  
  
    <allowedMethod>PUT</allowedMethod>  
    <allowedMethod>POST</allowedMethod>  
    <allowedMethod>DELETE</allowedMethod>  
  
    <allowedHeader>*</allowedHeader>  
  </corsRule>  
  <corsRule>  
    <allowedOrigin>http://www.example2.com</allowedOrigin>  
  
    <allowedMethod>PUT</allowedMethod>  
    <allowedMethod>POST</allowedMethod>  
    <allowedMethod>DELETE</allowedMethod>  
  
    <allowedHeader>*</allowedHeader>  
  </corsRule>  
  <corsRule>  
    <allowedOrigin>*</allowedOrigin>  
    <allowedMethod>GET</allowedMethod>  
  </corsRule>
```

```
</CORSConfiguration>
```

Example 2

The CORS configuration also allows optional configuration parameters, as shown in the following CORS configuration. In this example, the CORS configuration allows cross-origin PUT, POST, and DELETE requests from the `http://www.example.com` origin.

JSON

```
[  
  {  
    "AllowedHeaders": [  
      "*"  
    ],  
    "AllowedMethods": [  
      "PUT",  
      "POST",  
      "DELETE"  
    ],  
    "AllowedOrigins": [  
      "http://www.example.com"  
    ],  
    "ExposeHeaders": [  
      "x-amz-server-side-encryption",  
      "x-amz-request-id",  
      "x-amz-id-2"  
    ],  
    "MaxAgeSeconds": 3000  
  }  
]
```

XML

```
<CORSConfiguration>  
  <CORSRule>  
    <AllowedOrigin>http://www.example.com</AllowedOrigin>  
    <AllowedMethod>PUT</AllowedMethod>  
    <AllowedMethod>POST</AllowedMethod>  
    <AllowedMethod>DELETE</AllowedMethod>  
    <AllowedHeader>*</AllowedHeader>  
    <MaxAgeSeconds>3000</MaxAgeSeconds>  
    <ExposeHeader>x-amz-server-side-encryption</  
    ExposeHeader>  
    <ExposeHeader>x-amz-request-id</  
    ExposeHeader>  
    <ExposeHeader>x-amz-id-2</ExposeHeader>  
  </CORSRule>  
</CORSConfiguration>
```

The `CORSRule` element in the preceding configuration includes the following optional elements:

- `MaxAgeSeconds`—Specifies the amount of time in seconds (in this example, 3000) that the browser caches an Amazon S3 response to a preflight OPTIONS request for the specified resource. By caching the response, the browser does not have to send preflight requests to Amazon S3 if the original request will be repeated.
- `ExposeHeader`—Identifies the response headers (in this example, `x-amz-server-side-encryption`, `x-amz-request-id`, and `x-amz-id-2`) that customers are able to access from their applications (for example, from a JavaScript `XMLHttpRequest` object).

AllowedMethod element

In the CORS configuration, you can specify the following values for the `AllowedMethod` element.

- GET
- PUT
- POST
- DELETE
- HEAD

AllowedOrigin element

In the `AllowedOrigin` element, you specify the origins that you want to allow cross-domain requests from, for example, `http://www.example.com`. The origin string can contain only one * wildcard character, such as `http://*.example.com`. You can optionally specify * as the origin to enable all the origins to send cross-origin requests. You can also specify `https` to enable only secure origins.

AllowedHeader element

The `AllowedHeader` element specifies which headers are allowed in a preflight request through the `Access-Control-Request-Headers` header. Each header name in the `Access-Control-Request-Headers` header must match a corresponding entry in the rule. Amazon S3 will send only the allowed headers in a response that were requested. For a sample list of headers that can be used in requests to Amazon S3, go to [Common Request Headers](#) in the *Amazon Simple Storage Service API Reference* guide.

Each `AllowedHeader` string in the rule can contain at most one * wildcard character. For example, `<AllowedHeader>x-amz-*</AllowedHeader>` will enable all Amazon-specific headers.

ExposeHeader element

Each `ExposeHeader` element identifies a header in the response that you want customers to be able to access from their applications (for example, from a JavaScript `XMLHttpRequest` object). For a list of common Amazon S3 response headers, go to [Common Response Headers](#) in the *Amazon Simple Storage Service API Reference* guide.

MaxAgeSeconds element

The `MaxAgeSeconds` element specifies the time in seconds that your browser can cache the response for a preflight request as identified by the resource, the HTTP method, and the origin.

Configuring cross-origin resource sharing (CORS)

Cross-origin resource sharing (CORS) defines a way for client web applications that are loaded in one domain to interact with resources in a different domain. With CORS support, you can build rich client-side web applications with Amazon S3 and selectively allow cross-origin access to your Amazon S3 resources.

This section shows you how to enable CORS using the Amazon S3 console, the Amazon S3 REST API, and the AWS SDKs. To configure your bucket to allow cross-origin requests, you add a CORS configuration to the bucket. A CORS configuration is a document that defines rules that identify the origins that you will allow to access your bucket, the operations (HTTP methods) supported for each origin, and other operation-specific information. In the S3 console, the CORS configuration must be a JSON document.

For example CORS configurations in JSON and XML, see [CORS configuration \(p. 574\)](#).

Using the S3 console

This section explains how to use the Amazon S3 console to add a cross-origin resource sharing (CORS) configuration to an S3 bucket.

When you enable CORS on the bucket, the access control lists (ACLs) and other access permission policies continue to apply.

Important

In the new S3 console, the CORS configuration must be JSON. For examples CORS configurations in JSON and XML, see [CORS configuration \(p. 574\)](#).

To add a CORS configuration to an S3 bucket

1. Sign in to the AWS Management Console and open the Amazon S3 console at <https://console.aws.amazon.com/s3/>.
2. In the **Buckets** list, choose the name of the bucket that you want to create a bucket policy for.
3. Choose **Permissions**.
4. In the **Cross-origin resource sharing (CORS)** section, choose **Edit**.
5. In the **CORS configuration editor** text box, type or copy and paste a new CORS configuration, or edit an existing configuration.

The CORS configuration is a JSON file. The text that you type in the editor must be valid JSON. For more information, see [CORS configuration \(p. 574\)](#).

6. Choose **Save changes**.

Note

Amazon S3 displays the Amazon Resource Name (ARN) for the bucket next to the **CORS configuration editor** title. For more information about ARNs, see [Amazon Resource Names \(ARNs\) and AWS Service Namespaces](#) in the *Amazon Web Services General Reference*.

Using the AWS SDKs

You can use the AWS SDK to manage cross-origin resource sharing (CORS) for a bucket. For more information about CORS, see [Using cross-origin resource sharing \(CORS\) \(p. 573\)](#).

The following examples:

- Creates a CORS configuration and sets the configuration on a bucket
- Retrieves the configuration and modifies it by adding a rule
- Adds the modified configuration to the bucket
- Deletes the configuration

Java

Example

Example

For instructions on how to create and test a working sample, see [Testing the Amazon S3 Java Code Examples \(p. 1192\)](#).

```
import com.amazonaws.AmazonServiceException;
```

```
import com.amazonaws.SdkClientException;
import com.amazonaws.auth.profile.ProfileCredentialsProvider;
import com.amazonaws.regions.Regions;
import com.amazonaws.services.s3.AmazonS3;
import com.amazonaws.services.s3.AmazonS3ClientBuilder;
import com.amazonaws.services.s3.model.BucketCrossOriginConfiguration;
import com.amazonaws.services.s3.model.CORSRule;

import java.io.IOException;
import java.util.ArrayList;
import java.util.Arrays;
import java.util.List;

public class CORS {

    public static void main(String[] args) throws IOException {
        Regions clientRegion = Regions.DEFAULT_REGION;
        String bucketName = "*** Bucket name ***";

        // Create two CORS rules.
        List<CORSRule.AllowedMethods> rule1AM = new
        ArrayList<CORSRule.AllowedMethods>();
        rule1AM.add(CORSRule.AllowedMethods.PUT);
        rule1AM.add(CORSRule.AllowedMethods.POST);
        rule1AM.add(CORSRule.AllowedMethods.DELETE);
        CORSRule rule1 = new CORSRule().withId("CORSRule1").withAllowedMethods(rule1AM)
            .withAllowedOrigins(Arrays.asList("http://*.example.com"));

        List<CORSRule.AllowedMethods> rule2AM = new
        ArrayList<CORSRule.AllowedMethods>();
        rule2AM.add(CORSRule.AllowedMethods.GET);
        CORSRule rule2 = new CORSRule().withId("CORSRule2").withAllowedMethods(rule2AM)
            .withAllowedOrigins(Arrays.asList("*")).withMaxAgeSeconds(3000)
            .withExposedHeaders(Arrays.asList("x-amz-server-side-encryption"));

        List<CORSRule> rules = new ArrayList<CORSRule>();
        rules.add(rule1);
        rules.add(rule2);

        // Add the rules to a new CORS configuration.
        BucketCrossOriginConfiguration configuration = new
        BucketCrossOriginConfiguration();
        configuration.setRules(rules);

        try {
            AmazonS3 s3Client = AmazonS3ClientBuilder.standard()
                .withCredentials(new ProfileCredentialsProvider())
                .withRegion(clientRegion)
                .build();

            // Add the configuration to the bucket.
            s3Client.setBucketCrossOriginConfiguration(bucketName, configuration);

            // Retrieve and display the configuration.
            configuration = s3Client.getBucketCrossOriginConfiguration(bucketName);
            printCORSConfiguration(configuration);

            // Add another new rule.
            List<CORSRule.AllowedMethods> rule3AM = new
            ArrayList<CORSRule.AllowedMethods>();
            rule3AM.add(CORSRule.AllowedMethods.HEAD);
            CORSRule rule3 = new
            CORSRule().withId("CORSRule3").withAllowedMethods(rule3AM)
                .withAllowedOrigins(Arrays.asList("http://www.example.com"));

            rules = configuration.getRules();
        }
    }
}
```

```

        rules.add(rule3);
        configuration.setRules(rules);
        s3Client.setBucketCrossOriginConfiguration(bucketName, configuration);

        // Verify that the new rule was added by checking the number of rules in
        // the configuration.
        configuration = s3Client.getBucketCrossOriginConfiguration(bucketName);
        System.out.println("Expected # of rules = 3, found " +
        configuration.getRules().size());

        // Delete the configuration.
        s3Client.deleteBucketCrossOriginConfiguration(bucketName);
        System.out.println("Removed CORS configuration.");

        // Retrieve and display the configuration to verify that it was
        // successfully deleted.
        configuration = s3Client.getBucketCrossOriginConfiguration(bucketName);
        printCORSConfiguration(configuration);
    } catch (AmazonServiceException e) {
        // The call was transmitted successfully, but Amazon S3 couldn't process
        // it, so it returned an error response.
        e.printStackTrace();
    } catch (SdkClientException e) {
        // Amazon S3 couldn't be contacted for a response, or the client
        // couldn't parse the response from Amazon S3.
        e.printStackTrace();
    }
}

private static void printCORSConfiguration(BucketCrossOriginConfiguration
configuration) {
    if (configuration == null) {
        System.out.println("Configuration is null.");
    } else {
        System.out.println("Configuration has " + configuration.getRules().size() +
        " rules\n");

        for (CORSRule rule : configuration.getRules()) {
            System.out.println("Rule ID: " + rule.getId());
            System.out.println("MaxAgeSeconds: " + rule.getMaxAgeSeconds());
            System.out.println("AllowedMethod: " + rule.getAllowedMethods());
            System.out.println("AllowedOrigins: " + rule.getAllowedOrigins());
            System.out.println("AllowedHeaders: " + rule.getAllowedHeaders());
            System.out.println("ExposeHeader: " + rule.getExposedHeaders());
            System.out.println();
        }
    }
}
}

```

.NET

Example

For information about creating and testing a working sample, see [Running the Amazon S3 .NET Code Examples \(p. 1193\)](#).

```

using Amazon;
using Amazon.S3;
using Amazon.S3.Model;
using System;
using System.Collections.Generic;
using System.Threading.Tasks;

```

```
namespace Amazon.DocSamples.S3
{
    class CORSTest
    {
        private const string bucketName = "*** bucket name ***";
        // Specify your bucket region (an example region is shown).
        private static readonly RegionEndpoint bucketRegion = RegionEndpoint.USWest2;
        private static IAmazonS3 s3Client;

        public static void Main()
        {
            s3Client = new AmazonS3Client(bucketRegion);
            CORSConfigTestAsync().Wait();
        }
        private static async Task CORSConfigTestAsync()
        {
            try
            {
                // Create a new configuration request and add two rules
                CORSConfiguration configuration = new CORSConfiguration
                {
                    Rules = new System.Collections.Generic.List<CORSRule>
                    {
                        new CORSRule
                        {
                            Id = "CORSRule1",
                            AllowedMethods = new List<string> {"PUT", "POST",
                            "DELETE"},
                            AllowedOrigins = new List<string> {"http://*.example.com"}
                        },
                        new CORSRule
                        {
                            Id = "CORSRule2",
                            AllowedMethods = new List<string> {"GET"},
                            AllowedOrigins = new List<string> {"*"},
                            MaxAgeSeconds = 3000,
                            ExposeHeaders = new List<string> {"x-amz-server-side-
encryption"}
                        }
                    };
                };

                // Add the configuration to the bucket.
                await PutCORSConfigurationAsync(configuration);

                // Retrieve an existing configuration.
                configuration = await RetrieveCORSConfigurationAsync();

                // Add a new rule.
                configuration.Rules.Add(new CORSRule
                {
                    Id = "CORSRule3",
                    AllowedMethods = new List<string> { "HEAD" },
                    AllowedOrigins = new List<string> { "http://www.example.com" }
                });

                // Add the configuration to the bucket.
                await PutCORSConfigurationAsync(configuration);

                // Verify that there are now three rules.
                configuration = await RetrieveCORSConfigurationAsync();
                Console.WriteLine();
                Console.WriteLine("Expected # of rules=3; found:{0}",
                configuration.Rules.Count);
                Console.WriteLine();
            }
        }
    }
}
```

```
Console.WriteLine("Pause before configuration delete. To continue,  
click Enter...");  
Console.ReadKey();  
  
// Delete the configuration.  
await DeleteCORSConfigurationAsync();  
  
// Retrieve a nonexistent configuration.  
configuration = await RetrieveCORSConfigurationAsync();  
}  
catch (AmazonS3Exception e)  
{  
    Console.WriteLine("Error encountered on server. Message:{0}' when  
writing an object", e.Message);  
}  
catch (Exception e)  
{  
    Console.WriteLine("Unknown encountered on server. Message:{0}' when  
writing an object", e.Message);  
}  
}  
  
static async Task PutCORSConfigurationAsync(CORSConfiguration configuration)  
{  
  
    PutCORSConfigurationRequest request = new PutCORSConfigurationRequest  
    {  
        BucketName = bucketName,  
        Configuration = configuration  
    };  
  
    var response = await s3Client.PutCORSConfigurationAsync(request);  
}  
  
static async Task<CORSConfiguration> RetrieveCORSConfigurationAsync()  
{  
    GetCORSConfigurationRequest request = new GetCORSConfigurationRequest  
    {  
        BucketName = bucketName  
    };  
    var response = await s3Client.GetCORSConfigurationAsync(request);  
    var configuration = response.Configuration;  
    PrintCORSRules(configuration);  
    return configuration;  
}  
  
static async Task DeleteCORSConfigurationAsync()  
{  
    DeleteCORSConfigurationRequest request = new DeleteCORSConfigurationRequest  
    {  
        BucketName = bucketName  
    };  
    await s3Client.DeleteCORSConfigurationAsync(request);  
}  
  
static void PrintCORSRules(CORSConfiguration configuration)  
{  
    Console.WriteLine();  
  
    if (configuration == null)  
    {  
        Console.WriteLine("\nConfiguration is null");  
        return;  
    }
```

```
Console.WriteLine("Configuration has {0} rules:",
configuration.Rules.Count);
foreach (CORSRule rule in configuration.Rules)
{
    Console.WriteLine("Rule ID: {0}", rule.Id);
    Console.WriteLine("MaxAgeSeconds: {0}", rule.MaxAgeSeconds);
    Console.WriteLine("AllowedMethod: {0}", string.Join(", ",
rule.AllowedMethods.ToArray()));
    Console.WriteLine("AllowedOrigins: {0}", string.Join(", ",
rule.AllowedOrigins.ToArray()));
    Console.WriteLine("AllowedHeaders: {0}", string.Join(", ",
rule.AllowedHeaders.ToArray()));
    Console.WriteLine("ExposeHeader: {0}", string.Join(", ",
rule.ExposeHeaders.ToArray()));
}
}
```

Using the REST API

To set a CORS configuration on your bucket, you can use the AWS Management Console. If your application requires it, you can also send REST requests directly. The following sections in the *Amazon Simple Storage Service API Reference* describe the REST API actions related to the CORS configuration:

- [PutBucketCors](#)
- [GetBucketCors](#)
- [DeleteBucketCors](#)
- [OPTIONS object](#)

Troubleshooting CORS

If you encounter unexpected behavior while accessing buckets set with the CORS configuration, try the following steps to troubleshoot:

1. Verify that the CORS configuration is set on the bucket.

If the CORS configuration is set, the console displays an **Edit CORS Configuration** link in the **Permissions** section of the **Properties** bucket.

2. Capture the complete request and response using a tool of your choice. For each request Amazon S3 receives, there must be a CORS rule that matches the data in your request, as follows:

- a. Verify that the request has the Origin header.

If the header is missing, Amazon S3 doesn't treat the request as a cross-origin request, and doesn't send CORS response headers in the response.

- b. Verify that the Origin header in your request matches at least one of the AllowedOrigin elements in the specified CORSRule.

The scheme, the host, and the port values in the Origin request header must match the AllowedOrigin elements in the CORSRule. For example, if you set the CORSRule to allow the origin `http://www.example.com`, then both `https://www.example.com` and `http://www.example.com:80` origins in your request don't match the allowed origin in your configuration.

- c. Verify that the method in your request (or in a preflight request, the method specified in the Access-Control-Request-Method) is one of the AllowedMethod elements in the same CORSRule.

- d. For a preflight request, if the request includes an `Access-Control-Request-Headers` header, verify that the `CORSRule` includes the `AllowedHeader` entries for each value in the `Access-Control-Request-Headers` header.

Blocking public access to your Amazon S3 storage

The Amazon S3 Block Public Access feature provides settings for access points, buckets, and accounts to help you manage public access to Amazon S3 resources. By default, new buckets, access points, and objects don't allow public access. However, users can modify bucket policies, access point policies, or object permissions to allow public access. S3 Block Public Access settings override these policies and permissions so that you can limit public access to these resources.

With S3 Block Public Access, account administrators and bucket owners can easily set up centralized controls to limit public access to their Amazon S3 resources that are enforced regardless of how the resources are created.

For instructions on configuring public block access, see [Configuring block public access \(p. 589\)](#).

When Amazon S3 receives a request to access a bucket or an object, it determines whether the bucket or the bucket owner's account has a block public access setting applied. If the request was made through an access point, Amazon S3 also checks for block public access settings for the access point. If there is an existing block public access setting that prohibits the requested access, Amazon S3 rejects the request.

Amazon S3 Block Public Access provides four settings. These settings are independent and can be used in any combination. Each setting can be applied to an access point, a bucket, or an entire AWS account. If the block public access settings for the access point, bucket, or account differ, then Amazon S3 applies the most restrictive combination of the access point, bucket, and account settings.

When Amazon S3 evaluates whether an operation is prohibited by a block public access setting, it rejects any request that violates an access point, bucket, or account setting.

Warning

Public access is granted to buckets and objects through access control lists (ACLs), access point policies, bucket policies, or all. To help ensure that all of your Amazon S3 access points, buckets, and objects have their public access blocked, we recommend that you turn on all four settings for block public access for your account. These settings block public access for all current and future buckets and access points.

Before applying these settings, verify that your applications will work correctly without public access. If you require some level of public access to your buckets or objects, for example to host a static website as described at [Hosting a static website using Amazon S3 \(p. 1116\)](#), you can customize the individual settings to suit your storage use cases.

Note

- You can enable block public access settings only for access points, buckets, and AWS accounts. Amazon S3 doesn't support block public access settings on a per-object basis.
- When you apply block public access settings to an account, the settings apply to all AWS Regions globally. The settings might not take effect in all Regions immediately or simultaneously, but they eventually propagate to all Regions.

Topics

- [Block public access settings \(p. 585\)](#)
- [Performing block public access operations on an access point \(p. 586\)](#)
- [The meaning of "public" \(p. 586\)](#)
- [Using Access Analyzer for S3 to review public buckets \(p. 589\)](#)
- [Permissions \(p. 589\)](#)

- [Configuring block public access \(p. 589\)](#)
- [Configuring block public access settings for your account \(p. 590\)](#)
- [Configuring block public access settings for your S3 buckets \(p. 591\)](#)

Block public access settings

S3 Block Public Access provides four settings. You can apply these settings in any combination to individual access points, buckets, or entire AWS accounts. If you apply a setting to an account, it applies to all buckets and access points that are owned by that account. Similarly, if you apply a setting to a bucket, it applies to all access points associated with that bucket.

The following table contains the available settings.

Name	Description
BlockPublicAccls	<p>Setting this option to <code>TRUE</code> causes the following behavior:</p> <ul style="list-style-type: none"> • PUT Bucket acl and PUT Object acl calls fail if the specified access control list (ACL) is public. • PUT Object calls fail if the request includes a public ACL. • If this setting is applied to an account, then PUT Bucket calls fail if the request includes a public ACL. <p>When this setting is set to <code>TRUE</code>, the specified operations fail (whether made through the REST API, AWS CLI, or AWS SDKs). However, existing policies and ACLs for buckets and objects are not modified. This setting enables you to protect against public access while allowing you to audit, refine, or otherwise alter the existing policies and ACLs for your buckets and objects.</p> <p>Note Access points don't have ACLs associated with them. If you apply this setting to an access point, it acts as a passthrough to the underlying bucket. If an access point has this setting enabled, requests made through the access point behave as though the underlying bucket has this setting enabled, regardless of whether the bucket actually has this setting enabled.</p>
IgnorePublicAccls	<p>Setting this option to <code>TRUE</code> causes Amazon S3 to ignore all public ACLs on a bucket and any objects that it contains. This setting enables you to safely block public access granted by ACLs while still allowing PUT Object calls that include a public ACL (as opposed to <code>BlockPublicAccls</code>, which rejects PUT Object calls that include a public ACL). Enabling this setting doesn't affect the persistence of any existing ACLs and doesn't prevent new public ACLs from being set.</p> <p>Note Access points don't have ACLs associated with them. If you apply this setting to an access point, it acts as a passthrough to the underlying bucket. If an access point has this setting enabled, requests made through the access point behave as though the underlying bucket has this setting enabled, regardless of whether the bucket actually has this setting enabled.</p>
BlockPublicPolicy	<p>Setting this option to <code>TRUE</code> for a bucket causes Amazon S3 to reject calls to PUT Bucket policy if the specified bucket policy allows public access, and to reject calls to PUT access point policy for all of the bucket's access points if the specified policy allows public access. Setting this option to <code>TRUE</code> for an access</p>

Name	Description
	<p>point causes Amazon S3 to reject calls to PUT access point policy and PUT Bucket policy that are made through the access point if the specified policy (for either the access point or the underlying bucket) is public.</p> <p>This setting enables you to allow users to manage access point and bucket policies without allowing them to publicly share the bucket or the objects it contains. Enabling this setting doesn't affect existing access point or bucket policies.</p> <p>Important To use this setting effectively, you should apply it at the <i>account</i> level. A bucket policy can allow users to alter a bucket's block public access settings. Therefore, users who have permission to change a bucket policy could insert a policy that allows them to disable the block public access settings for the bucket. If this setting is enabled for the entire account, rather than for a specific bucket, Amazon S3 blocks public policies even if a user alters the bucket policy to disable this setting.</p>
RestrictPublicBucket	<p>Setting this option to TRUE restricts access to an access point or bucket with a public policy to only AWS service principals and authorized users within the bucket owner's account. This setting blocks all cross-account access to the access point or bucket (except by AWS service principals), while still allowing users within the account to manage the access point or bucket.</p> <p>Enabling this setting doesn't affect existing access point or bucket policies, except that Amazon S3 blocks public and cross-account access derived from any public access point or bucket policy, including non-public delegation to specific accounts.</p>

Important

- Calls to GET Bucket acl and GET Object acl always return the effective permissions in place for the specified bucket or object. For example, suppose that a bucket has an ACL that grants public access, but the bucket also has the `IgnorePublicAcls` setting enabled. In this case, GET Bucket acl returns an ACL that reflects the access permissions that Amazon S3 is enforcing, rather than the actual ACL that is associated with the bucket.
- Block public access settings don't alter existing policies or ACLs. Therefore, removing a block public access setting causes a bucket or object with a public policy or ACL to again be publicly accessible.

Performing block public access operations on an access point

To perform block public access operations on an access point, use the AWS CLI service `s3control`.

Important

Note that it isn't currently possible to change an access point's block public access settings after creating the access point. Thus, the only way to specify block public access settings for an access point is by including them when creating the access point.

The meaning of "public"

Buckets

ACLs

Amazon S3 considers a bucket or object ACL public if it grants any permissions to members of the predefined AllUsers or AuthenticatedUsers groups. For more information about predefined groups, see [Amazon S3 predefined groups \(p. 557\)](#).

Policies

When evaluating a bucket policy, Amazon S3 begins by assuming that the policy is public. It then evaluates the policy to determine whether it qualifies as non-public. To be considered non-public, a bucket policy must grant access only to fixed values (values that don't contain a wildcard or an AWS Identity and Access Management Policy Variable) of one or more of the following:

- A set of Classless Inter-Domain Routings (CIDRs), using aws:SourceIp. For more information about CIDR, see [RFC 4632](#) on the RFC Editor website.
- An AWS principal, user, role, or service principal (e.g. aws:PrincipalOrgID)
- aws:SourceArn
- aws:SourceVpc
- aws:SourceVpce
- aws:SourceOwner
- aws:SourceAccount
- s3:x-amz-server-side-encryption-aws-kms-key-id
- aws:userid, outside the pattern "AROLEID:/*"
- s3:DataAccessPointArn

Note

When used in a bucket policy, this value can contain a wildcard for the access point name without rendering the policy public, as long as the account id is fixed. For example, allowing access to arn:aws:s3:us-west-2:123456789012:accesspoint/* would permit access to any access point associated with account 123456789012 in Region us-west-2, without rendering the bucket policy public. Note that this behavior is different for access point policies. For more information, see [Access points \(p. 588\)](#).

- s3:DataAccessPointAccount

Under these rules, the following example policies are considered public.

```
{  
    "Principal": { "Federated": "graph.facebook.com" },  
    "Resource": "*",  
    "Action": "s3:PutObject",  
    "Effect": "Allow"  
}
```

```
{  
    "Principal": "*",  
    "Resource": "*",  
    "Action": "s3:PutObject",  
    "Effect": "Allow"  
}
```

```
{  
    "Principal": "*",  
    "Resource": "*",  
    "Action": "s3:PutObject",  
    "Effect": "Allow",  
    "Condition": { "StringLike": {"aws:SourceVpc": "vpc-*"} }  
}
```

You can make these policies non-public by including any of the condition keys listed previously, using a fixed value. For example, you can make the last policy preceding non-public by setting `aws:SourceVpc` to a fixed value, like the following.

```
{  
    "Principal": "*",  
    "Resource": "*",  
    "Action": "s3:PutObject",  
    "Effect": "Allow",  
    "Condition": {"StringEquals": {"aws:SourceVpc": "vpc-91237329"}}  
}
```

For more information about bucket policies, see [Bucket policies and user policies \(p. 411\)](#).

Example

This example shows how Amazon S3 evaluates a bucket policy that contains both public and non-public access grants.

Suppose that a bucket has a policy that grants access to a set of fixed principals. Under the previously described rules, this policy isn't public. Thus, if you enable the `RestrictPublicBuckets` setting, the policy remains in effect as written, because `RestrictPublicBuckets` only applies to buckets that have public policies. However, if you add a public statement to the policy, `RestrictPublicBuckets` takes effect on the bucket. It allows only AWS service principals and authorized users of the bucket owner's account to access the bucket.

As an example, suppose that a bucket owned by "Account-1" has a policy that contains the following:

1. A statement that grants access to AWS CloudTrail (which is an AWS service principal)
2. A statement that grants access to account "Account-2"
3. A statement that grants access to the public, for example by specifying `"Principal": "*"` with no limiting Condition

This policy qualifies as public because of the third statement. With this policy in place and `RestrictPublicBuckets` enabled, Amazon S3 allows access only by CloudTrail. Even though statement 2 isn't public, Amazon S3 disables access by "Account-2." This is because statement 3 renders the entire policy public, so `RestrictPublicBuckets` applies. As a result, Amazon S3 disables cross-account access, even though the policy delegates access to a specific account, "Account-2." But if you remove statement 3 from the policy, then the policy doesn't qualify as public, and `RestrictPublicBuckets` no longer applies. Thus, "Account-2" regains access to the bucket, even if you leave `RestrictPublicBuckets` enabled.

Access points

Amazon S3 evaluates block public access settings slightly differently for access points compared to buckets. The rules that Amazon S3 applies to determine when an access point policy is public are generally the same for access points as for buckets, except in the following situations:

- An access point that has a VPC network origin is always considered non-public, regardless of the contents of its access point policy.
- An access point policy that grants access to a set of access points using `s3:DataAccessPointArn` is considered public. Note that this behavior is different than for bucket policies. For example, a bucket policy that grants access to values of `s3:DataAccessPointArn` that match `arn:aws:s3:us-west-2:123456789012:accesspoint/*` is not considered public. However, the same statement in an access point policy would render the access point public.

Using Access Analyzer for S3 to review public buckets

You can use Access Analyzer for S3 to review buckets with bucket ACLs, bucket policies, or access point policies that grant public access. Access Analyzer for S3 alerts you to buckets that are configured to allow access to anyone on the internet or other AWS accounts, including AWS accounts outside of your organization. For each public or shared bucket, you receive findings that report the source and level of public or shared access.

Armed with the knowledge presented in the findings, you can take immediate and precise corrective action. In Access Analyzer for S3, you can block all public access to a bucket with a single click. You can also drill down into bucket-level permission settings to configure granular levels of access. For specific and verified use cases that require public or shared access, you can acknowledge and record your intent for the bucket to remain public or shared by archiving the findings for the bucket.

In rare events, Access Analyzer for S3 might report no findings for a bucket that an Amazon S3 block public access evaluation reports as public. This happens because Amazon S3 block public access reviews policies for current actions and any potential actions that might be added in the future, leading to a bucket becoming public. On the other hand, Access Analyzer for S3 only analyzes the current actions specified for the Amazon S3 service in the evaluation of access status.

For more information about Access Analyzer for S3, see [Reviewing bucket access using Access Analyzer for S3 \(p. 593\)](#).

Permissions

To use Amazon S3 Block Public Access features, you must have the following permissions.

Operation	Required permissions
GET bucket policy status	s3:GetBucketPolicyStatus
GET bucket Block Public Access settings	s3:GetBucketPublicAccessBlock
PUT bucket Block Public Access settings	s3:PutBucketPublicAccessBlock
DELETE bucket Block Public Access settings	s3:PutBucketPublicAccessBlock
GET account Block Public Access settings	s3:GetAccountPublicAccessBlock
PUT account Block Public Access settings	s3:PutAccountPublicAccessBlock
DELETE account Block Public Access settings	s3:PutAccountPublicAccessBlock
PUT access point Block Public Access settings	s3>CreateAccessPoint

Note

The DELETE operations require the same permissions as the PUT operations. There are no separate permissions for the DELETE operations.

Configuring block public access

For more information about configuring block public access for your AWS account and your Amazon S3 buckets, see the following topics.

- [Configuring block public access settings for your account \(p. 590\)](#)
- [Configuring block public access settings for your S3 buckets \(p. 591\)](#)

Configuring block public access settings for your account

Amazon S3 Block Public Access provides settings for access points, buckets, and accounts to help you manage public access to Amazon S3 resources. By default, new buckets, access points, and objects do not allow public access.

For more information, see [Blocking public access to your Amazon S3 storage \(p. 584\)](#).

You can use the S3 console, AWS CLI, AWS SDKs, and REST API to configure block public access settings for your all the buckets in your account. For more information, see the sections below.

To configure block public access settings for your buckets, see [Configuring block public access settings for your S3 buckets \(p. 591\)](#). For information about access points, see [Performing block public access operations on an access point \(p. 586\)](#).

Using the S3 console

Amazon S3 block public access prevents the application of any settings that allow public access to data within S3 buckets. This section describes how to edit block public access settings for all the S3 buckets in your AWS account. For more information about blocking public access, see [Blocking public access to your Amazon S3 storage \(p. 584\)](#).

To edit block public access settings for all the S3 buckets in an AWS account

1. Sign in to the AWS Management Console and open the Amazon S3 console at <https://console.aws.amazon.com/s3/>.
2. Choose **Block Public Access settings for this account**.
3. Choose **Edit** to change the block public access settings for all the buckets in your AWS account.
4. Choose the settings that you want to change, and then choose **Save changes**.
5. When you're asked for confirmation, enter **confirm**. Then choose **Confirm** to save your changes.

Using the AWS CLI

You can use Amazon S3 Block Public Access through the AWS CLI. For more information about setting up and using the AWS CLI, see [What is the AWS Command Line Interface?](#)

Account

To perform block public access operations on an account, use the AWS CLI service `s3control`. The account-level operations that use this service are as follows:

- `PUT PublicAccessBlock` (for an account)
- `GET PublicAccessBlock` (for an account)
- `DELETE PublicAccessBlock` (for an account)

For additional information and examples, see [put-public-access-block](#) in the *AWS CLI Reference*.

Using the AWS SDKs

Java

The following examples show you how to use Amazon S3 Block Public Access with the AWS SDK for Java to put a public access block configuration on an Amazon S3 account. For instructions on how to create and test a working sample, see [Using the AWS SDK for Java \(p. 1191\)](#).

```
AWSS3ControlClientBuilder controlClientBuilder = AWSS3ControlClientBuilder.standard();
controlClientBuilder.setRegion(<region>);
controlClientBuilder.setCredentials(<credentials>);

AWSS3Control client = controlClientBuilder.build();
client.putPublicAccessBlock(new PutPublicAccessBlockRequest()
    .withAccountId(<account-id>)
    .withPublicAccessBlockConfiguration(new PublicAccessBlockConfiguration()
        .withIgnorePublicAcls(<value>)
        .withBlockPublicAccls(<value>)
        .withBlockPublicPolicy(<value>)
        .withRestrictPublicBuckets(<value>))));
```

Important

This example pertains only to account-level operations, which use the `AWSS3Control` client class. For bucket-level operations, see the preceding example.

Other SDKs

For information about using the other AWS SDKs, see [Developing with Amazon S3 using the AWS SDKs, and explorers \(p. 1184\)](#).

Using the REST API

For information about using Amazon S3 Block Public Access through the REST APIs, see the following topics in the *Amazon Simple Storage Service API Reference*.

- Account-level operations
 - [PUT PublicAccessBlock](#)
 - [GET PublicAccessBlock](#)
 - [DELETE PublicAccessBlock](#)

Configuring block public access settings for your S3 buckets

Amazon S3 Block Public Access provides settings for access points, buckets, and accounts to help you manage public access to Amazon S3 resources. By default, new buckets, access points, and objects do not allow public access.

For more information, see [Blocking public access to your Amazon S3 storage \(p. 584\)](#).

You can use the S3 console, AWS CLI, AWS SDKs, and REST API to configure block public access settings for your bucket. For more information, see the sections below.

To configure block public access settings for all the buckets in your account, see [Configuring block public access settings for your account \(p. 590\)](#). For information about configuring block public access for access points, see [Performing block public access operations on an access point \(p. 586\)](#).

Using the S3 console

Amazon S3 Block Public Access prevents the application of any settings that allow public access to data within S3 buckets. This section describes how to edit Block Public Access settings for one or more S3 buckets. For information about blocking public access using the AWS CLI, AWS SDKs, and the Amazon S3 REST APIs, see [Blocking public access to your Amazon S3 storage \(p. 584\)](#).

You can see if your bucket is publicly accessible in the **Buckets** list. In the **Access** column, Amazon S3 labels the permissions for a bucket as follows:

- **Public** – Everyone has access to one or more of the following: List objects, Write objects, Read and write permissions.
- **Objects can be public** – The bucket is not public, but anyone with the appropriate permissions can grant public access to objects.
- **Buckets and objects not public** – The bucket and objects do not have any public access.
- **Only authorized users of this account** – Access is isolated to IAM users and roles in this account and AWS service principals because there is a policy that grants public access.

You can also filter bucket searches by access type. Choose an access type from the drop-down list that is next to the **Search for buckets** bar.

If you see an **Error** when you list your buckets and their public access settings, you might not have the required permissions. Check to make sure you have the following permissions added to your user or role policy:

```
s3:GetAccountPublicAccessBlock  
s3:GetBucketPublicAccessBlock  
s3:GetBucketPolicyStatus  
s3:GetBucketLocation  
s3:GetBucketAcl  
s3>ListAccessPoints  
s3>ListAllMyBuckets
```

In some rare cases, requests can also fail because of an AWS Region outage.

To edit the Amazon S3 block public access settings for a single S3 bucket

Follow these steps if you need to change the public access settings for a single S3 bucket.

1. Sign in to the AWS Management Console and open the Amazon S3 console at <https://console.aws.amazon.com/s3/>.
2. In the **Bucket name** list, choose the name of the bucket that you want.
3. Choose **Permissions**.
4. Choose **Edit** to change the public access settings for the bucket. For more information about the four Amazon S3 Block Public Access Settings, see [Block public access settings \(p. 585\)](#).
5. Choose the setting that you want to change, and then choose **Save**.
6. When you're asked for confirmation, enter **confirm**. Then choose **Confirm** to save your changes.

You can change Amazon S3 Block Public Access settings when you create a bucket. For more information, see [Creating a bucket \(p. 119\)](#).

Using the AWS CLI

To perform block public access operations on a bucket, use the AWS CLI service `s3api`. The bucket-level operations that use this service are as follows:

- `PUT PublicAccessBlock` (for a bucket)
- `GET PublicAccessBlock` (for a bucket)
- `DELETE PublicAccessBlock` (for a bucket)
- `GET BucketPolicyStatus`

For more information and examples, see [put-public-access-block](#) in the *AWS CLI Reference*.

Using the AWS SDKs

Java

```
AmazonS3 client = AmazonS3ClientBuilder.standard()
    .withCredentials(<credentials>)
    .build();

client.setPublicAccessBlock(new SetPublicAccessBlockRequest()
    .withBucketName(<bucket-name>)
    .withPublicAccessBlockConfiguration(new PublicAccessBlockConfiguration()
        .withBlockPublicAcls(<value>)
        .withIgnorePublicAccls(<value>)
        .withBlockPublicPolicy(<value>)
        .withRestrictPublicBuckets(<value>)));
```

Important

This example pertains only to bucket-level operations, which use the `AmazonS3` client class. For account-level operations, see the following example.

Other SDKs

For information about using the other AWS SDKs, see [Developing with Amazon S3 using the AWS SDKs, and explorers \(p. 1184\)](#).

Using the REST API

For information about using Amazon S3 Block Public Access through the REST APIs, see the following topics in the [Amazon Simple Storage Service API Reference](#).

- Bucket-level operations
 - [PUT PublicAccessBlock](#)
 - [GET PublicAccessBlock](#)
 - [DELETE PublicAccessBlock](#)
 - [GET BucketPolicyStatus](#)

Reviewing bucket access using Access Analyzer for S3

Access Analyzer for S3 alerts you to S3 buckets that are configured to allow access to anyone on the internet or other AWS accounts, including AWS accounts outside of your organization. For each public or shared bucket, you receive findings into the source and level of public or shared access. For example, Access Analyzer for S3 might show that a bucket has read or write access provided through a bucket access control list (ACL), a bucket policy, a Multi-Region Access Point policy, or an access point policy. Armed with this knowledge, you can take immediate and precise corrective action to restore your bucket access to what you intended.

When reviewing an at-risk bucket in Access Analyzer for S3, you can block all public access to the bucket with a single click. We recommend that you block all access to your buckets unless you require public access to support a specific use case. Before you block all public access, ensure that your applications will continue to work correctly without public access. For more information, see [Blocking public access to your Amazon S3 storage \(p. 584\)](#).

You can also drill down into bucket-level permission settings to configure granular levels of access. For specific and verified use cases that require public access, such as static website hosting, public

downloads, or cross-account sharing, you can acknowledge and record your intent for the bucket to remain public or shared by archiving the findings for the bucket. You can revisit and modify these bucket configurations at any time. You can also download your findings as a CSV report for auditing purposes.

Access Analyzer for S3 is available at no extra cost on the Amazon S3 console. Access Analyzer for S3 is powered by AWS Identity and Access Management (IAM) Access Analyzer. To use Access Analyzer for S3 in the Amazon S3 console, you must visit the IAM console and enable IAM Access Analyzer on a per-Region basis.

For more information about IAM Access Analyzer, see [What is Access Analyzer?](#) in the *IAM User Guide*. For more information about Access Analyzer for S3, review the following sections.

Important

- Access Analyzer for S3 requires an account-level analyzer. To use Access Analyzer for S3, you must visit IAM Access Analyzer and create an analyzer that has an account as the zone of trust. For more information, see [Enabling Access Analyzer](#) in *IAM User Guide*.
- When a bucket policy or bucket ACL is added or modified, Access Analyzer generates and updates findings based on the change within 30 minutes. Findings related to account level block public access settings may not be generated or updated for up to 6 hours after you change the settings. Findings related to Multi-Region Access Points may not be generated or updated for up to six hours after the Multi-Region Access Point is created, deleted, or you change its policy.

Topics

- [What information does Access Analyzer for S3 provide? \(p. 594\)](#)
- [Enabling Access Analyzer for S3 \(p. 595\)](#)
- [Blocking all public access \(p. 595\)](#)
- [Reviewing and changing bucket access \(p. 596\)](#)
- [Archiving bucket findings \(p. 597\)](#)
- [Activating an archived bucket finding \(p. 597\)](#)
- [Viewing finding details \(p. 597\)](#)
- [Downloading an Access Analyzer for S3 report \(p. 598\)](#)

What information does Access Analyzer for S3 provide?

Access Analyzer for S3 provides findings for buckets that can be accessed outside your AWS account. Buckets that are listed under **Buckets with public access** can be accessed by anyone on the internet. If Access Analyzer for S3 identifies public buckets, you also see a warning at the top of the page that shows you the number of public buckets in your Region. Buckets listed under **Buckets with access from other AWS accounts — including third-party AWS accounts** are shared conditionally with other AWS accounts, including accounts outside of your organization.

For each bucket, Access Analyzer for S3 provides the following information:

- **Bucket name**
- **Discovered by Access analyzer** - When Access Analyzer for S3 discovered the public or shared bucket access.
- **Shared through** - How the bucket is shared—through a bucket policy, a bucket ACL, a Multi-Region Access Point policy, or an access point policy. Multi-Region Access Points are reflected under access points. A bucket can be shared through both policies and ACLs. If you want to find and review the source for your bucket access, you can use the information in this column as a starting point for taking immediate and precise corrective action.

- **Status** - The status of the bucket finding. Access Analyzer for S3 displays findings for all public and shared buckets.
 - **Active** - Finding has not been reviewed.
 - **Archived** - Finding has been reviewed and confirmed as intended.
 - **All** - All findings for buckets that are public or shared with other AWS accounts, including AWS accounts outside of your organization.
- **Access level** - Access permissions granted for the bucket:
 - **List** - List resources.
 - **Read** - Read but not edit resource contents and attributes.
 - **Write** - Create, delete, or modify resources.
 - **Permissions** - Grant or modify resource permissions.
 - **Tagging** - Update tags associated with the resource.

Enabling Access Analyzer for S3

To use Access Analyzer for S3, you must complete the following prerequisite steps.

1. Grant the required permissions.

For more information, see [Permissions Required to use Access Analyzer](#) in the *IAM User Guide*.

2. Visit IAM to create an account-level analyzer for each Region where you want to use Access Analyzer.

Access Analyzer for S3 requires an account-level analyzer. To use Access Analyzer for S3, you must create an analyzer that has an account as the zone of trust. For more information, see [Enabling Access Analyzer](#) in *IAM User Guide*.

Blocking all public access

If you want to block all access to a bucket in a single click, you can use the **Block all public access** button in Access Analyzer for S3. When you block all public access to a bucket, no public access is granted. We recommend that you block all public access to your buckets unless you require public access to support a specific and verified use case. Before you block all public access, ensure that your applications will continue to work correctly without public access.

If you don't want to block all public access to your bucket, you can edit your block public access settings on the Amazon S3 console to configure granular levels of access to your buckets. For more information, see [Blocking public access to your Amazon S3 storage \(p. 584\)](#).

In rare events, Access Analyzer for S3 might report no findings for a bucket that an Amazon S3 block public access evaluation reports as public. This happens because Amazon S3 block public access reviews policies for current actions and any potential actions that might be added in the future, leading to a bucket becoming public. On the other hand, Access Analyzer for S3 only analyzes the current actions specified for the Amazon S3 service in the evaluation of access status.

To block all public access to a bucket using Access Analyzer for S3

1. Sign in to the AWS Management Console and open the Amazon S3 console at <https://console.aws.amazon.com/s3/>.
2. In the navigation pane on the left, under **Dashboards**, choose **Access analyzer for S3**.
3. In Access Analyzer for S3, choose a bucket.
4. Choose **Block all public access**.
5. To confirm your intent to block all public access to the bucket, in **Block all public access (bucket settings)**, enter **confirm**.

Amazon S3 blocks all public access to your bucket. The status of the bucket finding updates to **resolved**, and the bucket disappears from the Access Analyzer for S3 listing. If you want to review resolved buckets, open IAM Access Analyzer on the IAM console.

Reviewing and changing bucket access

If you did not intend to grant access to the public or other AWS accounts, including accounts outside of your organization, you can modify the bucket ACL, bucket policy, the Multi-Region Access Point policy, or the access point policy to remove the access to the bucket. The **Shared through** column shows all sources of bucket access: bucket policy, bucket ACL, and/or access point policy. Multi-Region Access Points are reflected under access points.

To review and change a bucket policy, a bucket ACL, a Multi-Region Access Point, or an access point policy

1. Open the Amazon S3 console at <https://console.aws.amazon.com/s3/>.
2. In the navigation pane, choose **Access analyzer for S3**.
3. To see whether public access or shared access is granted through a bucket policy, a bucket ACL, a Multi-Region Access Point policy, or an access point policy, look in the **Shared through** column.
4. Under **Buckets**, choose the name of the bucket with the bucket policy, bucket ACL, Multi-Region Access Point policy, or access point policy that you want to change or review.
5. If you want to change or view a bucket ACL:
 - a. Choose **Permissions**.
 - b. Choose **Access Control List**.
 - c. Review your bucket ACL, and make changes as required.

For more information, see [Configuring ACLs \(p. 562\)](#).
6. If you want to change or review a bucket policy:
 - a. Choose **Permissions**.
 - b. Choose **Bucket Policy**.
 - c. Review or change your bucket policy as required.

For more information, see [Adding a bucket policy using the Amazon S3 console \(p. 488\)](#).
7. If you want to change or view a Multi-Region Access Point policy:
 - a. Choose **Multi-Region Access Point**.
 - b. Choose the Multi-Region Access Point name.
 - c. Review or change your Multi-Region Access Point policy as required.

For more information, see [Multi-Region Access Point permissions \(p. 328\)](#).
8. If you want to review or change an access point policy:
 - a. Choose **access points**.
 - b. Choose the access point name.
 - c. Review or change access as required.

For more information, see [Using Amazon S3 access points with the Amazon S3 console \(p. 312\)](#).

If you edit or remove a bucket ACL, a bucket policy, or an access point policy to remove public or shared access, the status for the bucket findings updates to resolved. The resolved bucket findings disappear from the Access Analyzer for S3 listing, but you can view them in IAM Access Analyzer.

Archiving bucket findings

If a bucket grants access to the public or other AWS accounts, including accounts outside of your organization, to support a specific use case (for example, a static website, public downloads, or cross-account sharing), you can archive the finding for the bucket. When you archive bucket findings, you acknowledge and record your intent for the bucket to remain public or shared. Archived bucket findings remain in your Access Analyzer for S3 listing so that you always know which buckets are public or shared.

To archive bucket findings in Access Analyzer for S3

1. Open the Amazon S3 console at <https://console.aws.amazon.com/s3/>.
2. In the navigation pane, choose **Access analyzer for S3**.
3. In Access Analyzer for S3, choose an active bucket.
4. To acknowledge your intent for this bucket to be accessed by the public or other AWS accounts, including accounts outside of your organization, choose **Archive**.
5. Enter **confirm**, and choose **Archive**.

Activating an archived bucket finding

After you archive findings, you can always revisit them and change their status back to active, indicating that the bucket requires another review.

To activate an archived bucket finding in Access Analyzer for S3

1. Open the Amazon S3 console at <https://console.aws.amazon.com/s3/>.
2. In the navigation pane, choose **Access analyzer for S3**.
3. Choose the archived bucket findings.
4. Choose **Mark as active**.

Viewing finding details

If you need to see more information about a bucket, you can open the bucket finding details in IAM Access Analyzer on the IAM console.

To view finding details in Access Analyzer for S3

1. Open the Amazon S3 console at <https://console.aws.amazon.com/s3/>.
2. In the navigation pane, choose **Access analyzer for S3**.
3. In Access Analyzer for S3, choose a bucket.
4. Choose **View details**.

The finding details open in IAM Access Analyzer on the IAM console.

Downloading an Access Analyzer for S3 report

You can download your bucket findings as a CSV report that you can use for auditing purposes. The report includes the same information that you see in Access Analyzer for S3 on the Amazon S3 console.

To download a report

1. Open the Amazon S3 console at <https://console.aws.amazon.com/s3/>.
2. In the navigation pane on the left, choose **Access analyzer for S3**.
3. In the Region filter, choose the Region.
Access Analyzer for S3 updates to shows buckets for the chosen Region.
4. Choose **Download report**.

A CSV report is generated and saved to your computer.

Verifying bucket ownership with bucket owner condition

Amazon S3 bucket owner condition ensures that the buckets you use in your S3 operations belong to the AWS accounts that you expect.

Most S3 operations read from or write to specific S3 buckets. These operations include uploading, copying, and downloading objects, retrieving or modifying bucket configurations, and retrieving or modifying object configurations. When you perform these operations, you specify the bucket that you want to use by including its name with the request. For example, to retrieve an object from S3, you make a request that specifies the name of a bucket and the object key to retrieve from that bucket.

Because Amazon S3 identifies buckets based on their names, an application that uses an incorrect bucket name in a request could inadvertently perform operations against a different bucket than expected. To help avoid unintentional bucket interactions in situations like this, you can use *bucket owner condition*. Bucket owner condition enables you to verify that the target bucket is owned by the expected AWS account, providing an additional layer of assurance that your S3 operations are having the effects you intend.

Topics

- [When to use bucket owner condition \(p. 598\)](#)
- [Verifying a bucket owner \(p. 599\)](#)
- [Examples \(p. 599\)](#)
- [Restrictions and limitations \(p. 601\)](#)

When to use bucket owner condition

We recommend using bucket owner condition whenever you perform a supported S3 operation and know the account ID of the expected bucket owner. Bucket owner condition is available for all S3 object operations and most S3 bucket operations. For a list of S3 operations that don't support bucket owner condition, see [Restrictions and limitations \(p. 601\)](#).

To see the benefit of using bucket owner condition, consider the following scenario involving AWS customer Bea:

1. Bea develops an application that uses Amazon S3. During development, Bea uses her testing-only AWS account to create a bucket named `bea-data-test`, and configures her application to make requests to `bea-data-test`.

2. Bea deploys her application, but forgets to reconfigure the application to use a bucket in her production AWS account.
3. In production, Bea's application makes requests to `bea-data-test`, which succeed. This results in production data being written to the bucket in Bea's test account.

Bea can help protect against situations like this by using bucket owner condition. With bucket owner condition, Bea can include the AWS account ID of the expected bucket owner in her requests. Amazon S3 then checks the account ID of the bucket owner before processing each request. If the actual bucket owner doesn't match the expected bucket owner, the request fails.

If Bea uses bucket owner condition, the scenario described earlier won't result in Bea's application inadvertently writing to a test bucket. Instead, the requests that her application makes at step 3 will fail with an `Access Denied` error message. By using bucket owner condition, Bea helps eliminate the risk of accidentally interacting with buckets in the wrong AWS account.

Verifying a bucket owner

To use bucket owner condition, you include a parameter with your request that specifies the expected bucket owner. Most S3 operations involve only a single bucket, and require only this single parameter to use bucket owner condition. For `CopyObject` operations, this first parameter specifies the expected owner of the destination bucket, and you include a second parameter to specify the expected owner of the source bucket.

When you make a request that includes a bucket owner condition parameter, S3 checks the account ID of the bucket owner against the specified parameter before processing the request. If the parameter matches the bucket owner's account ID, S3 processes the request. If the parameter doesn't match the bucket owner's account ID, the request fails with an `Access Denied` error message.

You can use bucket owner condition with the AWS Command Line Interface (AWS CLI), AWS SDKs, and Amazon S3 REST APIs. When using bucket owner condition with the AWS CLI and Amazon S3 REST APIs, use the following parameter names.

Access method	Parameter for non-copy operations	Copy operation source parameter	Copy operation destination parameter
AWS CLI	<code>--expected-bucket-owner</code>	<code>--expected-source-bucket-owner</code>	<code>--expected-bucket-owner</code>
Amazon S3 REST APIs	<code>x-amz-expected-bucket-owner</code> header	<code>x-amz-source-expected-bucket-owner</code> header	<code>x-amz-expected-bucket-owner</code> header

The parameter names that are required to use bucket owner condition with the AWS SDKs vary depending on the language. To determine the required parameters, see the SDK documentation for your desired language. You can find the SDK documentation at [Tools to Build on AWS](#).

Examples

The following examples show how you can implement bucket owner condition in Amazon S3 using the AWS CLI or the AWS SDK for Java 2.x.

Example

Example: Upload an object

The following example uploads an object to S3 bucket **DOC-EXAMPLE-BUCKET1**, using bucket owner condition to ensure that **DOC-EXAMPLE-BUCKET1** is owned by AWS account 111122223333.

AWS CLI

```
aws s3api put-object \  
    --bucket DOC-EXAMPLE-BUCKET1 --key exampleobject --  
body example_file.txt \  
    --expected-bucket-owner 111122223333
```

AWS SDK for Java 2.x

```
public void putObjectExample() {  
    S3Client s3Client = S3Client.create();  
    PutObjectRequest request = PutObjectRequest.builder()  
        .bucket("DOC-EXAMPLE-BUCKET1")  
        .key("exampleobject")  
        .expectedBucketOwner("111122223333")  
        .build();  
    Path path = Paths.get("example_file.txt");  
    s3Client.putObject(request, path);  
}
```

Example

Example: Copy an object

The following example copies the object **object1** from S3 bucket **DOC-EXAMPLE-BUCKET1** to S3 bucket **DOC-EXAMPLE-BUCKET2**. It uses bucket owner condition to ensure that the buckets are owned by the expected accounts according to the following table.

Bucket	Expected owner
DOC-EXAMPLE-BUCKET1	111122223333
DOC-EXAMPLE-BUCKET2	444455556666

AWS CLI

```
aws s3api copy-object --copy-source DOC-EXAMPLE-BUCKET1/object1 \  
    --bucket DOC-EXAMPLE-BUCKET2 --key object1copy \  
    --expected-source-bucket-owner 111122223333 --expected-  
bucket-owner 444455556666
```

AWS SDK for Java 2.x

```
public void copyObjectExample() {  
    S3Client s3Client = S3Client.create();  
    CopyObjectRequest request = CopyObjectRequest.builder()  
        .copySource("DOC-EXAMPLE-BUCKET1/object1")  
        .destinationBucket("DOC-EXAMPLE-BUCKET2")  
        .destinationKey("object1copy")  
        .expectedSourceBucketOwner("111122223333")  
        .expectedBucketOwner("444455556666")  
        .build();  
    s3Client.copyObject(request);
```

```
}
```

Example

Example: Retrieve a bucket policy

The following example retrieves the access policy for S3 bucket `DOC-EXAMPLE-BUCKET1`, using bucket owner condition to ensure that `DOC-EXAMPLE-BUCKET1` is owned by AWS account 111122223333.

AWS CLI

```
aws s3api get-bucket-policy --bucket DOC-EXAMPLE-BUCKET1 --expected-bucket-owner 111122223333
```

AWS SDK for Java 2.x

```
public void getBucketPolicyExample() {
    S3Client s3Client = S3Client.create();
    GetBucketPolicyRequest request = GetBucketPolicyRequest.builder()
        .bucket("DOC-EXAMPLE-BUCKET1")
        .expectedBucketOwner("111122223333")
        .build();
    try {
        GetBucketPolicyResponse response = s3Client.getBucketPolicy(request);
    }
    catch (S3Exception e) {
        // The call was transmitted successfully, but Amazon S3 couldn't process
        // it, so it returned an error response.
        e.printStackTrace();
    }
}
```

Restrictions and limitations

Amazon S3 bucket owner condition has the following restrictions and limitations:

- The value of the bucket owner condition parameter must be an AWS account ID (12-digit alphanumeric string). Service principals aren't supported.
- Bucket owner condition isn't available for [CreateBucket](#), [ListBuckets](#), or any of the operations included in [AWS S3 Control](#). Amazon S3 ignores any bucket owner condition parameters included with requests to these operations.
- Bucket owner condition only verifies that the account specified in the verification parameter owns the bucket. Bucket owner condition doesn't check the configuration of the bucket. It also doesn't guarantee that the bucket's configuration meets any specific conditions or matches any past state.

Controlling ownership of objects and disabling ACLs for your bucket

S3 Object Ownership is an Amazon S3 bucket-level setting that you can use to disable [access control lists \(ACLs\)](#) (p. 554) and take ownership of every object in your bucket, simplifying access management for data stored in Amazon S3. By default, when another AWS account uploads an object to your S3 bucket, that account (the object writer) owns the object, has access to it, and can grant other users access to it

through ACLs. You can use Object Ownership to change this default behavior. With Object Ownership, ACLs are disabled, and you, as the bucket owner, automatically own every object in your bucket. As a result, access control for your data is based on policies, such as IAM policies, S3 bucket policies, virtual private cloud (VPC) endpoint policies, and AWS Organizations service control policies (SCPs).

A majority of modern use cases in Amazon S3 no longer require the use of ACLs, and we recommend that you disable ACLs except in unusual circumstances where you need to control access for each object individually. With Object Ownership, you can disable ACLs and rely on policies for access control. When you disable ACLs, you can easily maintain a bucket with objects uploaded by different AWS accounts. You, as the bucket owner, own all the objects in the bucket and can manage access to them using policies.

Object Ownership has three settings that you can use to control ownership of objects uploaded to your bucket and to disable or enable ACLs:

ACLs disabled

- **Bucket owner enforced (recommended)** – ACLs are disabled, and the bucket owner automatically owns and has full control over every object in the bucket. ACLs no longer affect permissions to data in the S3 bucket. The bucket uses policies to define access control.

ACLs enabled

- **Bucket owner preferred** – The bucket owner owns and has full control over new objects that other accounts write to the bucket with the `bucket-owner-full-control` canned ACL.
- **Object writer (default)** – The AWS account that uploads an object owns the object, has full control over it, and can grant other users access to it through ACLs.

For the majority of modern use cases in S3, we recommend that you disable ACLs by choosing the bucket owner enforced setting and use your bucket policy to share data with users outside of your account as needed. This approach simplifies permissions management and auditing. You can disable ACLs on both newly created and already existing buckets. In the case of an existing bucket that already has objects in it, after you disable ACLs, the object and bucket ACLs are no longer part of an access evaluation, and access is granted or denied on the basis of policies. For existing buckets, you can re-enable ACLs at any time after you disable them, and your preexisting bucket and object ACLs are restored.

Before you disable ACLs, we recommend that you review your bucket policy to ensure that it covers all the ways that you intend to grant access to your bucket outside of your account. You must reset your bucket ACL to the default (full control to the bucket owner). After you disable ACLs, your bucket accepts only PUT requests that do not specify an ACL or PUT requests with bucket owner full control ACLs, such as the `bucket-owner-full-control` canned ACL or equivalent forms of this ACL expressed in XML. Existing applications that support bucket owner full control ACLs see no impact. PUT requests that contain other ACLs (for example, custom grants to certain AWS accounts) fail and return a 400 error with the error code `AccessControlListNotSupported`.

In contrast, a bucket with the bucket owner preferred setting continues to accept and honor bucket and object ACLs. With this setting, new objects that are written with the `bucket-owner-full-control` canned ACL are automatically owned by the bucket owner rather than the object writer. All other ACL behaviors remain in place. To require all Amazon S3 PUT operations to include the `bucket-owner-full-control` canned ACL, you can [add a bucket policy \(p. 623\)](#) that allows only object uploads using this ACL.

Object Ownership settings

This table shows the impact that each Object Ownership setting has on ACLs, objects, object ownership, and object uploads.

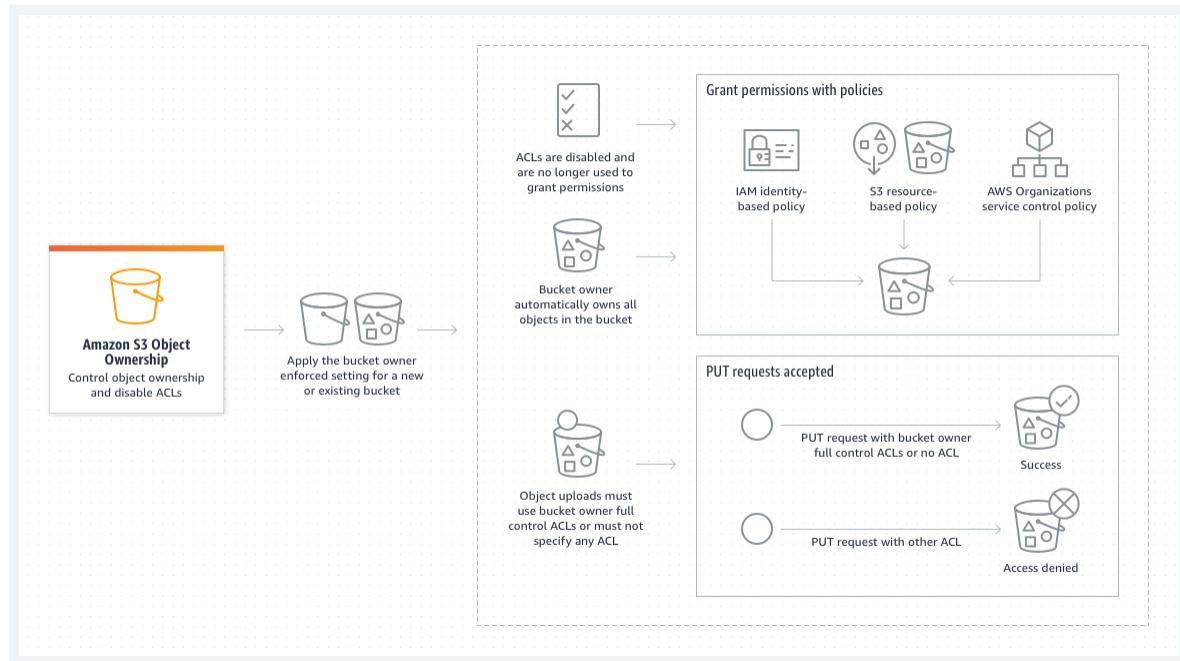
Setting	Applies to	Effect on object ownership	Effect on ACLs	Uploads accepted
Bucket owner enforced (recommended)	All new and existing objects	Bucket owner owns every object.	ACLs are disabled and no longer affect access permissions to your bucket. Requests to set or update ACLs fail. However, requests to read ACLs are supported. Bucket owner has full ownership and control. Object writer no longer has full ownership and control.	Uploads with bucket owner full control ACLs or uploads that don't specify an ACL
Bucket owner preferred	New objects	If an object upload includes the <code>bucket-owner-full-control</code> canned ACL, the bucket owner owns the object. Objects uploaded with other ACLs are owned by the writing account.	ACLs can be updated and can grant permissions. If an object upload includes the <code>bucket-owner-full-control</code> canned ACL, the bucket owner has full control access, and the object writer no longer has full control access.	All uploads
Object writer (default)	New objects	Object writer owns the object.	ACLs can be updated and can grant permissions. Object writer has full control access.	All uploads

Changes introduced by disabling ACLs

When you apply the bucket owner enforced setting for Object Ownership to disable ACLs, you automatically own and take full control over every object in the bucket without taking any additional actions. After you apply this setting, you will see three changes:

- All bucket ACLs and object ACLs are disabled, which gives full access to you, as the bucket owner. When you perform a read ACL request on your bucket or object, you will see that full access is given only to the bucket owner.
- You, as the bucket owner, automatically own and have full control over every object in your bucket.

- ACLs no longer affect access permissions to your bucket. As a result, access control for your data is based on policies, such as IAM policies, S3 bucket policies, VPC endpoint policies, and Organizations SCPs.



If you use S3 Versioning, the bucket owner owns and has full control over all object versions in your bucket. Applying the bucket owner enforced setting does not add a new version of an object.

New objects can be uploaded to your bucket only if they use bucket owner full control ACLs or don't specify an ACL. Object uploads fail if they specify any other ACL. For more information, see [Troubleshooting \(p. 624\)](#).

Because the following example `PutObject` operation using the AWS Command Line Interface (AWS CLI) includes the `bucket-owner-full-control` canned ACL, the object can be uploaded to a bucket with disabled ACLs.

```
aws s3api put-object --bucket DOC-EXAMPLE-BUCKET --key key-name --body path-to-file --acl bucket-owner-full-control
```

Because the following `PutObject` operation doesn't specify an ACL, it also succeeds for a bucket with disabled ACLs.

```
aws s3api put-object --bucket DOC-EXAMPLE-BUCKET --key key-name --body path-to-file
```

Note

If other AWS accounts need access to objects after uploading, you must grant additional permissions to those accounts through bucket policies. For more information, see [Example walkthroughs: Managing access to your Amazon S3 resources \(p. 524\)](#).

Re-enabling ACLs

You can re-enable ACLs by changing from the bucket owner enforced setting to another Object Ownership setting at any time. If you used object ACLs for permissions management before you applied the bucket owner enforced setting and you didn't migrate these object ACL permissions to your bucket

policy, after you re-enable ACLs, these permissions are restored. Additionally, objects written to the bucket while the bucket owner enforced setting was applied are still owned by the bucket owner.

For example, if you change from the bucket owner enforced setting back to object writer, you, as the bucket owner, no longer own and have full control over objects that were previously owned by other AWS accounts. Instead, the uploading accounts again own these objects. Objects owned by other accounts use ACLs for permissions, so you can't use policies to grant permissions to these objects. However, you, as the bucket owner, still own any objects that were written to the bucket while the bucket owner enforced setting was applied. These objects are not owned by the object writer, even if you re-enable ACLs.

Prerequisites for disabling ACLs

Before you disable ACLs for an existing bucket, complete the following prerequisites.

Review bucket and object ACLs and migrate ACL permissions

When you disable ACLs, permissions granted by bucket and object ACLs no longer affect access.

Before you disable ACLs, review your bucket and object ACLs. If your bucket ACLs grant read or write permissions to others outside of your account, you must migrate these permissions to your bucket policy before you can apply the bucket owner enforced setting. If you don't migrate bucket ACLs that grant read or write access outside of your account, your request to apply the bucket owner enforced setting fails and returns the [InvalidBucketAclWithObjectOwnership \(p. 624\)](#) error code.

For example, if you want to disable ACLs for a bucket that receives server access logs, you must migrate the bucket ACL permissions for the S3 log delivery group to the logging service principal in a bucket policy. For more information, see [Grant access to S3 log delivery group for server access logging \(p. 612\)](#).

If you want the object writer to maintain full control of the object they upload, object writer is the best Object Ownership setting for your use case. If you want to control access at the individual object level, bucket owner preferred is the best choice. These use cases are uncommon.

To review ACLs and migrate ACL permissions to bucket policies, see [Prerequisites for disabling ACLs \(p. 607\)](#).

Review and update bucket policies that use ACL-related condition keys

After you apply the bucket owner enforced setting to disable ACLs, new objects can be uploaded to your bucket only if the request uses bucket owner full control ACLs or doesn't specify an ACL. Before disabling ACLs, review your bucket policy for ACL-related condition keys.

If your bucket policy uses an ACL-related condition key to require the `bucket-owner-full-control` canned ACL (for example, `s3:x-amz-acl`), you don't need to update your bucket policy. The following bucket policy uses the `s3:x-amz-acl` to require the `bucket-owner-full-control` canned ACL for S3 `PutObject` requests. This policy *still* requires the object writer to specify the `bucket-owner-full-control` canned ACL. However, buckets with ACLs disabled still accept this ACL, so requests continue to succeed with no client-side changes required.

```
{  
    "Version": "2012-10-17",  
    "Statement": [  
        {  
            "Sid": "Only allow writes to my bucket with bucket owner full control",  
            "Effect": "Allow",  
            "Principal": {  
                "AWS": [  
                    "arn:aws:iam::111122223333:user/ExampleUser"  
                ]  
            }  
        }  
    ]  
}
```

```
        ],
    },
    "Action": [
        "s3:PutObject"
    ],
    "Resource": "arn:aws:s3:::DOC-EXAMPLE-BUCKET/*",
    "Condition": {
        "StringEquals": {
            "s3:x-amz-acl": "bucket-owner-full-control"
        }
    }
}
]
```

However, if your bucket policy uses an ACL-related condition key that requires a different ACL, you must remove this condition key. This example bucket policy requires the public-read ACL for S3 PutObject requests and therefore must be updated before disabling ACLs.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "Only allow writes to my bucket with public read access",
            "Effect": "Allow",
            "Principal": {
                "AWS": [
                    "arn:aws:iam::11122223333:user/ExampleUser" ] },
            "Action": [
                "s3:PutObject"
            ],
            "Resource": "arn:aws:s3:::DOC-EXAMPLE-BUCKET/*",
            "Condition": {
                "StringEquals": {
                    "s3:x-amz-acl": "public-read"
                }
            }
        }
    ]
}
```

Object Ownership permissions

To apply, update, or delete an Object Ownership setting for a bucket, you need the `s3:PutBucketOwnershipControls` permission. To return the Object Ownership setting for a bucket, you need the `s3:GetBucketOwnershipControls` permission. For more information, see [Setting Object Ownership when you create a bucket \(p. 616\)](#) and [Viewing the Object Ownership setting for an S3 bucket \(p. 621\)](#).

Disabling ACLs for all new buckets

You can require that all new buckets are created with ACLs disabled by using IAM or Organizations policies. You can use the `s3:x-amz-object-ownership` condition key in an IAM or Organizations policy to require the bucket owner enforced setting for Object Ownership on all newly created buckets. By requiring the bucket owner enforced setting, you ensure that ACLs are disabled for all new buckets in your account or organization. For more information, see [Disabling ACLs for all new buckets \(bucket owner enforced\) \(p. 622\)](#).

Replication and Object Ownership

When you use S3 replication and the source and destination buckets are owned by different AWS accounts, you can disable ACLs (with the bucket owner enforced setting for Object Ownership) to change replica ownership to the AWS account that owns the destination bucket. This setting mimics the existing owner override behavior without the need of the `s3:ObjectOwnerOverrideToBucketOwner` permission. All objects that are replicated to the destination bucket with the bucket owner enforced setting are owned by the destination bucket owner. For more information about the owner override option for replication configurations, see [Changing the replica owner \(p. 812\)](#).

If you use the default object writer setting for Object Ownership or apply the bucket owner preferred setting for the destination bucket, you can use the Amazon S3 Replication owner override option to transfer ownership of replicated objects to the destination bucket owner.

Setting Object Ownership

You can apply an Object Ownership setting using the S3 console, AWS CLI, AWS SDKs, Amazon S3 REST API, or AWS CloudFormation. The following REST API and AWS CLI commands support Object Ownership:

REST API	AWS CLI	Description
PutBucketOwnershipControls	<code>put-bucket-ownership-controls</code>	Creates or modifies the Object Ownership setting for an existing S3 bucket.
CreateBucket	<code>create-bucket</code>	Creates a bucket using the <code>x-amz-object-ownership</code> request header to specify the Object Ownership setting.
GetBucketOwnershipControls	<code>get-bucket-ownership-controls</code>	Retrieves the Object Ownership setting for an Amazon S3 bucket.
DeleteBucketOwnershipControls	<code>delete-bucket-ownership-controls</code>	Deletes the Object Ownership setting for an Amazon S3 bucket.

For more information about applying and working with Object Ownership settings, see the following topics.

Topics

- [Prerequisites for disabling ACLs \(p. 607\)](#)
- [Setting Object Ownership when you create a bucket \(p. 616\)](#)
- [Setting Object Ownership on an existing bucket \(p. 618\)](#)
- [Viewing the Object Ownership setting for an S3 bucket \(p. 621\)](#)
- [Disabling ACLs for all new buckets and enforcing Object Ownership \(p. 622\)](#)
- [Troubleshooting \(p. 624\)](#)

Prerequisites for disabling ACLs

If your bucket ACL grants access outside of your AWS account, before you disable ACLs, you must migrate your bucket ACL permissions to your bucket policy and reset your bucket ACL to the default private ACL. If you don't migrate these bucket ACLs, your request to apply the bucket owner enforced setting to

disable ACLs fails and returns the [InvalidBucketAclWithObjectOwnership \(p. 624\)](#) error code. We also recommend that you review your object ACL permissions and migrate them to your bucket policy. For more information about other suggested prerequisites, see [Prerequisites for disabling ACLs \(p. 605\)](#).

Each of your existing bucket and object ACLs has an equivalent in an IAM policy. The following bucket policy examples show you how READ and WRITE permissions for bucket and object ACLs map to IAM permissions. For more information about how each ACL translates to IAM permissions, see [Mapping of ACL permissions and access policy permissions \(p. 558\)](#).

To review and migrate ACL permissions to bucket policies, see the following topics.

Topics

- [Bucket policies examples \(p. 608\)](#)
- [Using the S3 console to review and migrate ACL permissions \(p. 609\)](#)
- [Using the AWS CLI to review and migrate ACL permissions \(p. 610\)](#)
- [Example walkthroughs \(p. 611\)](#)

Bucket policies examples

These example bucket policies show you how to migrate READ and WRITE bucket and object ACL permissions for a third-party AWS account to a bucket policy. READ_ACP and WRITE_ACP ACLs are less relevant for policies because they grant ACL-related permissions (s3:GetBucketAcl, s3:GetObjectAcl, s3:PutBucketAcl, and s3:PutObjectAcl).

Example – READ ACL for a bucket

If your bucket had a READ ACL that grants AWS account 111122223333 permission to list the contents of your bucket, you can write a bucket policy that grants s3>ListBucket, s3>ListBucketVersions, s3>ListBucketMultipartUploads permissions for your bucket.

```
{  
    "Version": "2012-10-17",  
    "Statement": [  
        {  
            "Sid": "Permission to list the objects in a bucket",  
            "Effect": "Allow",  
            "Principal": {  
                "AWS": [  
                    "arn:aws:iam::111122223333:root"  
                ]  
            },  
            "Action": [  
                "s3>ListBucket",  
                "s3>ListBucketVersions",  
                "s3>ListBucketMultipartUploads"  
            ],  
            "Resource": "arn:aws:s3:::DOC-EXAMPLE-BUCKET"  
        }  
    ]  
}
```

Example – READ ACLs for every object in a bucket

If every object in your bucket has a READ ACL that grants access to AWS account 111122223333, you can write a bucket policy that grants s3GetObject and s3GetObjectVersion permissions to this account for every object in your bucket.

```
{  
    "Version": "2012-10-17",
```

```
"Statement": [
    {
        "Sid": "Read permission for every object in a bucket",
        "Effect": "Allow",
        "Principal": {
            "AWS": [
                "arn:aws:iam::111122223333:root"
            ]
        },
        "Action": [
            "s3:GetObject",
            "s3:GetObjectVersion"
        ],
        "Resource": "arn:aws:s3:::DOC-EXAMPLE-BUCKET/*"
    }
]
```

This example resource element grants access to a specific object.

```
"Resource": "arn:aws:s3:::DOC-EXAMPLE-BUCKET/OBJECT-KEY"
```

Example – Write ACL that grants permissions to write objects to a bucket

If your bucket has a write ACL that grants AWS account 111122223333 permission to write objects to your bucket, you can write a bucket policy that grants s3:PutObject permission for your bucket.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "Permission to write objects to a bucket",
            "Effect": "Allow",
            "Principal": {
                "AWS": [
                    "arn:aws:iam::111122223333:root"
                ]
            },
            "Action": [
                "s3:PutObject"
            ],
            "Resource": "arn:aws:s3:::DOC-EXAMPLE-BUCKET/*"
        }
    ]
}
```

Using the S3 console to review and migrate ACL permissions

To review ACL permissions

1. Sign in to the AWS Management Console and open the Amazon S3 console at <https://console.aws.amazon.com/s3/>.
2. To view bucket ACL permissions, in the **Buckets** list, choose the bucket name.
3. Choose **Permissions**.
4. Under **Access control list (ACL)**, review your bucket ACL permissions.
5. Choose the **Objects** tab.
6. In the **Objects** list, choose your object name.
7. Choose **Permissions**.
8. Under **Access control list (ACL)**, review your object ACL permissions.

To migrate ACL permissions and update your bucket ACL

1. Sign in to the AWS Management Console and open the Amazon S3 console at <https://console.aws.amazon.com/s3/>.
2. In the **Buckets** list, choose the bucket name.
3. On the **Permissions** tab, under **Bucket policy**, choose **Edit**.
4. In the **Policy** box, add or update your bucket policy.

For example bucket policies, see [Bucket policies examples \(p. 608\)](#) and [Example walkthroughs \(p. 611\)](#).
5. Choose **Save changes**.
6. [Update your bucket ACL \(p. 562\)](#) to remove ACL grants to other groups or AWS accounts.
7. [Apply the bucket owner enforced setting \(p. 618\)](#) for Object Ownership.

Using the AWS CLI to review and migrate ACL permissions

1. To return the bucket ACL for your bucket, use the `get-bucket-acl` AWS CLI command:

```
aws s3api get-bucket-acl --bucket DOC-EXAMPLE-BUCKET
```

For example, this bucket ACL grants `WRITE` and `READ` access to a third-party account. In this ACL, the third-party account is identified by the [canonical user ID \(p. 561\)](#). To apply the bucket owner enforced setting and disable ACLs, you must migrate these permissions for the third-party account to a bucket policy.

```
{  
    "Owner": {  
        "DisplayName": "DOC-EXAMPLE-ACCOUNT-OWNER",  
        "ID": "852b113e7a2f25102679df27bb0ae12b3f85be6BucketOwnerCanonicalUserID"  
    },  
    "Grants": [  
        {  
            "Grantee": {  
                "DisplayName": "DOC-EXAMPLE-ACCOUNT-OWNER",  
                "ID": "852b113e7a2f25102679df27bb0ae12b3f85be6BucketOwnerCanonicalUserID",  
                "Type": "CanonicalUser"  
            },  
            "Permission": "FULL_CONTROL"  
        },  
        {  
            "Grantee": {  
                "DisplayName": "THIRD-PARTY-EXAMPLE-ACCOUNT",  
                "ID": "72806de9d1ae8b171cca9e2494a8d1335dfced4ThirdPartyAccountCanonicalUserID",  
                "Type": "CanonicalUser"  
            },  
            "Permission": "READ"  
        },  
        {  
            "Grantee": {  
                "DisplayName": "THIRD-PARTY-EXAMPLE-ACCOUNT",  
                "ID": "72806de9d1ae8b171cca9e2494a8d1335dfced4ThirdPartyAccountCanonicalUserID",  
                "Type": "CanonicalUser"  
            },  
            "Permission": "WRITE"  
        }  
    ]
```

```
    ]  
}
```

For other example ACLs, see [Example walkthroughs \(p. 611\)](#).

2. Migrate your bucket ACL permissions to a bucket policy:

This example bucket policy grants s3:PutObject and s3>ListBucket permissions for a third-party account. In the bucket policy, the third-party account is identified by the AWS account ID ([111122223333](#)).

```
aws s3api put-bucket-policy --bucket DOC-EXAMPLE-BUCKET --policy file://policy.json  
  
policy.json:  
{  
    "Version": "2012-10-17",  
    "Statement": [  
        {  
            "Sid": "PolicyForCrossAccountAllowUpload",  
            "Effect": "Allow",  
            "Principal": {  
                "AWS": [  
                    "arn:aws:iam::111122223333:root"  
                ]  
            },  
            "Action": [  
                "s3:PutObject",  
                "s3>ListBucket"  
            ],  
            "Resource": [  
                "arn:aws:s3:::DOC-EXAMPLE-BUCKET",  
                "arn:aws:s3:::DOC-EXAMPLE-BUCKET/*"  
            ]  
        }  
    ]  
}
```

For more example bucket policies, see [Bucket policies examples \(p. 608\)](#) and [Example walkthroughs \(p. 611\)](#).

3. To return the ACL for a specific object, use the [get-object-acl](#) AWS CLI command.

```
aws s3api get-object-acl --bucket DOC-EXAMPLE-BUCKET --key EXAMPLE-OBJECT-KEY
```

4. If required, migrate object ACL permissions to your bucket policy.

This example resource element grants access to a specific object in a bucket policy.

```
"Resource": "arn:aws:s3:::DOC-EXAMPLE-BUCKET/EXAMPLE-OBJECT-KEY"
```

5. Reset the ACL for your bucket to the default ACL.

```
aws s3api put-bucket-acl --bucket DOC-EXAMPLE-BUCKET --acl private
```

6. [Apply the bucket owner enforced setting \(p. 618\)](#) for Object Ownership.

Example walkthroughs

The following examples show you how to migrate ACL permissions to bucket policies for specific use cases.

Topics

- [Grant access to S3 log delivery group for server access logging \(p. 612\)](#)
- [Grant public read access to the objects in a bucket \(p. 613\)](#)
- [Grant Amazon ElastiCache for Redis access to your S3 bucket \(p. 614\)](#)

Grant access to S3 log delivery group for server access logging

If you want to apply the bucket owner enforced setting to disable ACLs for a server access logging target bucket, you must migrate bucket ACL permissions for the S3 log delivery group to the logging service principal (`logging.s3.amazonaws.com`) in a bucket policy. For more information about log delivery permissions, see [Permissions for log delivery \(p. 981\)](#).

This bucket ACL grants `WRITE` and `READ_ACP` access to the S3 log delivery group:

```
{  
    "Owner": {  
        "DisplayName": "DOC-EXAMPLE-ACCOUNT-OWNER",  
        "ID": "852b113e7a2f25102679df27bb0ae12b3f85be6BucketOwnerCanonicalUserID"  
    },  
    "Grants": [  
        {  
            "Grantee": {  
                "Type": "CanonicalUser",  
                "DisplayName": "DOC-EXAMPLE-ACCOUNT-OWNER",  
                "ID": "852b113e7a2f25102679df27bb0ae12b3f85be6BucketOwnerCanonicalUserID"  
            },  
            "Permission": "FULL_CONTROL"  
        },  
        {  
            "Grantee": {  
                "Type": "Group",  
                "URI": "http://acs.amazonaws.com/groups/s3/LogDelivery"  
            },  
            "Permission": "WRITE"  
        },  
        {  
            "Grantee": {  
                "Type": "Group",  
                "URI": "http://acs.amazonaws.com/groups/s3/LogDelivery"  
            },  
            "Permission": "READ_ACP"  
        }  
    ]  
}
```

To migrate bucket ACL permissions for the S3 log delivery group to the logging service principal in a bucket policy

1. Add the following bucket policy to your target bucket, replacing the example values.

```
aws s3api put-bucket-policy --bucket DOC-EXAMPLE-BUCKET --policy file://policy.json  
  
policy.json: {  
    {  
        "Version": "2012-10-17",  
        "Statement": [  
            {  
                "Sid": "S3ServerAccessLogsPolicy",  
                "Effect": "Allow",  
                "Principal": {
```

```
        "Service": "logging.s3.amazonaws.com"
    },
    "Action": [
        "s3:PutObject"
    ],
    "Resource": "arn:aws:s3:::DOC-EXAMPLE-BUCKET/EXAMPLE-LOGGING-PREFIX*",
    "Condition": {
        "ArnLike": {
            "aws:SourceArn": "arn:aws:s3:::SOURCE-BUCKET-NAME"
        },
        "StringEquals": {
            "aws:SourceAccount": "SOURCE-AWS-ACCOUNT-ID"
        }
    }
]
}
```

2. Reset the ACL for your target bucket to the default ACL.

```
aws s3api put-bucket-acl --bucket DOC-EXAMPLE-BUCKET --acl private
```

3. Apply the bucket owner enforced setting (p. 618) for Object Ownership to your target bucket.

Grant public read access to the objects in a bucket

If your object ACLs grant public read access to all of the objects in your bucket, you can migrate these ACL permissions to a bucket policy.

This object ACL grants public read access to an object in a bucket:

```
{
    "Owner": {
        "DisplayName": "DOC-EXAMPLE-ACCOUNT-OWNER",
        "ID": "852b113e7a2f25102679df27bb0ae12b3f85be6BucketOwnerCanonicalUserID"
    },
    "Grants": [
        {
            "Grantee": {
                "DisplayName": "DOC-EXAMPLE-ACCOUNT-OWNER",
                "ID": "852b113e7a2f25102679df27bb0ae12b3f85be6BucketOwnerCanonicalUserID",
                "Type": "CanonicalUser"
            },
            "Permission": "FULL_CONTROL"
        },
        {
            "Grantee": {
                "Type": "Group",
                "URI": "http://acs.amazonaws.com/groups/global/AllUsers"
            },
            "Permission": "READ"
        }
    ]
}
```

To migrate public read ACL permissions to a bucket policy

1. To grant public read access to all of the objects in your bucket, add the following bucket policy, replacing the example values.

```
aws s3api put-bucket-policy --bucket DOC-EXAMPLE-BUCKET --policy file://policy.json
```

```
policy.json:
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "PublicReadGetObject",
            "Effect": "Allow",
            "Principal": "*",
            "Action": [
                "s3:GetObject"
            ],
            "Resource": [
                "arn:aws:s3:::DOC-EXAMPLE-BUCKET/*"
            ]
        }
    ]
}
```

To grant public access to a specific object in a bucket policy, use the following format for the **Resource** element.

```
"Resource": "arn:aws:s3:::DOC-EXAMPLE-BUCKET/OBJECT-KEY"
```

To grant public access to all of the objects with a specific prefix, use the following format for the **Resource** element.

```
"Resource": "arn:aws:s3:::DOC-EXAMPLE-BUCKET/PREFIX/*"
```

2. Apply the bucket owner enforced setting (p. 618) for Object Ownership.

Grant Amazon ElastiCache for Redis access to your S3 bucket

You can [export your ElastiCache for Redis backup](#) to an S3 bucket, which gives you access to the backup from outside ElastiCache. To export your backup to an S3 bucket, you must grant ElastiCache permissions to copy a snapshot to the bucket. If you've granted permissions to ElastiCache in a bucket ACL, you must migrate these permissions to a bucket policy before you apply the bucket owner enforced setting to disable ACLs. For more information, see [Grant ElastiCache access to your Amazon S3 bucket](#) in the *Amazon ElastiCache User Guide*.

The following example shows the bucket ACL permissions that grant permissions to ElastiCache.

```
{
    "Owner": {
        "DisplayName": "DOC-EXAMPLE-ACCOUNT-OWNER",
        "ID": "852b113e7a2f25102679df27bb0ae12b3f85be6BucketOwnerCanonicalUserID"
    },
    "Grants": [
        {
            "Grantee": {
                "DisplayName": "DOC-EXAMPLE-ACCOUNT-OWNER",
                "ID": "852b113e7a2f25102679df27bb0ae12b3f85be6BucketOwnerCanonicalUserID",
                "Type": "CanonicalUser"
            },
            "Permission": "FULL_CONTROL"
        },
        {
            "Grantee": {
                "DisplayName": "aws-scs-s3-readonly",
                "ID": "540804c33a284a299d2547575ce1010f2312ef3da9b3a053c8bc45bf233e4353",
                "Type": "AWS"
            },
            "Permission": "READ"
        }
    ]
}
```

```
        "Type": "CanonicalUser"
    },
    "Permission": "READ"
},
{
    "Grantee": {
        "DisplayName": "aws-scs-s3-readonly",
        "ID": "540804c33a284a299d2547575ce1010f2312ef3da9b3a053c8bc45bf233e4353",
        "Type": "CanonicalUser"
    },
    "Permission": "WRITE"
},
{
    "Grantee": {
        "DisplayName": "aws-scs-s3-readonly",
        "ID": "540804c33a284a299d2547575ce1010f2312ef3da9b3a053c8bc45bf233e4353",
        "Type": "CanonicalUser"
    },
    "Permission": "READ_ACP"
}
]
```

To migrate bucket ACL permissions for ElastiCache for Redis to a bucket policy

1. Add the following bucket policy to your bucket, replacing the example values.

```
aws s3api put-bucket-policy --bucket DOC-EXAMPLE-BUCKET --policy file://policy.json

policy.json:
"Id": "Policy15397346",
"Statement": [
    {
        "Sid": "Stmt15399483",
        "Effect": "Allow",
        "Principal": {
            "Service": "Region.elasticache-snapshot.amazonaws.com"
        },
        "Action": [
            "s3:PutObject",
            "s3:GetObject",
            "s3>ListBucket",
            "s3:GetBucketAcl",
            "s3>ListMultipartUploadParts",
            "s3>ListBucketMultipartUploads"
        ],
        "Resource": [
            "arn:aws:s3:::DOC-EXAMPLE-BUCKET",
            "arn:aws:s3:::DOC-EXAMPLE-BUCKET/*"
        ]
    }
]
```

2. Reset the ACL for your bucket to the default ACL:

```
aws s3api put-bucket-acl --bucket DOC-EXAMPLE-BUCKET --acl private
```

3. [Apply the bucket owner enforced setting \(p. 618\)](#) for Object Ownership.

Setting Object Ownership when you create a bucket

When you create a bucket, you can configure S3 Object Ownership. To set Object Ownership for an existing bucket, see [Setting Object Ownership on an existing bucket \(p. 618\)](#).

S3 Object Ownership is an Amazon S3 bucket-level setting that you can use to disable [access control lists \(ACLs\) \(p. 554\)](#) and take ownership of every object in your bucket, simplifying access management for data stored in Amazon S3. We recommend that you disable ACLs unless you need to control access at the individual object level.

Object Ownership has three settings that you can use to control ownership of objects uploaded to your bucket and to disable or enable ACLs:

ACLs disabled

- **Bucket owner enforced (recommended)** – ACLs are disabled, and the bucket owner automatically owns and has full control over every object in the bucket. ACLs no longer affect permissions to data in the S3 bucket. The bucket uses policies to define access control.

ACLs enabled

- **Bucket owner preferred** – The bucket owner owns and has full control over new objects that other accounts write to the bucket with the `bucket-owner-full-control` canned ACL.
- **Object writer (default)** – The AWS account that uploads an object owns the object, has full control over it, and can grant other users access to it through ACLs.

Permissions: To create a bucket and select a setting for Object Ownership, you must have both the `s3:CreateBucket` and `s3:PutBucketOwnershipControls` permissions. For more information about Amazon S3 permissions, see [Actions, resources, and condition keys for Amazon S3 \(p. 431\)](#).

Using the S3 console

1. Sign in to the AWS Management Console and open the Amazon S3 console at <https://console.aws.amazon.com/s3/>.
2. Choose **Create bucket**.

The **Create bucket** wizard opens.

3. In **Bucket name**, enter a DNS-compliant name for your bucket.

The bucket name must:

- Be unique across all of Amazon S3.
- Be between 3 and 63 characters long.
- Not contain uppercase characters.
- Start with a lowercase letter or number.

After you create the bucket, you cannot change its name. For information about naming buckets, see [Bucket naming rules \(p. 118\)](#).

Important

Avoid including sensitive information, such as account number, in the bucket name. The bucket name is visible in the URLs that point to the objects in the bucket.

4. In **Region**, choose the AWS Region where you want the bucket to reside.

Choose a Region close to you to minimize latency and costs and address regulatory requirements. Objects stored in a Region never leave that Region unless you explicitly transfer them to another Region. For a list of Amazon S3 AWS Regions, see [AWS service endpoints](#) in the *Amazon Web Services General Reference*.

5. Under **Object Ownership**, to disable or enable ACLs and control ownership of objects uploaded in your bucket, choose one of the following settings:

ACLs disabled

- **Bucket owner enforced** – ACLs are disabled, and the bucket owner automatically owns and has full control over every object in the bucket. ACLs no longer affect permissions to data in the S3 bucket. The bucket uses policies to define access control.

To require that all new buckets are created with ACLs disabled by using IAM or AWS Organizations policies, see [Disabling ACLs for all new buckets \(bucket owner enforced\) \(p. 622\)](#).

ACLs enabled

- **Bucket owner preferred** – The bucket owner owns and has full control over new objects that other accounts write to the bucket with the `bucket-owner-full-control` canned ACL.

If you apply the bucket owner preferred setting, to require all Amazon S3 uploads to include the `bucket-owner-full-control` canned ACL, you can [add a bucket policy \(p. 623\)](#) that only allows object uploads that use this ACL.

- **Object writer** – The AWS account that uploads an object owns the object, has full control over it, and can grant other users access to it through ACLs.

Note

To apply the **Bucket owner enforced** setting or the **Bucket owner preferred** setting, you must have the following permission: `s3:CreateBucket` and `s3:PutBucketOwnershipControls`.

6. In **Bucket settings for Block Public Access**, choose the Block Public Access settings that you want to apply to the bucket.

We recommend that you keep all settings enabled unless you know that you need to turn off one or more of them for your use case, such as to host a public website. Block Public Access settings that you enable for the bucket are also enabled for all access points that you create on the bucket. For more information about blocking public access, see [Blocking public access to your Amazon S3 storage \(p. 584\)](#).

7. (Optional) If you want to enable S3 Object Lock, do the following:

- a. Choose **Advanced settings**.

Important

You can only enable S3 Object Lock for a bucket when you create it. If you enable Object Lock for the bucket, you cannot disable it later. Enabling Object Lock also enables versioning for the bucket. After you enable Object Lock for the bucket, you must configure the Object Lock default retention and legal hold settings to protect new objects from being deleted or overwritten. For more information, see [Configuring S3 Object Lock using the console \(p. 684\)](#).

- b. If you want to enable Object Lock, choose **Enable**, read the warning that appears, and acknowledge it.

For more information about the S3 Object Lock feature, see [Using S3 Object Lock \(p. 680\)](#).

Note

To create an Object Lock enabled bucket, you must have the following permissions:
`s3:CreateBucket`, `s3:PutBucketVersioning` and `s3:PutBucketObjectLockConfiguration`.

8. Choose **Create bucket**.

Using the AWS CLI

To set Object Ownership when you create a new bucket, use the `create-bucket` AWS CLI command with the `--object-ownership` parameter.

This example applies the bucket owner enforced setting for a new bucket using the AWS CLI:

```
aws s3api create-bucket --bucket DOC-EXAMPLE-BUCKET --region us-east-1 --object-ownership BucketOwnerEnforced
```

Using the AWS SDK for Java

This example sets the bucket owner enforced setting for a new bucket using the AWS SDK for Java:

```
// Build the ObjectOwnership for CreateBucket
CreateBucketRequest createBucketRequest = CreateBucketRequest.builder()
    .bucket(bucketName)
    .objectOwnership(ObjectOwnership.BucketOwnerEnforced)
    .build()

// Send the request to S3
s3client.createBucket(createBucketRequest);
```

Using AWS CloudFormation

To use the `AWS::S3::Bucket` AWS CloudFormation resource to set Object Ownership when you create a new bucket, see [OwnershipControls within AWS::S3::Bucket](#) in the *AWS CloudFormation User Guide*.

Using the REST API

To apply the bucket owner enforced setting for S3 Object Ownership, use the `CreateBucket` API operation with the `x-amz-object-ownership` request header set to `BucketOwnerEnforced`. For information and examples, see [CreateBucket](#) in the *Amazon Simple Storage Service API Reference*.

Next steps: After you apply the bucket owner enforced or bucket owner preferred settings for Object Ownership, you can further take the following steps:

- [Bucket owner enforced \(p. 622\)](#) – Require that all new buckets are created with ACLs disabled by using an IAM or Organizations policy.
- [Bucket owner preferred \(p. 623\)](#) – Add an S3 bucket policy to require the `bucket-owner-full-control` canned ACL for all object uploads to your bucket.

Setting Object Ownership on an existing bucket

You can configure S3 Object Ownership on an existing S3 bucket. To apply Object Ownership when you create a bucket, see [Setting Object Ownership when you create a bucket \(p. 616\)](#).

S3 Object Ownership is an Amazon S3 bucket-level setting that you can use to disable [access control lists \(ACLs\) \(p. 554\)](#) and take ownership of every object in your bucket, simplifying access management for

data stored in Amazon S3. We recommend that you disable ACLs unless you need to control access at the individual object level.

Object Ownership has three settings that you can use to control ownership of objects uploaded to your bucket and to disable or enable ACLs:

ACLs disabled

- **Bucket owner enforced (recommended)** – ACLs are disabled, and the bucket owner automatically owns and has full control over every object in the bucket. ACLs no longer affect permissions to data in the S3 bucket. The bucket uses policies to define access control.

ACLs enabled

- **Bucket owner preferred** – The bucket owner owns and has full control over new objects that other accounts write to the bucket with the `bucket-owner-full-control` canned ACL.
- **Object writer (default)** – The AWS account that uploads an object owns the object, has full control over it, and can grant other users access to it through ACLs.

Prerequisites: Before you apply the bucket owner enforced setting to disable ACLs, you must migrate bucket ACL permissions to bucket policies and reset your bucket ACLs to the default private ACL. We also recommend that you migrate object ACL permissions to bucket policies and edit bucket policies that require ACLs other than bucket owner full control ACLs. For more information, see [Prerequisites for disabling ACLs \(p. 607\)](#).

Permissions: To use this operation, you must have the `s3:PutBucketOwnershipControls` permission. For more information about Amazon S3 permissions, see [Actions, resources, and condition keys for Amazon S3 \(p. 431\)](#).

Using the S3 console

1. Sign in to the AWS Management Console and open the Amazon S3 console at <https://console.aws.amazon.com/s3/>.
2. In the **Buckets** list, choose the name of the bucket that you want to apply an S3 Object Ownership setting to.
3. Choose the **Permissions** tab.
4. Under **Object Ownership**, choose **Edit**.
5. Under **Object Ownership**, to disable or enable ACLs and control ownership of objects uploaded in your bucket, choose one of the following settings:

ACLs disabled

- **Bucket owner enforced** – ACLs are disabled, and the bucket owner automatically owns and has full control over every object in the bucket. ACLs no longer affect permissions to data in the S3 bucket. The bucket uses policies to define access control.

To require that all new buckets are created with ACLs disabled by using IAM or AWS Organizations policies, see [Disabling ACLs for all new buckets \(bucket owner enforced\) \(p. 622\)](#).

ACLs enabled

- **Bucket owner preferred** – The bucket owner owns and has full control over new objects that other accounts write to the bucket with the `bucket-owner-full-control` canned ACL.

If you apply the bucket owner preferred setting, to require all Amazon S3 uploads to include the `bucket-owner-full-control` canned ACL, you can [add a bucket policy \(p. 623\)](#) that only allows object uploads that use this ACL.

- **Object writer** – The AWS account that uploads an object owns the object, has full control over it, and can grant other users access to it through ACLs.

6. Choose **Save**.

Using the AWS CLI

To apply an Object Ownership setting for an existing bucket, use the `put-bucket-ownership-controls` command with the `--ownership-controls` parameter.

This example applies the bucket owner enforced setting for an existing bucket using the AWS CLI:

```
aws s3api put-bucket-ownership-controls --bucket DOC-EXAMPLE-BUCKET --ownership-controls  
Rules=[{ObjectOwnership=BucketOwnerEnforced}]
```

Using the AWS SDK for Java

This example applies the `BucketOwnerEnforced` setting for Object Ownership on an existing bucket using the AWS SDK for Java:

```
// Build the ObjectOwnership for BucketOwnerEnforced  
OwnershipControlsRule rule = OwnershipControlsRule.builder()  
    .objectOwnership(ObjectOwnership.BucketOwnerEnforced)  
    .build();  
  
OwnershipControls ownershipControls = OwnershipControls.builder()  
    .rules(rule)  
    .build()  
  
// Build the PutBucketOwnershipControlsRequest  
PutBucketOwnershipControlsRequest putBucketOwnershipControlsRequest =  
    PutBucketOwnershipControlsRequest.builder()  
        .bucket(BUCKET_NAME)  
        .ownershipControls(ownershipControls)  
        .build();  
  
// Send the request to S3  
s3client.putBucketOwnershipControls(putBucketOwnershipControlsRequest);
```

Using AWS CloudFormation

To use AWS CloudFormation to apply an Object Ownership setting for an existing bucket, see [AWS::S3::Bucket OwnershipControls](#) in the *AWS CloudFormation User Guide*.

Using the REST API

To use the REST API to apply an Object Ownership setting to an existing S3 bucket, use `PutBucketOwnershipControls`. For more information, see [PutBucketOwnershipControls](#) in the *Amazon Simple Storage Service API Reference*.

Next steps: After you apply the bucket owner enforced or bucket owner preferred settings for Object Ownership, you can further take the following steps:

- [Bucket owner enforced \(p. 622\)](#) – Require that all new buckets are created with ACLs disabled by using an IAM or Organizations policy.

- [Bucket owner preferred \(p. 623\)](#) – Add an S3 bucket policy to require the `bucket-owner-full-control` canned ACL for all object uploads to your bucket.

Viewing the Object Ownership setting for an S3 bucket

S3 Object Ownership is an Amazon S3 bucket-level setting that you can use to disable [access control lists \(ACLs\) \(p. 554\)](#) and take ownership of every object in your bucket, simplifying access management for data stored in Amazon S3. We recommend that you disable ACLs unless you need to control access at the individual object level.

Object Ownership has three settings that you can use to control ownership of objects uploaded to your bucket and to disable or enable ACLs:

ACLs disabled

- **Bucket owner enforced (recommended)** – ACLs are disabled, and the bucket owner automatically owns and has full control over every object in the bucket. ACLs no longer affect permissions to data in the S3 bucket. The bucket uses policies to define access control.

ACLs enabled

- **Bucket owner preferred** – The bucket owner owns and has full control over new objects that other accounts write to the bucket with the `bucket-owner-full-control` canned ACL.
- **Object writer (default)** – The AWS account that uploads an object owns the object, has full control over it, and can grant other users access to it through ACLs.

You can view the S3 Object Ownership settings for an Amazon S3 bucket. To set Object Ownership for a new bucket, see [Setting Object Ownership when you create a bucket \(p. 616\)](#). To set Object Ownership for an existing bucket, see [Setting Object Ownership on an existing bucket \(p. 618\)](#).

Permissions: To use this operation, you must have the `s3:GetBucketOwnershipControls` permission. For more information about Amazon S3 permissions, see [Actions, resources, and condition keys for Amazon S3 \(p. 431\)](#).

Using the S3 console

1. Sign in to the AWS Management Console and open the Amazon S3 console at <https://console.aws.amazon.com/s3/>.
2. In the **Buckets** list, choose the name of the bucket that you want to apply an Object Ownership setting to.
3. Choose the **Permissions** tab.
4. Under **Object Ownership**, you can view the Object Ownership settings for your bucket.

Using the AWS CLI

To retrieve the S3 Object Ownership setting for an S3 bucket, use the `get-bucket-ownership-controls` AWS CLI command.

```
aws s3api get-bucket-ownership-controls --bucket DOC-EXAMPLE-BUCKET
```

Using the REST API

To retrieve the Object Ownership setting for an S3 bucket, use the `GetBucketOwnershipControls` API operation. For more information, see [GetBucketOwnershipControls](#).

Disabling ACLs for all new buckets and enforcing Object Ownership

We recommend that you disable ACLs on your Amazon S3 buckets. You can do this by applying the bucket owner enforced setting for S3 Object Ownership. When you apply this setting, ACLs are disabled and you automatically own and have full control over all objects in your bucket. You can require that all new buckets are created with ACLs disabled by using AWS Identity and Access Management (IAM) policies or AWS Organizations service control policies (SCPs), as described in the next section.

To enforce object ownership for new objects without disabling ACLs, you can apply the bucket owner preferred setting. When you apply this setting, we strongly recommend that you update your bucket policy to require the `bucket-owner-full-control` canned ACL for all PUT requests to your bucket. Clients should also be updated to send the `bucket-owner-full-control` canned ACL to your bucket from other accounts.

Topics

- [Disabling ACLs for all new buckets \(bucket owner enforced\) \(p. 622\)](#)
- [Requiring the `bucket-owner-full-control` canned ACL for Amazon S3 PUT operations \(bucket owner preferred\) \(p. 623\)](#)

Disabling ACLs for all new buckets (bucket owner enforced)

The following example IAM policy denies the `s3:CreateBucket` permission for a specific IAM user or role unless the bucket owner enforced setting is applied for Object Ownership. The key-value pair in the Condition block specifies `s3:x-amz-object-ownership` as its key and the `BucketOwnerEnforced` setting as its value. In other words, the IAM user can create buckets only if they set the bucket owner enforced setting for Object Ownership and disable ACLs. You can also use this policy as a boundary SCP for your AWS organization.

```
{  
    "Version": "2012-10-17",  
    "Statement": [  
        {  
            "Sid": "RequireBucketOwnerFullControl",  
            "Action": "s3:CreateBucket",  
            "Effect": "Deny",  
            "Resource": "*",  
            "Condition": {  
                "StringNotEquals": {  
                    "s3:x-amz-object-ownership": "BucketOwnerEnforced"  
                }  
            }  
        }  
    ]  
}
```

Requiring the bucket-owner-full-control canned ACL for Amazon S3 PUT operations (bucket owner preferred)

With the bucket owner preferred setting for Object Ownership, you, as the bucket owner, own and have full control over new objects that other accounts write to your bucket with the `bucket-owner-full-control` canned ACL. However, if other accounts write objects to your bucket without the `bucket-owner-full-control` canned ACL, the object writer maintains full control access. You, as the bucket owner, can implement a bucket policy that allows writes only if they specify the `bucket-owner-full-control` canned ACL.

Note

If you have ACLs disabled with the bucket owner enforced setting, you, as the bucket owner, automatically own and have full control over all the objects in your bucket. You don't need to use this section to update your bucket policy to enforce object ownership for the bucket owner.

The following bucket policy specifies that account `111122223333` can upload objects to `DOC-EXAMPLE-BUCKET` only when the object's ACL is set to `bucket-owner-full-control`. Be sure to replace `111122223333` with your account and `DOC-EXAMPLE-BUCKET` with the name of your bucket.

```
{  
    "Version": "2012-10-17",  
    "Statement": [  
        {  
            "Sid": "Only allow writes to my bucket with bucket owner full control",  
            "Effect": "Allow",  
            "Principal": {  
                "AWS": [  
                    "arn:aws:iam::111122223333:user/ExampleUser"  
                ]  
            },  
            "Action": [  
                "s3:PutObject"  
            ],  
            "Resource": "arn:aws:s3:::DOC-EXAMPLE-BUCKET/*",  
            "Condition": {  
                "StringEquals": {  
                    "s3:x-amz-acl": "bucket-owner-full-control"  
                }  
            }  
        }  
    ]  
}
```

The following is an example copy operation that includes the `bucket-owner-full-control` canned ACL using the AWS Command Line Interface (AWS CLI).

```
aws s3 cp file.txt s3://DOC-EXAMPLE-BUCKET --acl bucket-owner-full-control
```

After the bucket policy is put in effect, if the client does not include the `bucket-owner-full-control` canned ACL, the operation fails, and the uploader receives the following error:

An error occurred (AccessDenied) when calling the PutObject operation: Access Denied.

Note

If clients need access to objects after uploading, you must grant additional permissions to the uploading account. For information about granting accounts access to your resources, see [Example walkthroughs: Managing access to your Amazon S3 resources \(p. 524\)](#).

Troubleshooting

When you apply the bucket owner enforced setting for S3 Object Ownership, access control lists (ACLs) are disabled and you, as the bucket owner, automatically own all objects in your bucket. ACLs no longer affect permissions for the objects in your bucket. You can use policies to grant permissions. All S3 PUT requests must specify bucket owner full control ACLs or not specify an ACL, or they fail. For more information, see [Controlling ownership of objects and disabling ACLs for your bucket \(p. 601\)](#).

If an invalid ACL is specified or bucket ACL permissions grant access outside of your AWS account, you might see the following error responses.

AccessControlListNotSupported

After you apply the bucket owner enforced setting for Object Ownership, ACLs are disabled. Requests to set ACLs or update ACLs fail with a 400 error and return the AccessControlListNotSupported error code. Requests to read ACLs are still supported. Requests to read ACLs always return a response that shows full control for the bucket owner. In your PUT operations, you must either specify bucket owner full control ACLs or not specify an ACL, or your PUT operations fail.

The following example put-object operation using the AWS CLI includes the public-read canned ACL.

```
aws s3api put-object --bucket DOC-EXAMPLE-BUCKET --key object-key-name --body doc-example-body --acl public-read
```

If the bucket uses the bucket owner enforced setting to disable ACLs, this operation fails, and the uploader receives the following error message:

An error occurred (AccessControlListNotSupported) when calling the PutObject operation: The bucket does not allow ACLs

InvalidBucketAclWithObjectOwnership

If you want to apply the bucket owner enforced setting to disable ACLs, your bucket ACL must give full control only to the bucket owner. Your bucket ACL cannot give access to an external AWS account or any other group. For example, if your CreateBucket request sets bucket owner enforced and specifies a bucket ACL that provides access to an external AWS account, your request fails with a 400 error and returns the InvalidBucketAclWithObjectOwnership error code. Similarly, if your PutBucketOwnershipControls request sets bucket owner enforced on a bucket that has a bucket ACL that grants permissions to others, the request fails.

Example : Existing bucket ACL grants public read access

For example, if an existing bucket ACL grants public read access, you cannot apply the bucket owner enforced setting for Object Ownership until you migrate these ACL permissions to a bucket policy and reset your bucket ACL to the default private ACL. For more information, see [Prerequisites for disabling ACLs \(p. 607\)](#).

This example bucket ACL grants public read access:

```
{  
    "Owner": {  
        "ID": "852b113e7a2f25102679df27bb0ae12b3f85be6BucketOwnerCanonicalUserID"  
    },  
    "Grants": [  
        {  
            "Grantee": {  
                "Type": "Group",  
                "URI": "http://acs.amazonaws.com/groups/global/AllUsers"  
            },  
            "Permission": "Read"  
        }  
    ]  
}
```

```
        "Grantee": {
            "ID": "852b113e7a2f25102679df27bb0ae12b3f85be6BucketOwnerCanonicalUserID",
            "Type": "CanonicalUser"
        },
        "Permission": "FULL_CONTROL"
    },
    {
        "Grantee": {
            "Type": "Group",
            "URI": "http://acs.amazonaws.com/groups/global/AllUsers"
        },
        "Permission": "READ"
    }
]
```

This example `put-bucket-ownership-controls` AWS CLI operation applies the bucket owner enforced setting for Object Ownership:

```
aws s3api put-bucket-ownership-controls --bucket DOC-EXAMPLE-BUCKET --ownership-controls
Rules=[{ObjectOwnership=BucketOwnerEnforced}]
```

Because the bucket ACL grants public read access, the request fails and returns the following error code:

An error occurred (`InvalidBucketAclWithObjectOwnership`) when calling the `PutBucketOwnershipControls` operation: Bucket cannot have ACLs set with ObjectOwnership's `BucketOwnerEnforced` setting

Logging and monitoring in Amazon S3

Monitoring is an important part of maintaining the reliability, availability, and performance of Amazon S3 and your AWS solutions. You should collect monitoring data from all of the parts of your AWS solution so that you can more easily debug a multi-point failure if one occurs. AWS provides several tools for monitoring your Amazon S3 resources and responding to potential incidents.

For more information, see [Monitoring Amazon S3 \(p. 959\)](#).

Amazon CloudWatch Alarms

Using Amazon CloudWatch alarms, you watch a single metric over a time period that you specify. If the metric exceeds a given threshold, a notification is sent to an Amazon SNS topic or AWS Auto Scaling policy. CloudWatch alarms do not invoke actions because they are in a particular state. Rather the state must have changed and been maintained for a specified number of periods. For more information, see [Monitoring metrics with Amazon CloudWatch \(p. 1002\)](#).

AWS CloudTrail Logs

CloudTrail provides a record of actions taken by a user, role, or an AWS service in Amazon S3. Using the information collected by CloudTrail, you can determine the request that was made to Amazon S3, the IP address from which the request was made, who made the request, when it was made, and additional details. For more information, see [Logging Amazon S3 API calls using AWS CloudTrail \(p. 962\)](#).

Amazon S3 Access Logs

Server access logs provide detailed records about requests that are made to a bucket. Server access logs are useful for many applications. For example, access log information can be useful in security and access audits. For more information, see [Logging requests using server access logging \(p. 978\)](#).

AWS Trusted Advisor

Trusted Advisor draws upon best practices learned from serving hundreds of thousands of AWS customers. Trusted Advisor inspects your AWS environment and then makes recommendations when opportunities exist to save money, improve system availability and performance, or help close security gaps. All AWS customers have access to five Trusted Advisor checks. Customers with a Business or Enterprise support plan can view all Trusted Advisor checks.

Trusted Advisor has the following Amazon S3-related checks:

- Logging configuration of Amazon S3 buckets.
- Security checks for Amazon S3 buckets that have open access permissions.
- Fault tolerance checks for Amazon S3 buckets that don't have versioning enabled, or have versioning suspended.

For more information, see [AWS Trusted Advisor in the AWS Support User Guide](#).

The following security best practices also address logging and monitoring:

- [Identify and audit all your Amazon S3 buckets](#)
- [Implement monitoring using Amazon Web Services monitoring tools](#)
- [Enable AWS Config](#)
- [Enable Amazon S3 server access logging](#)
- [Use CloudTrail](#)
- [Monitor Amazon Web Services security advisories](#)

Compliance Validation for Amazon S3

The security and compliance of Amazon S3 is assessed by third-party auditors as part of multiple AWS compliance programs, including the following:

- System and Organization Controls (SOC)
- Payment Card Industry Data Security Standard (PCI DSS)
- Federal Risk and Authorization Management Program (FedRAMP)
- Health Insurance Portability and Accountability Act (HIPAA)

AWS provides a frequently updated list of AWS services in scope of specific compliance programs at [AWS Services in Scope by Compliance Program](#).

Third-party audit reports are available for you to download using AWS Artifact. For more information, see [Downloading Reports in AWS Artifact](#).

For more information about AWS compliance programs, see [AWS Compliance Programs](#).

Your compliance responsibility when using Amazon S3 is determined by the sensitivity of your data, your organization's compliance objectives, and applicable laws and regulations. If your use of Amazon S3 is subject to compliance with standards like HIPAA, PCI, or FedRAMP, AWS provides resources to help:

- [Security and Compliance Quick Start Guides](#) that discuss architectural considerations and steps for deploying security- and compliance-focused baseline environments on AWS.
- [Architecting for HIPAA Security and Compliance](#) outlines how companies use AWS to help them meet HIPAA requirements.
- [AWS Compliance Resources](#) provide several different workbooks and guides that might apply to your industry and location.
- [AWS Config](#) can be used to assess how well your resource configurations comply with internal practices, industry guidelines, and regulations.
- [AWS Security Hub](#) provides you with a comprehensive view of your security state within AWS and helps you check your compliance with security industry standards and best practices.
- [Using S3 Object Lock \(p. 680\)](#) can help you meet technical requirements of financial services regulators (such as the SEC, FINRA, and CFTC) that require write once, read many (WORM) data storage for certain types of books and records information.
- [Amazon S3 Inventory \(p. 739\)](#) can help you audit and report on the replication and encryption status of your objects for business, compliance, and regulatory needs.

Resilience in Amazon S3

The AWS global infrastructure is built around Regions and Availability Zones. AWS Regions provide multiple, physically separated and isolated Availability Zones that are connected with low latency, high throughput, and highly redundant networking. These Availability Zones offer you an effective way to design and operate applications and databases. They are more highly available, fault tolerant, and scalable than traditional single data center infrastructures or multi-data center infrastructures. If you specifically need to replicate your data over greater geographic distances, you can use [Replicating objects \(p. 753\)](#), which enables automatic, asynchronous copying of objects across buckets in different AWS Regions.

Each AWS Region has multiple Availability Zones. You can deploy your applications across multiple Availability Zones in the same Region for fault tolerance and low latency. Availability Zones are connected to each other with fast, private fiber-optic networking, enabling you to easily architect applications that automatically fail over between Availability Zones without interruption.

For more information about AWS Regions and Availability Zones, see [AWS Global Infrastructure](#).

In addition to the AWS global infrastructure, Amazon S3 offers several features to help support your data resiliency and backup needs.

Lifecycle configuration

A lifecycle configuration is a set of rules that define actions that Amazon S3 applies to a group of objects. With lifecycle configuration rules, you can tell Amazon S3 to transition objects to less expensive storage classes, archive them, or delete them. For more information, see [Managing your storage lifecycle \(p. 701\)](#).

Versioning

Versioning is a means of keeping multiple variants of an object in the same bucket. You can use versioning to preserve, retrieve, and restore every version of every object stored in your Amazon S3 bucket. With versioning, you can easily recover from both unintended user actions and application failures. For more information, see [Using versioning in S3 buckets \(p. 638\)](#).

S3 Object Lock

You can use S3 Object Lock to store objects using a *write once, read many* (WORM) model. Using S3 Object Lock, you can prevent an object from being deleted or overwritten for a fixed amount of time or indefinitely. S3 Object Lock enables you to meet regulatory requirements that require WORM storage or simply to add an additional layer of protection against object changes and deletion. For more information, see [Using S3 Object Lock \(p. 680\)](#).

Storage classes

Amazon S3 offers a range of storage classes to choose from depending on the requirements of your workload. The S3 Standard-IA and S3 One Zone-IA storage classes are designed for data you access about once a month and need milliseconds access. The S3 Glacier Instant Retrieval storage class is designed for long-lived archive data accessed with milliseconds access that you access about once a quarter. For archive data that does not require immediate access, such as backups, you can use the S3 Glacier Flexible Retrieval or S3 Glacier Deep Archive storage classes. For more information, see [Using Amazon S3 storage classes \(p. 688\)](#).

The following security best practices also address resilience:

- [Enable versioning](#)
- [Consider Amazon S3 cross-region replication](#)
- [Identify and audit all your Amazon S3 buckets](#)

Encryption of Amazon S3 backups

If you are storing backups using Amazon S3, the encryption of your backups depends on the configuration of those buckets. Amazon S3 provides a way to set the default encryption behavior for an S3 bucket. You can set default encryption on a bucket so that all objects are encrypted when they are stored in the bucket. The default encryption supports keys stored in AWS KMS (SSE-KMS). For more information, see [Setting default server-side encryption behavior for Amazon S3 buckets \(p. 132\)](#).

For more information about Versioning and Object Lock, see the following topics: [Using versioning in S3 buckets \(p. 638\)](#) [Using S3 Object Lock \(p. 680\)](#)

Infrastructure security in Amazon S3

As a managed service, Amazon S3 is protected by the AWS global network security procedures that are described in the security pillar of the [AWS Well-Architected Framework](#).

Access to Amazon S3 via the network is through AWS published APIs. Clients must support Transport Layer Security (TLS) 1.0. We recommend TLS 1.2. Clients must also support cipher suites with Perfect Forward Secrecy (PFS) such as Ephemeral Diffie-Hellman (DHE) or Elliptic Curve Diffie-Hellman Ephemeral (ECDHE). Additionally, requests must be signed using AWS Signature V4 or AWS Signature V2, requiring valid credentials to be provided.

These APIs are callable from any network location. However, Amazon S3 does support resource-based access policies, which can include restrictions based on the source IP address. You can also use Amazon S3 bucket policies to control access to buckets from specific virtual private cloud (VPC) endpoints, or specific VPCs. Effectively, this isolates network access to a given Amazon S3 bucket from only the specific VPC within the AWS network. For more information, see [Controlling access from VPC endpoints with bucket policies \(p. 489\)](#).

The following security best practices also address infrastructure security in Amazon S3:

- Consider VPC endpoints for Amazon S3 access
- Identify and audit all your Amazon S3 buckets

Configuration and vulnerability analysis in Amazon S3

AWS handles basic security tasks like guest operating system (OS) and database patching, firewall configuration, and disaster recovery. These procedures have been reviewed and certified by the appropriate third parties. For more details, see the following resources:

- [Compliance Validation for Amazon S3 \(p. 627\)](#)
- [Shared Responsibility Model](#)
- [Amazon Web Services: Overview of Security Processes](#)

The following security best practices also address configuration and vulnerability analysis in Amazon S3:

- [Identify and audit all your Amazon S3 buckets](#)
- [Enable AWS Config](#)

Security Best Practices for Amazon S3

Amazon S3 provides a number of security features to consider as you develop and implement your own security policies. The following best practices are general guidelines and don't represent a complete security solution. Because these best practices might not be appropriate or sufficient for your environment, treat them as helpful considerations rather than prescriptions.

Topics

- [Amazon S3 Preventative Security Best Practices \(p. 633\)](#)
- [Amazon S3 Monitoring and Auditing Best Practices \(p. 635\)](#)

Amazon S3 Preventative Security Best Practices

The following best practices for Amazon S3 can help prevent security incidents.

Ensure that your Amazon S3 buckets use the correct policies and are not publicly accessible

Unless you explicitly require anyone on the internet to be able to read or write to your S3 bucket, you should ensure that your S3 bucket is not public. The following are some of the steps you can take:

- Use Amazon S3 block public access. With Amazon S3 block public access, account administrators and bucket owners can easily set up centralized controls to limit public access to their Amazon S3 resources that are enforced regardless of how the resources are created. For more information, see [Blocking public access to your Amazon S3 storage \(p. 584\)](#).
- Identify Amazon S3 bucket policies that allow a wildcard identity such as Principal "*" (which effectively means "anyone") or allows a wildcard action "*" (which effectively allows the user to perform any action in the Amazon S3 bucket).
- Similarly, note Amazon S3 bucket access control lists (ACLs) that provide read, write, or full-access to "Everyone" or "Any authenticated AWS user."
- Use the `ListBuckets` API to scan all of your Amazon S3 buckets. Then use `GetBucketAcl`, `GetBucketWebsite`, and `GetBucketPolicy` to determine whether the bucket has compliant access controls and configuration.
- Use [AWS Trusted Advisor](#) to inspect your Amazon S3 implementation.
- Consider implementing on-going detective controls using the `s3-bucket-public-read-prohibited` and `s3-bucket-public-write-prohibited` managed AWS Config Rules.

For more information, see [Identity and access management in Amazon S3 \(p. 394\)](#).

Implement least privilege access

When granting permissions, you decide who is getting what permissions to which Amazon S3 resources. You enable specific actions that you want to allow on those resources. Therefore you should grant only the permissions that are required to perform a task. Implementing least privilege access is fundamental in reducing security risk and the impact that could result from errors or malicious intent.

The following tools are available to implement least privilege access:

- [Amazon S3 actions \(p. 415\)](#) and [Permissions Boundaries for IAM Entities](#)
- [Bucket policies and user policies \(p. 411\)](#)
- [Access control list \(ACL\) overview \(p. 554\)](#)
- [Service Control Policies](#)

For guidance on what to consider when choosing one or more of the preceding mechanisms, see [Access policy guidelines \(p. 400\)](#).

Use IAM roles for applications and AWS services that require Amazon S3 access

For applications on Amazon EC2 or other AWS services to access Amazon S3 resources, they must include valid AWS credentials in their AWS API requests. You should not store AWS credentials directly in the application or Amazon EC2 instance. These are long-term credentials that are not automatically rotated and could have a significant business impact if they are compromised.

Instead, you should use an IAM role to manage temporary credentials for applications or services that need to access Amazon S3. When you use a role, you don't have to distribute long-term credentials (such as a user name and password or access keys) to an Amazon EC2 instance or AWS service such as AWS Lambda. The role supplies temporary permissions that applications can use when they make calls to other AWS resources.

For more information, see the following topics in the *IAM User Guide*:

- [IAM Roles](#)
- [Common Scenarios for Roles: Users, Applications, and Services](#)

Consider encryption of data at rest

You have the following options for protecting data at rest in Amazon S3:

- **Server-Side Encryption** – Request Amazon S3 to encrypt your object before saving it on disks in its data centers and then decrypt it when you download the objects. Server-side encryption can help reduce risk to your data by encrypting the data with a key that is stored in a different mechanism than the mechanism that stores the data itself.

Amazon S3 provides these server-side encryption options:

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

For more information, see [Protecting data using server-side encryption \(p. 338\)](#).

- **Client-Side Encryption** – Encrypt data client-side and upload the encrypted data to Amazon S3. In this case, you manage the encryption process, the encryption keys, and related tools. As with server-side encryption, client-side encryption can help reduce risk by encrypting the data with a key that is stored in a different mechanism than the mechanism that stores the data itself.

Amazon S3 provides multiple client-side encryption options. For more information, see [Protecting data using client-side encryption \(p. 381\)](#).

Enforce encryption of data in transit

You can use HTTPS (TLS) to help prevent potential attackers from eavesdropping on or manipulating network traffic using person-in-the-middle or similar attacks. You should allow only encrypted connections over HTTPS (TLS) using the `aws:SecureTransport` condition on Amazon S3 bucket policies.

Also consider implementing on-going detective controls using the `s3-bucket-ssl-requests-only` managed AWS Config rule.

Consider S3 Object Lock

[Using S3 Object Lock \(p. 680\)](#) enables you to store objects using a "Write Once Read Many" (WORM) model. S3 Object Lock can help prevent accidental or inappropriate deletion of data. For example, you could use S3 Object Lock to help protect your AWS CloudTrail logs.

Enable versioning

Versioning is a means of keeping multiple variants of an object in the same bucket. You can use versioning to preserve, retrieve, and restore every version of every object stored in your Amazon S3

bucket. With versioning, you can easily recover from both unintended user actions and application failures.

Also consider implementing on-going detective controls using the [s3-bucket-versioning-enabled](#) managed AWS Config rule.

For more information, see [Using versioning in S3 buckets \(p. 638\)](#).

Consider Amazon S3 cross-region replication

Although Amazon S3 stores your data across multiple geographically diverse Availability Zones by default, compliance requirements might dictate that you store data at even greater distances. Cross-region replication (CRR) allows you to replicate data between distant AWS Regions to help satisfy these requirements. CRR enables automatic, asynchronous copying of objects across buckets in different AWS Regions. For more information, see [Replicating objects \(p. 753\)](#).

Note

CRR requires that both source and destination S3 buckets have versioning enabled.

Also consider implementing on-going detective controls using the [s3-bucket-replication-enabled](#) managed AWS Config rule.

Consider VPC endpoints for Amazon S3 access

A VPC endpoint for Amazon S3 is a logical entity within a virtual private cloud (VPC) that allows connectivity only to Amazon S3. You can use Amazon S3 bucket policies to control access to buckets from specific VPC endpoints, or specific VPCs. A VPC endpoint can help prevent traffic from potentially traversing the open internet and being subject to open internet environment.

VPC endpoints for Amazon S3 provide multiple ways to control access to your Amazon S3 data:

- You can control the requests, users, or groups that are allowed through a specific VPC endpoint.
- You can control which VPCs or VPC endpoints have access to your S3 buckets by using S3 bucket policies.
- You can help prevent data exfiltration by using a VPC that does not have an internet gateway.

For more information, see [Controlling access from VPC endpoints with bucket policies \(p. 489\)](#).

Amazon S3 Monitoring and Auditing Best Practices

The following best practices for Amazon S3 can help detect potential security weaknesses and incidents.

Identify and audit all your Amazon S3 buckets

Identification of your IT assets is a crucial aspect of governance and security. You need to have visibility of all your Amazon S3 resources to assess their security posture and take action on potential areas of weakness.

Use Tag Editor to identify security-sensitive or audit-sensitive resources, then use those tags when you need to search for these resources. For more information, see [Searching for Resources to Tag](#).

Use Amazon S3 Inventory to audit and report on the replication and encryption status of your objects for business, compliance, and regulatory needs. For more information, see [Amazon S3 Inventory \(p. 739\)](#).

Create resource groups for your Amazon S3 resources. For more information, see [What Is AWS Resource Groups?](#)

Implement monitoring using AWS monitoring tools

Monitoring is an important part of maintaining the reliability, security, availability, and performance of Amazon S3 and your AWS solutions. AWS provides several tools and services to help you monitor Amazon S3 and your other AWS services. For example, you can monitor CloudWatch metrics for

Amazon S3, particularly `PutRequests`, `GetRequests`, `4xxErrors`, and `DeleteRequests`. For more information, see [Monitoring metrics with Amazon CloudWatch \(p. 1002\)](#) and, [Monitoring Amazon S3 \(p. 959\)](#).

For a second example, see [Example: Amazon S3 Bucket Activity](#). This example describes how to create an Amazon CloudWatch alarm that is triggered when an Amazon S3 API call is made to PUT or DELETE bucket policy, bucket lifecycle, or bucket replication, or to PUT a bucket ACL.

Enable Amazon S3 server access logging

Server access logging provides detailed records of the requests that are made to a bucket. Server access logs can assist you in security and access audits, help you learn about your customer base, and understand your Amazon S3 bill. For instructions on enabling server access logging, see [Logging requests using server access logging \(p. 978\)](#).

Also consider implementing on-going detective controls using the `s3-bucket-logging-enabled` AWS Config managed rule.

Use AWS CloudTrail

AWS CloudTrail provides a record of actions taken by a user, a role, or an AWS service in Amazon S3. You can use information collected by CloudTrail to determine the request that was made to Amazon S3, the IP address from which the request was made, who made the request, when it was made, and additional details. For example, you can identify CloudTrail entries for Put actions that impact data access, in particular `PutBucketAcl`, `PutObjectAcl`, `PutBucketPolicy`, and `PutBucketWebsite`. When you set up your AWS account, CloudTrail is enabled by default. You can view recent events in the CloudTrail console. To create an ongoing record of activity and events for your Amazon S3 buckets, you can create a trail in the CloudTrail console. For more information, see [Logging Data Events for Trails](#) in the *AWS CloudTrail User Guide*.

When you create a trail, you can configure CloudTrail to log data events. Data events are records of resource operations performed on or within a resource. In Amazon S3, data events record object-level API activity for individual buckets. CloudTrail supports a subset of Amazon S3 object-level API operations such as `GetObject`, `DeleteObject`, and `PutObject`. For more information about how CloudTrail works with Amazon S3, see [Logging Amazon S3 API calls using AWS CloudTrail \(p. 962\)](#). In the Amazon S3 console, you can also configure your S3 buckets to [Enabling CloudTrail event logging for S3 buckets and objects \(p. 971\)](#).

AWS Config provides a managed rule (`cloudtrail-s3-dataevents-enabled`) that you can use to confirm that at least one CloudTrail trail is logging data events for your S3 buckets. For more information, see `cloudtrail-s3-dataevents-enabled` in the *AWS Config Developer Guide*.

Enable AWS Config

Several of the best practices listed in this topic suggest creating AWS Config rules. AWS Config enables you to assess, audit, and evaluate the configurations of your AWS resources. AWS Config monitors resource configurations, allowing you to evaluate the recorded configurations against the desired secure configurations. Using AWS Config, you can review changes in configurations and relationships between AWS resources, investigate detailed resource configuration histories, and determine your overall compliance against the configurations specified in your internal guidelines. This can help you simplify compliance auditing, security analysis, change management, and operational troubleshooting. For more information, see [Setting Up AWS Config with the Console](#) in the *AWS Config Developer Guide*. When specifying the resource types to record, ensure that you include Amazon S3 resources.

For an example of how to use AWS Config to monitor for and respond to Amazon S3 buckets that allow public access, see [How to Use AWS Config to Monitor for and Respond to Amazon S3 Buckets Allowing Public Access](#) on the *AWS Security Blog*.

Consider using Amazon Macie with Amazon S3

Amazon Macie is a data security and data privacy service that uses machine learning and pattern matching to help you discover, monitor, and protect sensitive data in your AWS environment. Macie

automates the discovery of sensitive data, such as personally identifiable information (PII) and financial data, to provide you with a better understanding of the data that your organization stores in Amazon S3.

Macie also provides you with an inventory of your S3 buckets, and it automatically evaluates and monitors those buckets for security and access control. If Macie detects sensitive data or potential issues with the security or privacy of your data, it creates detailed findings for you to review and remediate as necessary. For more information, see [What is Amazon Macie?](#)

Monitor AWS security advisories

You should regularly check security advisories posted in Trusted Advisor for your AWS account. In particular, note warnings about Amazon S3 buckets with “open access permissions.” You can do this programmatically using [describe-trusted-advisor-checks](#).

Further, actively monitor the primary email address registered to each of your AWS accounts. AWS will contact you, using this email address, about emerging security issues that might affect you.

AWS operational issues with broad impact are posted on the [AWS Service Health Dashboard](#). Operational issues are also posted to individual accounts via the Personal Health Dashboard. For more information, see the [AWS Health Documentation](#).

Managing your Amazon S3 storage

After you create buckets and upload objects in Amazon S3, you can manage your object storage using features such as versioning, storage classes, object locking, batch operations, replication, tags, and more. The following sections provide detailed information about the storage management capabilities and features that are available in Amazon S3.

Topics

- [Using versioning in S3 buckets \(p. 638\)](#)
- [Using AWS Backup for Amazon S3 \(p. 669\)](#)
- [Working with archived objects \(p. 670\)](#)
- [Using S3 Object Lock \(p. 680\)](#)
- [Using Amazon S3 storage classes \(p. 688\)](#)
- [Amazon S3 Intelligent-Tiering \(p. 693\)](#)
- [Managing your storage lifecycle \(p. 701\)](#)
- [Amazon S3 Inventory \(p. 739\)](#)
- [Replicating objects \(p. 753\)](#)
- [Categorizing your storage using tags \(p. 825\)](#)
- [Using cost allocation S3 bucket tags \(p. 834\)](#)
- [Filtering and retrieving data using Amazon S3 Select \(p. 852\)](#)
- [Performing large-scale batch operations on Amazon S3 objects \(p. 881\)](#)

Using versioning in S3 buckets

Versioning in Amazon S3 is a means of keeping multiple variants of an object in the same bucket. You can use the S3 Versioning feature to preserve, retrieve, and restore every version of every object stored in your buckets. With versioning you can recover more easily from both unintended user actions and application failures. After versioning is enabled for a bucket, if Amazon S3 receives multiple write requests for the same object simultaneously, it stores all of those objects.

Versioning-enabled buckets can help you recover objects from accidental deletion or overwrite. For example, if you delete an object, Amazon S3 inserts a delete marker instead of removing the object permanently. The delete marker becomes the current object version. If you overwrite an object, it results in a new object version in the bucket. You can always restore the previous version. For more information, see [Deleting object versions from a versioning-enabled bucket \(p. 659\)](#).

By default, S3 Versioning is disabled on buckets, and you must explicitly enable it. For more information, see [Enabling versioning on buckets \(p. 643\)](#).

Note

- The SOAP API does not support S3 Versioning. SOAP support over HTTP is deprecated, but it is still available over HTTPS. New Amazon S3 features are not supported for SOAP.
- Normal Amazon S3 rates apply for every version of an object stored and transferred. Each version of an object is the entire object; it is not just a diff from the previous version. Thus, if you have three versions of an object stored, you are charged for three objects.

Unversioned, versioning-enabled, and versioning-suspended buckets

Buckets can be in one of three states:

- Unversioned (the default)
- Versioning-enabled
- Versioning-suspended

You enable and suspend versioning at the bucket level. After you version-enable a bucket, it can never return to an unversioned state. But you can *suspend* versioning on that bucket.

The versioning state applies to all (never some) of the objects in that bucket. When you enable versioning in a bucket, all new objects are versioned and given a unique version ID. Objects that already existed in the bucket at the time versioning was enabled will thereafter *always* be versioned and given a unique version ID when they are modified by future requests. Note the following:

- Objects that are stored in your bucket before you set the versioning state have a version ID of `null`. When you enable versioning, existing objects in your bucket do not change. What changes is how Amazon S3 handles the objects in future requests. For more information, see [Working with objects in a versioning-enabled bucket \(p. 649\)](#).
- The bucket owner (or any user with appropriate permissions) can suspend versioning to stop accruing object versions. When you suspend versioning, existing objects in your bucket do not change. What changes is how Amazon S3 handles objects in future requests. For more information, see [Working with objects in a versioning-suspended bucket \(p. 667\)](#).

Using S3 Versioning with S3 Lifecycle

To customize your data retention approach and control storage costs, use object versioning with S3 Lifecycle. For more information, see [Managing your storage lifecycle \(p. 701\)](#). For information about creating S3 Lifecycle policies using the AWS Management Console, AWS CLI, AWS SDKs, or the REST API, see [Setting lifecycle configuration on a bucket \(p. 708\)](#).

Important

If you have an object expiration lifecycle policy in your unversioned bucket and you want to maintain the same permanent delete behavior when you enable versioning, you must add a noncurrent expiration policy. The noncurrent expiration lifecycle policy manages the deletes of the noncurrent object versions in the version-enabled bucket. (A version-enabled bucket maintains one current, and zero or more noncurrent, object versions.) For more information, see [Setting lifecycle configuration on a bucket \(p. 708\)](#).

For information about working with S3 Versioning, see the following topics.

Topics

- [How S3 Versioning works \(p. 640\)](#)
- [Enabling versioning on buckets \(p. 643\)](#)
- [Configuring MFA delete \(p. 648\)](#)
- [Working with objects in a versioning-enabled bucket \(p. 649\)](#)
- [Working with objects in a versioning-suspended bucket \(p. 667\)](#)

How S3 Versioning works

You can use S3 Versioning to keep multiple versions of an object in one bucket and enable you to restore objects that are accidentally deleted or overwritten. For example, if you delete an object, instead of removing it permanently, Amazon S3 inserts a delete marker, which becomes the current object version. You can then restore the previous version. For more information, see [Deleting object versions from a versioning-enabled bucket \(p. 659\)](#). If you overwrite an object, it results in a new object version in the bucket. You can always restore the previous version.

Note

Normal Amazon S3 rates apply for every version of an object stored and transferred. Each version of an object is the entire object; it is not just a diff from the previous version. Thus, if you have three versions of an object stored, you are charged for three objects.

Each S3 bucket that you create has a *versioning* subresource associated with it. (For more information, see [Bucket configuration options \(p. 116\)](#).) By default, your bucket is *unversioned*, and the versioning subresource stores the empty versioning configuration, as follows.

```
<VersioningConfiguration xmlns="http://s3.amazonaws.com/doc/2006-03-01/">
</VersioningConfiguration>
```

To enable versioning, you can send a request to Amazon S3 with a versioning configuration that includes a status.

```
<VersioningConfiguration xmlns="http://s3.amazonaws.com/doc/2006-03-01/">
<Status>Enabled</Status>
</VersioningConfiguration>
```

To suspend versioning, you set the status value to `Suspended`.

Note

If you enable versioning on a bucket for the first time, it might take a short amount of time for the change to be fully propagated. We recommend that you wait for 15 minutes after enabling versioning before issuing write operations (PUT or DELETE) on objects in the bucket.

The bucket owner and all authorized IAM users can enable versioning. The bucket owner is the AWS account that created the bucket (the root account). For more information about permissions, see [Identity and access management in Amazon S3 \(p. 394\)](#).

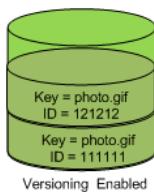
For more information about enabling and disabling S3 Versioning using the AWS Management Console, AWS Command Line Interface (AWS CLI), or REST API, see [the section called “Enabling versioning on buckets” \(p. 643\)](#).

Topics

- [Version IDs \(p. 640\)](#)
- [Versioning workflows \(p. 641\)](#)

Version IDs

If you enable versioning for a bucket, Amazon S3 automatically generates a unique version ID for the object that is being stored. For example, in one bucket you can have two objects with the same key but different version IDs, such as `photo.gif` (version 111111) and `photo.gif` (version 121212).



Each object has a version ID, whether or not S3 Versioning is enabled. If S3 Versioning is not enabled, Amazon S3 sets the value of version ID to null. If you enable S3 Versioning, Amazon S3 assigns a version ID value for the object. This value distinguishes it from other versions of the same key.

When you enable S3 Versioning on an existing bucket, objects that are already stored in the bucket are unchanged. The version IDs (null), contents, and permissions remain the same. After you enable S3 Versioning for a bucket, each object that is added to the bucket gets a version ID, which distinguishes it from other versions of the same key.

Only Amazon S3 generates version IDs, and they cannot be edited. Version IDs are Unicode, UTF-8 encoded, URL-ready, opaque strings that are no more than 1,024 bytes long. The following is an example:

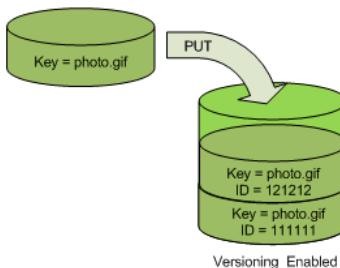
3sL4kqtJlcpXroDTDmJ+rmSpXd3dIbrHY+MTRCxzf3vjVBH40Nr8X8gdRQBpUMLUo

Note

For simplicity, the other examples in this topic use much shorter IDs.

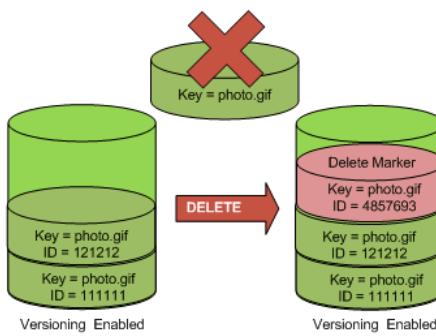
Versioning workflows

When you **PUT** an object in a versioning-enabled bucket, the noncurrent version is not overwritten. The following figure shows that when a new version of `photo.gif` is **PUT** into a bucket that already contains an object with the same name, the original object (ID = 111111) remains in the bucket, Amazon S3 generates a new version ID (121212), and adds the newer version to the bucket.

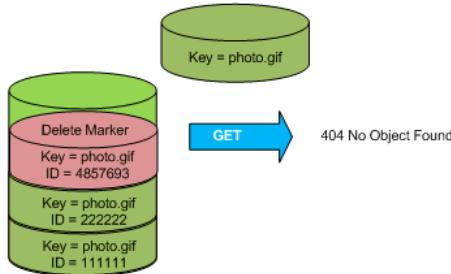


This functionality prevents you from accidentally overwriting or deleting objects and gives you the opportunity to retrieve a previous version of an object.

When you **DELETE** an object, all versions remain in the bucket and Amazon S3 inserts a delete marker, as shown in the following figure.

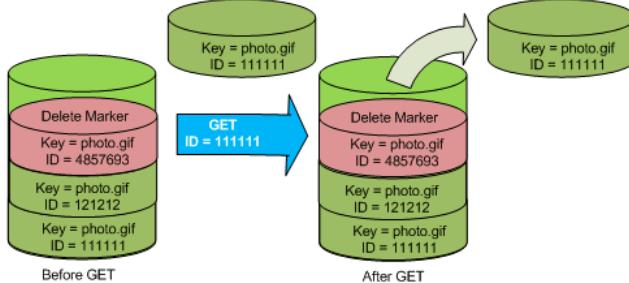


The delete marker becomes the current version of the object. By default, `GET` requests retrieve the most recently stored version. Performing a simple `GET` Object request when the current version is a delete marker returns a `404 Not Found` error, as shown in the following figure.

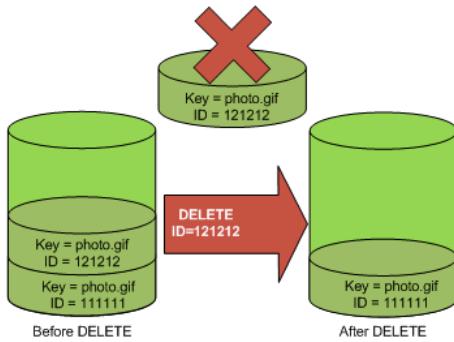


However, you can `GET` a noncurrent version of an object by specifying its version ID. In the following figure, you `GET` a specific object version, `111111`. Amazon S3 returns that object version even though it's not the current version.

For more information, see [Retrieving object versions from a versioning-enabled bucket \(p. 655\)](#).



You can permanently delete an object by specifying the version you want to delete. Only the owner of an Amazon S3 bucket can permanently delete a version. The following figure shows how `DELETE versionId` permanently deletes an object from a bucket and that Amazon S3 doesn't insert a delete marker.



You can add more security by configuring a bucket to enable MFA (multi-factor authentication) delete. When you do, the bucket owner must include two forms of authentication in any request to delete a version or change the versioning state of the bucket. For more information, see [Configuring MFA delete \(p. 648\)](#).

New versions are created only when you `PUT` a new object. Be aware that certain actions like `COPY` work by implementing `PUT`. Taking actions that modify the current object will not create a new version because they do not `PUT` a new object. This includes actions such as changing the tags on an object.

Important

If you notice a significant increase in the number of HTTP 503-slow down responses received for Amazon S3 `PUT` or `DELETE` object requests to a bucket that has S3 Versioning enabled, you might have one or more objects in the bucket for which there are millions of versions. For more information, see [Significant Increases in HTTP 503 Responses to Amazon S3 Requests to Buckets with Versioning Enabled \(p. 1521\)](#) in the Troubleshooting section.

Enabling versioning on buckets

You can use S3 Versioning to keep multiple versions of an object in one bucket. This section provides examples of how to enable versioning on a bucket using the console, REST API, AWS SDKs, and AWS Command Line Interface (AWS CLI).

Note

If you enable versioning on a bucket for the first time, it might take a short amount of time for the change to be fully propagated. We recommend that you wait for 15 minutes after enabling versioning before issuing write operations (`PUT` or `DELETE`) on objects in the bucket.

For more information about S3 Versioning, see [Using versioning in S3 buckets \(p. 638\)](#). For information about working with objects that are in versioning-enabled buckets, see [Working with objects in a versioning-enabled bucket \(p. 649\)](#).

Each S3 bucket that you create has a *versioning* subresource associated with it. (For more information, see [Bucket configuration options \(p. 116\)](#).) By default, your bucket is *unversioned*, and the versioning subresource stores the empty versioning configuration, as follows.

```
<VersioningConfiguration xmlns="http://s3.amazonaws.com/doc/2006-03-01/">
</VersioningConfiguration>
```

To enable versioning, you can send a request to Amazon S3 with a versioning configuration that includes a status.

```
<VersioningConfiguration xmlns="http://s3.amazonaws.com/doc/2006-03-01/">
<Status>Enabled</Status>
```

```
</VersioningConfiguration>
```

To suspend versioning, you set the status value to **Suspended**.

The bucket owner and all authorized IAM users can enable versioning. The bucket owner is the AWS account that created the bucket (the root account). For more information about permissions, see [Identity and access management in Amazon S3 \(p. 394\)](#).

The following sections provide more detail about enabling S3 Versioning using the console, AWS CLI, and the AWS SDKs.

Using the S3 console

Follow these steps to use the AWS Management Console to enable versioning on an S3 bucket.

To enable or disable versioning on an S3 bucket

1. Sign in to the AWS Management Console and open the Amazon S3 console at <https://console.aws.amazon.com/s3/>.
2. In the **Buckets** list, choose the name of the bucket that you want to enable versioning for.
3. Choose **Properties**.
4. Under **Bucket Versioning**, choose **Edit**.
5. Choose **Suspend** or **Enable**, and then choose **Save changes**.

Note

You can use AWS multi-factor authentication (MFA) with versioning. When you use MFA with versioning, you must provide your AWS account's access keys and a valid code from the account's MFA device to permanently delete an object version or suspend or reactivate versioning.

To use MFA with versioning, you enable **MFA Delete**. However, you can't enable **MFA Delete** using the AWS Management Console. You must use the AWS Command Line Interface (AWS CLI) or the API. For more information, see [Configuring MFA delete \(p. 648\)](#).

Using the AWS CLI

The following example enables versioning on an S3 bucket.

```
aws s3api put-bucket-versioning --bucket DOC-EXAMPLE-BUCKET1 --versioning-configuration  
Status=Enabled
```

The following example enables S3 Versioning and multi-factor authentication (MFA) delete on a bucket.

```
aws s3api put-bucket-versioning --bucket DOC-EXAMPLE-BUCKET1 --versioning-configuration  
Status=Enabled,MFADelete=Enabled --mfa "SERIAL 123456"
```

Note

Using MFA delete requires an approved physical or virtual authentication device. For more information about using MFA delete in Amazon S3, see [Configuring MFA delete \(p. 648\)](#).

For more information about enabling versioning using the AWS CLI, see [put-bucket-versioning](#) in the [AWS CLI Command Reference](#).

Using the AWS SDKs

The following examples enable versioning on a bucket and then retrieve versioning status using the AWS SDK for Java and the AWS SDK for .NET. For information about using other AWS SDKs, see the [AWS Developer Center](#).

.NET

For information about how to create and test a working sample, see [Running the Amazon S3 .NET Code Examples \(p. 1193\)](#).

```
using System;
using Amazon.S3;
using Amazon.S3.Model;

namespace s3.amazonaws.com.docsamples
{
    class BucketVersioningConfiguration
    {
        static string bucketName = "**** bucket name ****";

        public static void Main(string[] args)
        {
            using (var client = new AmazonS3Client(Amazon.RegionEndpoint.USEast1))
            {
                try
                {
                    EnableVersioningOnBucket(client);
                    string bucketVersioningStatus =
                    RetrieveBucketVersioningConfiguration(client);
                }
                catch (AmazonS3Exception amazonS3Exception)
                {
                    if (amazonS3Exception.ErrorCode != null &&
                        (amazonS3Exception.ErrorCode.Equals("InvalidAccessKeyId") ||
                        amazonS3Exception.ErrorCode.Equals("InvalidSecurity")))
                    {
                        Console.WriteLine("Check the provided AWS Credentials.");
                        Console.WriteLine(
                            "To sign up for service, go to http://aws.amazon.com/s3");
                    }
                    else
                    {
                        Console.WriteLine(
                            "Error occurred. Message:{0}' when listing objects",
                            amazonS3Exception.Message);
                    }
                }
            }

            Console.WriteLine("Press any key to continue...");
            Console.ReadKey();
        }

        static void EnableVersioningOnBucket(IAmazonS3 client)
        {

            PutBucketVersioningRequest request = new PutBucketVersioningRequest
            {
                BucketName = bucketName,
                VersioningConfig = new S3BucketVersioningConfig
                {
                    Status = VersionStatus.Enabled
                }
            };
        }
    }
}
```

```
        }

    };

    PutBucketVersioningResponse response =
client.PutBucketVersioning(request);
}

static string RetrieveBucketVersioningConfiguration(IAmazonS3 client)
{
    GetBucketVersioningRequest request = new GetBucketVersioningRequest
{
    BucketName = bucketName
};

GetBucketVersioningResponse response =
client.GetBucketVersioning(request);
return response.VersioningConfig.Status;
}
}
```

Java

For instructions on how to create and test a working sample, see [Testing the Amazon S3 Java Code Examples \(p. 1192\)](#).

```
import java.io.IOException;

import com.amazonaws.auth.profile.ProfileCredentialsProvider;
import com.amazonaws.regions.Region;
import com.amazonaws.regions.Regions;
import com.amazonaws.services.s3.AmazonS3Client;
import com.amazonaws.services.s3.model.AmazonS3Exception;
import com.amazonaws.services.s3.model.BucketVersioningConfiguration;
import com.amazonaws.services.s3.model.SetBucketVersioningConfigurationRequest;

public class BucketVersioningConfigurationExample {
    public static String bucketName = "*** bucket name ***";
    public static AmazonS3Client s3Client;

    public static void main(String[] args) throws IOException {
        s3Client = new AmazonS3Client(new ProfileCredentialsProvider());
        s3Client.setRegion(Region.getRegion(Regions.US_EAST_1));
        try {

            // 1. Enable versioning on the bucket.
            BucketVersioningConfiguration configuration =
                new BucketVersioningConfiguration().withStatus("Enabled");

            SetBucketVersioningConfigurationRequest setBucketVersioningConfigurationRequest =
                new SetBucketVersioningConfigurationRequest(bucketName, configuration);

            s3Client.setBucketVersioningConfiguration(setBucketVersioningConfigurationRequest);

            // 2. Get bucket versioning configuration information.
            BucketVersioningConfiguration conf =
                s3Client.getBucketVersioningConfiguration(bucketName);
            System.out.println("bucket versioning configuration status: " +
                conf.getStatus());

        } catch (AmazonS3Exception amazonS3Exception) {
            System.out.format("An Amazon S3 error occurred. Exception: %s",
                amazonS3Exception.toString());
        } catch (Exception ex) {
```

```
        System.out.format("Exception: %s", ex.toString());
    }
}
```

Python

For instructions on how to create and test a working sample, see [Using the AWS SDK for Python \(Boto\) \(p. 1196\)](#).

The following Python code example creates an Amazon S3 bucket, enables it for versioning, and configures a lifecycle that expires noncurrent object versions after 7 days.

```
def create_versioned_bucket(bucket_name, prefix):
    """
    Creates an Amazon S3 bucket, enables it for versioning, and configures a lifecycle
    that expires noncurrent object versions after 7 days.

    Adding a lifecycle configuration to a versioned bucket is a best practice.
    It helps prevent objects in the bucket from accumulating a large number of
    noncurrent versions, which can slow down request performance.

    Usage is shown in the usage_demo_single_object function at the end of this module.

    :param bucket_name: The name of the bucket to create.
    :param prefix: Identifies which objects are automatically expired under the
                   configured lifecycle rules.
    :return: The newly created bucket.
    """
    try:
        bucket = s3.create_bucket(
            Bucket=bucket_name,
            CreateBucketConfiguration={
                'LocationConstraint': s3.meta.client.meta.region_name
            }
        )
        logger.info("Created bucket %s.", bucket.name)
    except ClientError as error:
        if error.response['Error']['Code'] == 'BucketAlreadyOwnedByYou':
            logger.warning("Bucket %s already exists! Using it.", bucket_name)
            bucket = s3.Bucket(bucket_name)
        else:
            logger.exception("Couldn't create bucket %s.", bucket_name)
            raise

    try:
        bucket.Versioning().enable()
        logger.info("Enabled versioning on bucket %s.", bucket.name)
    except ClientError:
        logger.exception("Couldn't enable versioning on bucket %s.", bucket.name)
        raise

    try:
        expiration = 7
        bucket.LifecycleConfiguration().put(
            LifecycleConfiguration={
                'Rules': [
                    {
                        'Status': 'Enabled',
                        'Prefix': prefix,
                        'NoncurrentVersionExpiration': {'NoncurrentDays': expiration}
                    }
                ]
            }
        )
        logger.info("Configured lifecycle to expire noncurrent versions after %s days"

```

```
    "on bucket %s.", expiration, bucket.name)
except ClientError as error:
    logger.warning("Couldn't configure lifecycle on bucket %s because %s. "
                   "Continuing anyway.", bucket.name, error)

return bucket
```

Configuring MFA delete

When working with S3 Versioning in Amazon S3 buckets, you can optionally add another layer of security by configuring a bucket to enable *MFA (multi-factor authentication) delete*. When you do this, the bucket owner must include two forms of authentication in any request to delete a version or change the versioning state of the bucket.

MFA delete requires additional authentication for either of the following operations:

- Changing the versioning state of your bucket
- Permanently deleting an object version

MFA delete requires two forms of authentication together:

- Your security credentials
- The concatenation of a valid serial number, a space, and the six-digit code displayed on an approved authentication device

MFA delete thus provides added security if, for example, your security credentials are compromised. MFA delete can help prevent accidental bucket deletions by requiring the user who initiates the delete action to prove physical possession of an MFA device with an MFA code and adding an extra layer of friction and security to the delete action.

The bucket owner, the AWS account that created the bucket (root account), and all authorized IAM users can enable versioning. However, only the bucket owner (root account) can enable MFA delete. For more information, see [Securing Access to AWS Using MFA](#) on the AWS Security Blog.

Note

To use MFA delete with versioning, you enable `MFA Delete`. However, you cannot enable `MFA Delete` using the AWS Management Console. You must use the AWS Command Line Interface (AWS CLI) or the API.

For examples of using MFA delete with versioning, see the examples section in the topic [Enabling versioning on buckets \(p. 643\)](#).

You cannot use MFA delete with lifecycle configurations. For more information about lifecycle configurations and how they interact with other configurations, see [Lifecycle and other bucket configurations \(p. 719\)](#).

To enable or disable MFA delete, you use the same API that you use to configure versioning on a bucket. Amazon S3 stores the MFA delete configuration in the same *versioning* subresource that stores the bucket's versioning status.

```
<VersioningConfiguration xmlns="http://s3.amazonaws.com/doc/2006-03-01/">
  <Status>VersioningState</Status>
  <MfaDelete>MfaDeleteState</MfaDelete>
</VersioningConfiguration>
```

To use MFA delete, you can use either a hardware or virtual MFA device to generate an authentication code. The following example shows a generated authentication code displayed on a hardware device.



MFA delete and MFA-protected API access are features intended to provide protection for different scenarios. You configure MFA delete on a bucket to help ensure that the data in your bucket cannot be accidentally deleted. MFA-protected API access is used to enforce another authentication factor (MFA code) when accessing sensitive Amazon S3 resources. You can require any operations against these Amazon S3 resources to be done with temporary credentials created using MFA. For an example, see [Adding a bucket policy to require MFA \(p. 495\)](#).

For more information about how to purchase and activate an authentication device, see [Multi-factor authentication](#).

To enable S3 Versioning and configure MFA delete

Using the AWS CLI

The following example enables S3 Versioning and multi-factor authentication (MFA) delete on a bucket.

```
aws s3api put-bucket-versioning --bucket DOC-EXAMPLE-BUCKET1 --versioning-configuration  
Status=Enabled,MFADelete=Enabled --mfa "SERIAL 123456"
```

Using the REST API

For more information about specifying MFA delete using the Amazon S3 REST API, see [PutBucketVersioning](#) *Amazon Simple Storage Service API Reference*.

Working with objects in a versioning-enabled bucket

Objects that are stored in an Amazon S3 bucket before you set the versioning state have a version ID of null. When you enable versioning, existing objects in your bucket do not change. What changes is how Amazon S3 handles the objects in future requests.

Transitioning object versions

You can define lifecycle configuration rules for objects that have a well-defined lifecycle to transition object versions to the S3 Glacier Flexible Retrieval storage class at a specific time in the object's lifetime. For more information, see [Managing your storage lifecycle \(p. 701\)](#).

The topics in this section explain various object operations in a versioning-enabled bucket. For more information about versioning, see [Using versioning in S3 buckets \(p. 638\)](#).

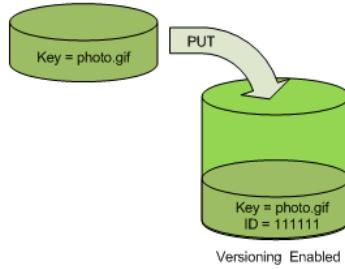
Topics

- [Adding objects to versioning-enabled buckets \(p. 649\)](#)
- [Listing objects in a versioning-enabled bucket \(p. 650\)](#)
- [Retrieving object versions from a versioning-enabled bucket \(p. 655\)](#)
- [Deleting object versions from a versioning-enabled bucket \(p. 659\)](#)
- [Configuring versioned object permissions \(p. 666\)](#)

Adding objects to versioning-enabled buckets

After you enable versioning on a bucket, Amazon S3 automatically adds a unique version ID to every object stored (using `PUT`, `POST`, or `COPY`) in the bucket.

The following figure shows that Amazon S3 adds a unique version ID to an object when it is added to a versioning-enabled bucket.



Note

The version ID values that Amazon S3 assigns are URL safe (can be included as part of a URI).

For more information about versioning, see [Using versioning in S3 buckets \(p. 638\)](#). You can add object versions to a versioning-enabled bucket using the console, AWS SDKs, and REST API.

Using the console

For instructions, see [Uploading objects \(p. 158\)](#).

Using the AWS SDKs

For examples of uploading objects using the AWS SDKs for Java, .NET, and PHP, see [Uploading objects \(p. 158\)](#). The examples for uploading objects in nonversioned and versioning-enabled buckets are the same, although in the case of versioning-enabled buckets, Amazon S3 assigns a version number. Otherwise, the version number is null.

For information about using other AWS SDKs, see the [AWS Developer Center](#).

Using the REST API

To add objects to versioning-enabled buckets

1. Enable versioning on a bucket using a `PUT Bucket versioning` request.

For more information, see [PutBucketVersioning in the Amazon Simple Storage Service API Reference](#).

2. Send a `PUT`, `POST`, or `COPY` request to store an object in the bucket.

When you add an object to a versioning-enabled bucket, Amazon S3 returns the version ID of the object in the `x-amz-version-id` response header, as shown in the following example.

```
x-amz-version-id: 3/L4kqtJlcpXroDTDmJ+rmSpXd3dIbrHY
```

Listing objects in a versioning-enabled bucket

This section provides examples of listing object versions from a versioning-enabled bucket. Amazon S3 stores object version information in the `versions` subresource that is associated with the bucket. For more information, see [Bucket configuration options \(p. 116\)](#). In order to list the objects in a versioning-enabled bucket, you need the `ListBucketVersions` permission.

Using the S3 console

Follow these steps to use the Amazon S3 console to see the different versions of an object.

To see multiple versions 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 **Buckets** list, choose the name of the bucket that contains the object.
3. To see a list of the versions of the objects in the bucket, choose the **Show versions** switch.

For each object version, the console shows a unique version ID, the date and time the object version was created, and other properties. (Objects stored in your bucket before you set the versioning state have a version ID of **null**.)

To list the objects without the versions, choose the **List versions** switch.

You also can view, download, and delete object versions in the object overview pane on the console. For more information, see [Viewing an object overview in the Amazon S3 console \(p. 256\)](#).

Important

You can undelete an object only if it was deleted as the latest (current) version. You can't undelete a previous version of an object that was deleted. For more information, see [Using versioning in S3 buckets \(p. 638\)](#).

Using the AWS SDKs

The examples in this section show how to retrieve an object listing from a versioning-enabled bucket. Each request returns up to 1,000 versions, unless you specify a lower number. If the bucket contains more versions than this limit, you send a series of requests to retrieve the list of all versions. This process of returning results in "pages" is called *pagination*.

To show how pagination works, the examples limit each response to two object versions. After retrieving the first page of results, each example checks to determine whether the version list was truncated. If it was, the example continues retrieving pages until all versions have been retrieved.

Note

The following examples also work with a bucket that isn't versioning-enabled, or for objects that don't have individual versions. In those cases, Amazon S3 returns the object listing with a version ID of **null**.

For information about using other AWS SDKs, see the [AWS Developer Center](#).

Java

For instructions on creating and testing a working sample, see [Testing the Amazon S3 Java Code Examples \(p. 1192\)](#).

```
import com.amazonaws.AmazonServiceException;
import com.amazonaws.SdkClientException;
import com.amazonaws.auth.profile.ProfileCredentialsProvider;
import com.amazonaws.regions.Regions;
import com.amazonaws.services.s3.AmazonS3;
import com.amazonaws.services.s3.AmazonS3ClientBuilder;
import com.amazonaws.services.s3.model.ListVersionsRequest;
import com.amazonaws.services.s3.model.S3VersionSummary;
import com.amazonaws.services.s3.model.VersionListing;

public class ListKeysVersioningEnabledBucket {

    public static void main(String[] args) {
        Regions clientRegion = Regions.DEFAULT_REGION;
        String bucketName = "*** Bucket name ***";
```

```

try {
    AmazonS3 s3Client = AmazonS3ClientBuilder.standard()
        .withCredentials(new ProfileCredentialsProvider())
        .withRegion(clientRegion)
        .build();

    // Retrieve the list of versions. If the bucket contains more versions
    // than the specified maximum number of results, Amazon S3 returns
    // one page of results per request.
    ListVersionsRequest request = new ListVersionsRequest()
        .withBucketName(bucketName)
        .withMaxResults(2);
    VersionListing versionListing = s3Client.listVersions(request);
    int numVersions = 0, numPages = 0;
    while (true) {
        numPages++;
        for (S3VersionSummary objectSummary :
            versionListing.getVersionSummaries()) {
            System.out.printf("Retrieved object %s, version %s\n",
                objectSummary.getKey(),
                objectSummary.getVersionId());
            numVersions++;
        }
        // Check whether there are more pages of versions to retrieve. If
        // there are, retrieve them. Otherwise, exit the loop.
        if (versionListing.isTruncated()) {
            versionListing = s3Client.listNextBatchOfVersions(versionListing);
        } else {
            break;
        }
    }
    System.out.println(numVersions + " object versions retrieved in " +
numPages + " pages");
} catch (AmazonServiceException e) {
    // The call was transmitted successfully, but Amazon S3 couldn't process
    // it, so it returned an error response.
    e.printStackTrace();
} catch (SdkClientException e) {
    // Amazon S3 couldn't be contacted for a response, or the client
    // couldn't parse the response from Amazon S3.
    e.printStackTrace();
}
}

```

.NET

For information about how to create and test a working sample, see [Running the Amazon S3 .NET Code Examples \(p. 1193\)](#).

```
using Amazon;
using Amazon.S3;
using Amazon.S3.Model;
using System;
using System.Threading.Tasks;

namespace Amazon.DocSamples.S3
{
    class ListObjectsVersioningEnabledBucketTest
    {
        static string bucketName = "*** bucket name ***";
        // Specify your bucket region (an example region is shown).
        private static readonly RegionEndpoint bucketRegion = RegionEndpoint.USWest2;
        private static IAmazonS3 s3Client;
```

```
public static void Main(string[] args)
{
    s3Client = new AmazonS3Client(bucketRegion);
    GetObjectListWithAllVersionsAsync().Wait();
}

static async Task GetObjectListWithAllVersionsAsync()
{
    try
    {
        ListVersionsRequest request = new ListVersionsRequest()
        {
            BucketName = bucketName,
            // You can optionally specify key name prefix in the request
            // if you want list of object versions of a specific object.

            // For this example we limit response to return list of 2 versions.
            MaxKeys = 2
        };
        do
        {
            ListVersionsResponse response = await
s3Client.ListVersionsAsync(request);
            // Process response.
            foreach (S3ObjectVersion entry in response.Versions)
            {
                Console.WriteLine("key = {0} size = {1}",
entry.Key, entry.Size);
            }

            // If response is truncated, set the marker to get the next
            // set of keys.
            if (response.IsTruncated)
            {
                request.KeyMarker = response.NextKeyMarker;
                request.VersionIdMarker = response.NextVersionIdMarker;
            }
            else
            {
                request = null;
            }
        } while (request != null);
    }
    catch (AmazonS3Exception e)
    {
        Console.WriteLine("Error encountered on server. Message:'{0}' when
writing an object", e.Message);
    }
    catch (Exception e)
    {
        Console.WriteLine("Unknown encountered on server. Message:'{0}' when
writing an object", e.Message);
    }
}
}
```

Using the REST API

Example — Listing all object versions in a bucket

To list all the versions of all the objects in a bucket, you use the `versions` subresource in a `GET Bucket` request. Amazon S3 can retrieve a maximum of 1,000 objects, and each object version counts fully as an

object. Therefore, if a bucket contains two keys (for example, `photo.gif` and `picture.jpg`), and the first key has 990 versions and the second key has 400 versions, a single request would retrieve all 990 versions of `photo.gif` and only the most recent 10 versions of `picture.jpg`.

Amazon S3 returns object versions in the order in which they were stored, with the most recently stored returned first.

In a `GET Bucket` request, include the `versions` subresource.

```
GET /?versions HTTP/1.1
Host: bucketName.s3.amazonaws.com
Date: Wed, 28 Oct 2009 22:32:00 +0000
Authorization: AWS AKIAIOSFODNN7EXAMPLE:0RQf4/cRonhpaBX5sCYVf1bNRuU=
```

Example — Retrieving all versions of a key

To retrieve a subset of object versions, you use the request parameters for `GET Bucket`. For more information, see [GET Bucket](#).

1. Set the `prefix` parameter to the key of the object that you want to retrieve.
2. Send a `GET Bucket` request using the `versions` subresource and `prefix`.

```
GET /?versions&prefix=objectName HTTP/1.1
```

Example — Retrieving objects using a prefix

The following example retrieves objects whose key is or begins with `myObject`.

```
GET /?versions&prefix=myObject HTTP/1.1
Host: bucket.s3.amazonaws.com
Date: Wed, 28 Oct 2009 22:32:00 GMT
Authorization: AWS AKIAIOSFODNN7EXAMPLE:0RQf4/cRonhpaBX5sCYVf1bNRuU=
```

You can use the other request parameters to retrieve a subset of all versions of the object. For more information, see [GET Bucket](#) in the *Amazon Simple Storage Service API Reference*.

Example — Retrieving a listing of additional objects if the response is truncated

If the number of objects that could be returned in a `GET` request exceeds the value of `max-keys`, the response contains `<iTruncated>true</iTruncated>`, and includes the first key (`in NextKeyMarker`) and the first version ID (`in NextVersionIdMarker`) that satisfy the request, but were not returned. You use those returned values as the starting position in a subsequent request to retrieve the additional objects that satisfy the `GET` request.

Use the following process to retrieve additional objects that satisfy the original `GET Bucket versions` request from a bucket. For more information about `key-marker`, `version-id-marker`, `NextKeyMarker`, and `NextVersionIdMarker`, see [GET Bucket](#) in the *Amazon Simple Storage Service API Reference*.

The following are additional responses that satisfy the original `GET` request:

- Set the value of `key-marker` to the key returned in `NextKeyMarker` in the previous response.
- Set the value of `version-id-marker` to the version ID returned in `NextVersionIdMarker` in the previous response.
- Send a `GET Bucket versions` request using `key-marker` and `version-id-marker`.

Example — Retrieving objects starting with a specified key and version ID

```
GET /?versions&key-marker=myObject&version-id-marker=298459348571 HTTP/1.1
Host: bucket.s3.amazonaws.com
Date: Wed, 28 Oct 2009 22:32:00 GMT
Authorization: AWS AKIAIOSFODNN7EXAMPLE:0RQf4/cRonhpaBX5sCYVf1bNRuU=
```

Using the AWS CLI

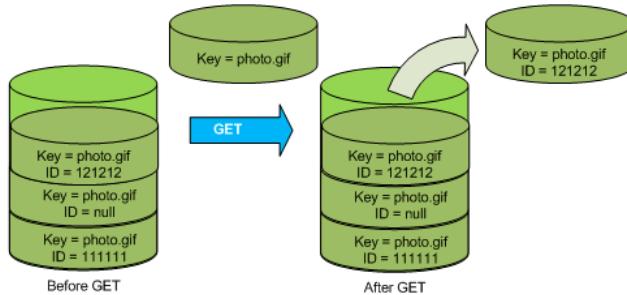
The following command returns metadata about all versions of the objects in a bucket.

```
aws s3api list-object-versions --bucket DOC-EXAMPLE-BUCKET
```

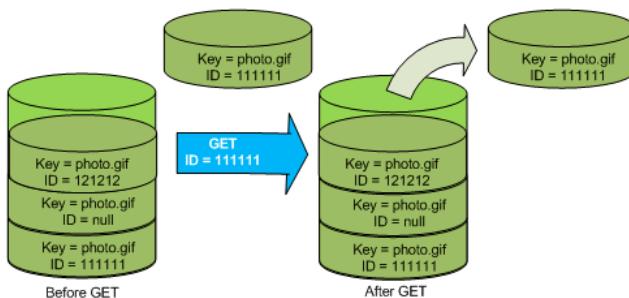
For more information about `list-object-versions` see [list-object-versions](#) in the *AWS CLI Command Reference*.

Retrieving object versions from a versioning-enabled bucket

Versioning in Amazon S3 is a way of keeping multiple variants of an object in the same bucket. A simple GET request retrieves the current version of an object. The following figure shows how GET returns the current version of the object, `photo.gif`.



To retrieve a specific version, you have to specify its version ID. The following figure shows that a `GET` `versionId` request retrieves the specified version of the object (not necessarily the current one).



You can retrieve object versions in Amazon S3 using the console, AWS SDKs, or REST API.

Using the S3 console

1. Sign in to the AWS Management Console and open the Amazon S3 console at <https://console.aws.amazon.com/s3/>.
2. In the **Buckets** list, choose the name of the bucket that contains the object.
3. In the **Objects** list, choose the name of the object.

4. Choose **Versions**.

Amazon S3 shows all the versions for the object.

5. Select the check box next to the **Version ID** for the versions that you want to retrieve.
6. Choose **Actions**, choose **Download**, and save the object.

You also can view, download, and delete object versions in the object overview panel. For more information, see [Viewing an object overview in the Amazon S3 console \(p. 256\)](#).

Important

You can undelete an object only if it was deleted as the latest (current) version. You can't undelete a previous version of an object that was deleted. For more information, see [Using versioning in S3 buckets \(p. 638\)](#).

Using the AWS SDKs

The examples for uploading objects in nonversioned and versioning-enabled buckets are the same. However, for versioning-enabled buckets, Amazon S3 assigns a version number. Otherwise, the version number is null.

For examples of downloading objects using AWS SDKs for Java, .NET, and PHP, see [Downloading objects](#).

Using the REST API

To retrieve a specific object version

1. Set `versionId` to the ID of the version of the object that you want to retrieve.
2. Send a `GET Object versionId` request.

Example — Retrieving a versioned object

The following request retrieves version `L4kqtJlcpXroDTDmpUMLUo` of `my-image.jpg`.

```
GET /my-image.jpg?versionId=L4kqtJlcpXroDTDmpUMLUo HTTP/1.1
Host: bucket.s3.amazonaws.com
Date: Wed, 28 Oct 2009 22:32:00 GMT
Authorization: AWS AKIAIOSFODNN7EXAMPLE:0RQf4/cRonhpaBX5sCYVf1bNRuU=
```

You can retrieve just the metadata of an object (not the content). For information, see [the section called "Retrieving version metadata" \(p. 656\)](#).

For information about restoring a previous object version, see [the section called "Restoring previous versions" \(p. 657\)](#).

Retrieving the metadata of an object version

If you only want to retrieve the metadata of an object (and not its content), you use the `HEAD` operation. By default, you get the metadata of the most recent version. To retrieve the metadata of a specific object version, you specify its version ID.

To retrieve the metadata of an object version

1. Set `versionId` to the ID of the version of the object whose metadata you want to retrieve.
2. Send a `HEAD Object versionId` request.

Example — Retrieving the metadata of a versioned object

The following request retrieves the metadata of version 3HL4kqCxf3vjVBH40Nrjfkd of my-image.jpg.

```
HEAD /my-image.jpg?versionId=3HL4kqCxf3vjVBH40Nrjfkd HTTP/1.1
Host: bucket.s3.amazonaws.com
Date: Wed, 28 Oct 2009 22:32:00 GMT
Authorization: AWS AKIAIOSFODNN7EXAMPLE:0RQf4/cRonhpaBX5sCYVf1bNRuU=
```

The following shows a sample response.

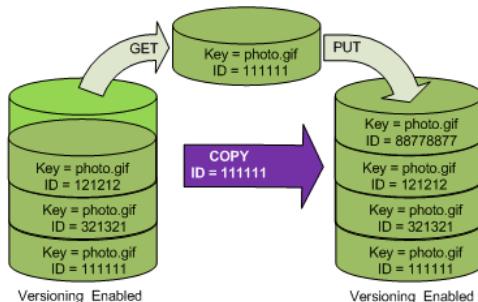
```
HTTP/1.1 200 OK
x-amz-id-2: ef8yU9AS1ed4OpIszzj7UDNEHGran
x-amz-request-id: 318BC8BC143432E5
x-amz-version-id: 3HL4kqtJlcpXroDTDmjVBH40Nrjfkd
Date: Wed, 28 Oct 2009 22:32:00 GMT
Last-Modified: Sun, 1 Jan 2006 12:00:00 GMT
ETag: "fba9dede5f27731c9771645a39863328"
Content-Length: 434234
Content-Type: text/plain
Connection: close
Server: AmazonS3
```

Restoring previous versions

You can use versioning to retrieve previous versions of an object. There are two approaches to doing so:

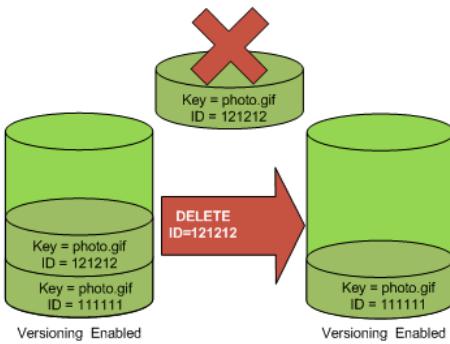
- Copy a previous version of the object into the same bucket.
The copied object becomes the current version of that object and all object versions are preserved.
- Permanently delete the current version of the object.
When you delete the current object version, you, in effect, turn the previous version into the current version of that object.

Because all object versions are preserved, you can make any earlier version the current version by copying a specific version of the object into the same bucket. In the following figure, the source object (ID = 111111) is copied into the same bucket. Amazon S3 supplies a new ID (88778877) and it becomes the current version of the object. So, the bucket has both the original object version (111111) and its copy (88778877). For more information about getting a previous version and then uploading it to make it the current version, see [Retrieving object versions from a versioning-enabled bucket](#) and [Uploading objects](#).



A subsequent GET retrieves version 88778877.

The following figure shows how deleting the current version (121212) of an object leaves the previous version (111111) as the current object. For more information about deleting an object, see [Deleting a single object](#).



A subsequent **GET** retrieves version 111111.

To restore previous object versions

Using the S3 console

1. Sign in to the AWS Management Console and open the Amazon S3 console at <https://console.aws.amazon.com/s3/>.
2. In the **Buckets** list, choose the name of the bucket that contains the object.
3. In the **Objects** list, choose the name of the object.
4. Choose **Versions**.

Amazon S3 shows all the versions for the object.

5. Select the check box next to the **Version ID** for the versions that you want to retrieve.
6. Choose **Actions**, choose **Download**, and save the object.

You also can view, download, and delete object versions in the object overview panel. For more information, see [Viewing an object overview in the Amazon S3 console \(p. 256\)](#).

Important

You can undelete an object only if it was deleted as the latest (current) version. You can't undelete a previous version of an object that was deleted. For more information, see [Using versioning in S3 buckets \(p. 638\)](#).

Using the AWS SDKs

For information about using other AWS SDKs, see the [AWS Developer Center](#).

Python

For instructions on how to create and test a working sample, see [Using the AWS SDK for Python \(Boto\) \(p. 1196\)](#).

The following Python code example restores a versioned object's previous version by deleting all versions that occurred after the specified rollback version.

```
def rollback_object(bucket, object_key, version_id):  
    """  
    Rolls back an object to an earlier version by deleting all versions that  
    occurred after the specified rollback version.  
    """
```

```

Usage is shown in the usage_demo_single_object function at the end of this module.

:param bucket: The bucket that holds the object to roll back.
:param object_key: The object to roll back.
:param version_id: The version ID to roll back to.
"""
# Versions must be sorted by last_modified date because delete markers are
# at the end of the list even when they are interspersed in time.
versions = sorted(bucket.object_versions.filter(Prefix=object_key),
                   key=attrgetter('last_modified'), reverse=True)

logger.debug(
    "Got versions:\n%s",
    '\n'.join([f"\t{version.version_id}, last modified {version.last_modified}"
              for version in versions]))

if version_id in [ver.version_id for ver in versions]:
    print(f"Rolling back to version {version_id}")
    for version in versions:
        if version.version_id != version_id:
            version.delete()
            print(f"Deleted version {version.version_id}")
        else:
            break

    print(f"Active version is now {bucket.Object(object_key).version_id}")
else:
    raise KeyError(f"{version_id} was not found in the list of versions for "
                  f"{object_key}.")

```

Deleting object versions from a versioning-enabled bucket

You can delete object versions from Amazon S3 buckets whenever you want. You can also define lifecycle configuration rules for objects that have a well-defined lifecycle to request Amazon S3 to expire current object versions or permanently remove noncurrent object versions. When your bucket is version-enabled or versioning is suspended, the lifecycle configuration actions work as follows:

- The `Expiration` action applies to the current object version. Instead of deleting the current object version, Amazon S3 retains the current version as a noncurrent version by adding a *delete marker*, which then becomes the current version.
- The `NoncurrentVersionExpiration` action applies to noncurrent object versions, and Amazon S3 permanently removes these object versions. You cannot recover permanently removed objects.

For more information, see [Managing your storage lifecycle \(p. 701\)](#).

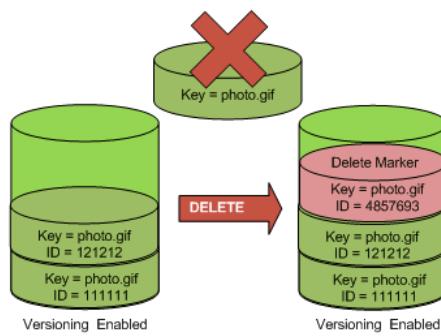
Delete request use cases

A `DELETE` request has the following use cases:

- When versioning is enabled, a simple `DELETE` cannot permanently delete an object. Instead, Amazon S3 inserts a delete marker in the bucket, and that marker becomes the current version of the object with a new ID.

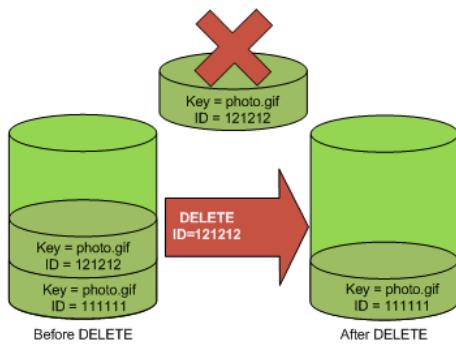
When you try to `GET` an object whose current version is a delete marker, Amazon S3 behaves as though the object has been deleted (even though it has not been erased) and returns a 404 error. For more information, see [Working with delete markers \(p. 662\)](#).

The following figure shows that a simple `DELETE` does not actually remove the specified object. Instead, Amazon S3 inserts a delete marker.



- To delete versioned objects permanently, you must use `DELETE Object versionId`.

The following figure shows that deleting a specified object version permanently removes that object.



To delete object versions

You can delete object versions in Amazon S3 using the console, AWS SDKs, the REST API, or the AWS Command Line Interface.

Using the S3 console

1. Sign in to the AWS Management Console and open the Amazon S3 console at <https://console.aws.amazon.com/s3/>.
2. In the **Buckets** list, choose the name of the bucket that contains the object.
3. In the **Objects** list, choose the name of the object.
4. Choose **Versions**.

Amazon S3 shows all the versions for the object.

5. Select the check box next to the **Version ID** for the versions that you want to permanently delete.
6. Choose **Delete**.
7. In **Permanently delete objects?**, enter **permanently delete**.

Warning

When you permanently delete an object version, the action cannot be undone.

8. Choose **Delete objects**.

Amazon S3 deletes the object version.

Using the AWS SDKs

For examples of deleting objects using the AWS SDKs for Java, .NET, and PHP, see [Deleting Amazon S3 objects \(p. 223\)](#). The examples for deleting objects in nonversioned and versioning-enabled buckets are the same. However, for versioning-enabled buckets, Amazon S3 assigns a version number. Otherwise, the version number is null.

For information about using other AWS SDKs, see the [AWS Developer Center](#).

Python

For instructions on how to create and test a working sample, see [Using the AWS SDK for Python \(Boto\) \(p. 1196\)](#).

The following Python code example permanently deletes a versioned object by deleting all of its versions.

```
def permanently_delete_object(bucket, object_key):
    """
    Permanently deletes a versioned object by deleting all of its versions.

    Usage is shown in the usage_demo_single_object function at the end of this module.

    :param bucket: The bucket that contains the object.
    :param object_key: The object to delete.
    """
    try:
        bucket.object_versions.filter(Prefix=object_key).delete()
        logger.info("Permanently deleted all versions of object %s.", object_key)
    except ClientError:
        logger.exception("Couldn't delete all versions of %s.", object_key)
        raise
```

Using the REST API

To delete a specific version of an object

- In a `DELETE`, specify a version ID.

Example — Deleting a specific version

The following example deletes version `UIORUnfnd89493jJFJ` of `photo.gif`.

```
DELETE /photo.gif?versionId=UIORUnfnd89493jJFJ HTTP/1.1
Host: bucket.s3.amazonaws.com
Date: Wed, 12 Oct 2009 17:50:00 GMT
Authorization: AWS AKIAIOSFODNN7EXAMPLE:xQE0diMbLRepdf3YB+FIEXAMPLE=
Content-Type: text/plain
Content-Length: 0
```

Using the AWS CLI

The following command deletes an object named `test.txt` from a bucket named `DOC-EXAMPLE-BUCKET1`. To remove a specific version of an object, you must be the bucket owner and you must use the `versionId` subresource.

```
aws s3api delete-object --bucket DOC-EXAMPLE-BUCKET1 --key test.txt --version-id versionID
```

For more information about `delete-object` see [delete-object](#) in the *AWS CLI Command Reference*.

For more information about deleting object versions, see the following topics:

- [Working with delete markers \(p. 662\)](#)
- [Removing delete markers to make an older version current \(p. 663\)](#)
- [Deleting an object from an MFA delete-enabled bucket \(p. 665\)](#)

Working with delete markers

A *delete marker* in Amazon S3 is a placeholder (or marker) for a versioned object that was named in a simple `DELETE` request. Because the object is in a versioning-enabled bucket, the object is not deleted. But the delete marker makes Amazon S3 behave as if it is deleted.

A delete marker has a key name (or key) and version ID like any other object. However, a delete marker differs from other objects in the following ways:

- It does not have data associated with it.
- It is not associated with an access control list (ACL) value.
- It does not retrieve anything from a `GET` request because it has no data; you get a 404 error.
- The only operation that you can use on a delete marker is an Amazon S3 API `DELETE` call. To do this, you must make the `DELETE` request using an AWS Identity and Access Management (IAM) user or role with the appropriate permissions.

Delete markers accrue a nominal charge for storage in Amazon S3. The storage size of a delete marker is equal to the size of the key name of the delete marker. A key name is a sequence of Unicode characters. The UTF-8 encoding adds 1–4 bytes of storage to your bucket for each character in the name.

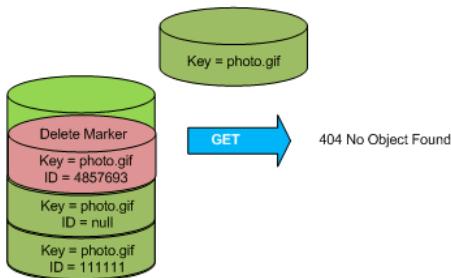
For more information about key names, see [Creating object key names \(p. 150\)](#). For information about deleting a delete marker, see [Managing delete markers \(p. 663\)](#).

Only Amazon S3 can create a delete marker, and it does so whenever you send a `DELETE Object` request on an object in a versioning-enabled or suspended bucket. The object named in the `DELETE` request is not actually deleted. Instead, the delete marker becomes the current version of the object. The object's key name (or key) becomes the key of the delete marker. If you try to get an object and its current version is a delete marker, Amazon S3 responds with the following:

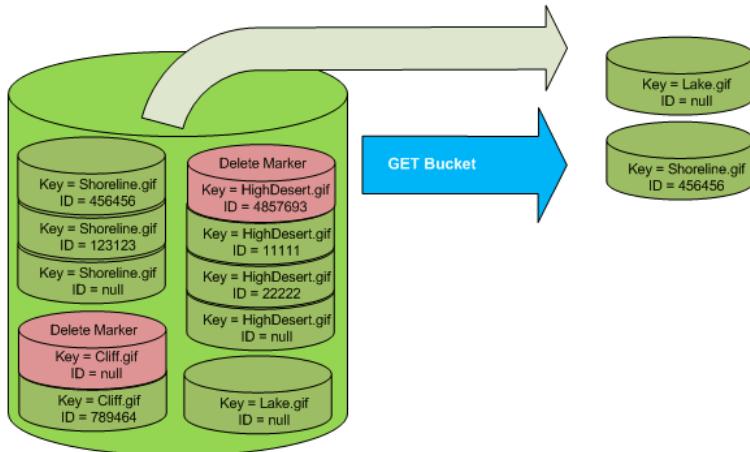
- A 404 (Object not found) error
- A response header, `x-amz-delete-marker: true`

The response header tells you that the object accessed was a delete marker. This response header never returns `false`. If the value is `false`, Amazon S3 does not include this response header in the response.

The following figure shows how a simple `GET` on an object whose current version is a delete marker, returns a 404 No Object Found error.



The only way to list delete markers (and other versions of an object) is by using the `versions` subresource in a `GET Bucket` `versions` request. A simple `GET` does not retrieve delete marker objects. The following figure shows that a `GET Bucket` request does not return objects whose current version is a delete marker.



Managing delete markers

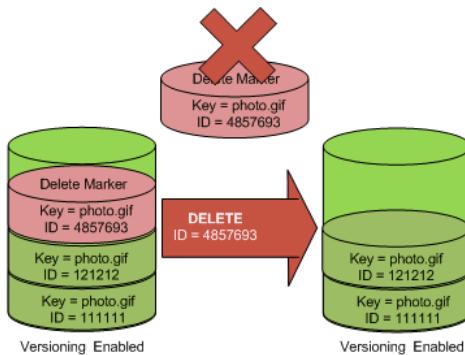
Configuring lifecycle to clean up expired delete markers automatically

An expired object delete marker is one where all object versions are deleted and only a single delete marker remains. If the lifecycle policy is set to delete current versions, or the `ExpiredObjectDeleteMarker` action is explicitly set, Amazon S3 removes the expired object's delete marker. For an example, see [Example 7: Removing expired object delete markers \(p. 736\)](#).

Removing delete markers to make an older version current

When you delete an object in a versioning-enabled bucket, all versions remain in the bucket, and Amazon S3 creates a delete marker for the object. To undelete the object, you must delete this delete marker. For more information about versioning and delete markers, see [Using versioning in S3 buckets \(p. 638\)](#).

To delete a delete marker permanently, you must include its version ID in a `DeleteObject` `versionId` request. The following figure shows how a `DeleteObject` `versionId` request permanently removes a delete marker.

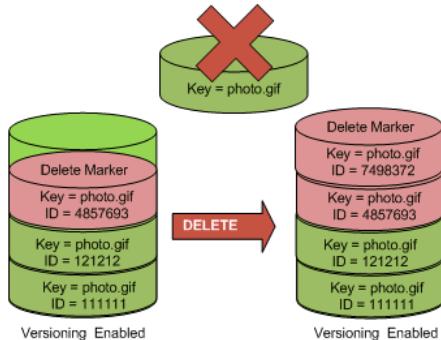


The effect of removing the delete marker is that a simple `GET` request will now retrieve the current version ID (121212) of the object.

Note

If you use a `DeleteObject` request where the current version is a delete marker (without specifying the version ID of the delete marker), Amazon S3 does not delete the delete marker, but instead `PUTs` another delete marker.

To delete a delete marker with a `NULL` version ID, you must pass the `NULL` as the version ID in the `DeleteObject` request. The following figure shows how a simple `DeleteObject` request made without a version ID where the current version is a delete marker, removes nothing, but instead adds an additional delete marker with a unique version ID (7498372).



Using the S3 console

Use the following steps to recover deleted objects that are not folders from your S3 bucket, including objects that are within those folders.

1. Sign in to the AWS Management Console and open the Amazon S3 console at <https://console.aws.amazon.com/s3/>.
2. In the **Buckets** list, choose the name of the bucket that you want.
3. To see a list of the **versions** of the objects in the bucket, choose the **List versions** switch. You'll be able to see the delete markers for deleted objects.
4. To undelete an object, you must delete the delete marker. Select the check box next to the **delete marker** of the object to recover, and then choose **Delete**.
5. Confirm the deletion on the **Delete objects** page.
 - a. For **Permanently delete objects?** enter **permanently delete**.
 - b. Choose **Delete objects**.

Note

You can't use the Amazon S3 console to undelete folders. You must use the AWS CLI or SDK. For examples, see [How can I retrieve an Amazon S3 object that was deleted in a versioning-enabled bucket?](#) in the AWS Knowledge Center.

Using the REST API

To permanently remove a delete marker

1. Set `versionId` to the ID of the version to the delete marker you want to remove.
2. Send a `DELETE Object versionId` request.

Example — Removing a delete marker

The following example removes the delete marker for `photo.gif` version 4857693.

```
DELETE /photo.gif?versionId=4857693 HTTP/1.1
Host: bucket.s3.amazonaws.com
Date: Wed, 28 Oct 2009 22:32:00 GMT
Authorization: AWS AKIAIOSFODNN7EXAMPLE:0RQf4/cRonhpaBX5sCYVf1bNRuU=
```

When you delete a delete marker, Amazon S3 includes the following in the response.

```
204 NoContent
x-amz-version-id: versionID
x-amz-delete-marker: true
```

Using the AWS SDKs

For information about using other AWS SDKs, see the [AWSDeveloper Center](#).

Python

For instructions on how to create and test a working sample, see [Using the AWS SDK for Python \(Boto\) \(p. 1196\)](#).

The following Python code example demonstrates how to remove a delete marker from an object and thus makes the most recent non-current version, the current version of the object.

```
def revive_object(bucket, object_key):
    """
    Revives a versioned object that was deleted by removing the object's active
    delete marker.
    A versioned object presents as deleted when its latest version is a delete marker.
    By removing the delete marker, we make the previous version the latest version
    and the object then presents as *not* deleted.

    Usage is shown in the usage_demo_single_object function at the end of this module.

    :param bucket: The bucket that contains the object.
    :param object_key: The object to revive.
    """
    # Get the latest version for the object.
    response = s3.meta.client.list_object_versions(
        Bucket=bucket.name, Prefix=object_key, MaxKeys=1)

    if 'DeleteMarkers' in response:
        latest_version = response['DeleteMarkers'][0]
        if latest_version['IsLatest']:
            logger.info("Object %s was indeed deleted on %s. Let's revive it.",
                        object_key, latest_version['LastModified'])
            obj = bucket.Object(object_key)
            obj.Version(latest_version['VersionId']).delete()
            logger.info("Revived %s, active version is now %s with body '%s'",
                        object_key, obj.version_id, obj.get()['Body'].read())
        else:
            logger.warning("Delete marker is not the latest version for %s!",
                           object_key)
    elif 'Versions' in response:
        logger.warning("Got an active version for %s, nothing to do.", object_key)
    else:
        logger.error("Couldn't get any version info for %s.", object_key)
```

Deleting an object from an MFA delete-enabled bucket

If a bucket's versioning configuration is MFA delete enabled, the bucket owner must include the `x-amz-mfa` request header in requests to permanently delete an object version or change the versioning state of the bucket. Requests that include `x-amz-mfa` must use HTTPS.

The header's value is the concatenation of your authentication device's serial number, a space, and the authentication code displayed on it. If you don't include this request header, the request fails.

For more information about authentication devices, see [Multi-factor Authentication](#).

Example — Deleting an object from an MFA delete-enabled bucket

The following example deletes `my-image.jpg` (with the specified version), which is in a bucket configured with MFA delete enabled.

Note the space between `[SerialNumber]` and `[AuthenticationCode]`. For more information, see [DeleteObject](#) in the *Amazon Simple Storage Service API Reference*.

```
DELETE /my-image.jpg?versionId=3HL4kqCxf3vjVBH40Nrjfkd HTTPS/1.1
Host: bucketName.s3.amazonaws.com
x-amz-mfa: 20899872 301749
Date: Wed, 28 Oct 2009 22:32:00 GMT
Authorization: AWS AKIAIOSFODNN7EXAMPLE:0RQf4/cRonhpaBX5sCYVf1bNRuU=
```

For more information about enabling MFA delete, see [Configuring MFA delete \(p. 648\)](#).

Configuring versioned object permissions

Permissions for objects in Amazon S3 are set at the version level. Each version has its own object owner. The AWS account that creates the object version is the owner. So, you can set different permissions for different versions of the same object. To do so, you must specify the version ID of the object whose permissions you want to set in a `PUT Object versionId acl` request. For a detailed description and instructions on using ACLs, see [Identity and access management in Amazon S3 \(p. 394\)](#).

Example — Setting permissions for an object version

The following request sets the permission of the grantee, `BucketOwner@amazon.com`, to `FULL_CONTROL` on the key, `my-image.jpg`, version ID, `3HL4kqtJvjVBH40Nrjfkd`.

```
PUT /my-image.jpg?acl&versionId=3HL4kqtJvjVBH40Nrjfkd HTTP/1.1
Host: bucket.s3.amazonaws.com
Date: Wed, 28 Oct 2009 22:32:00 GMT
Authorization: AWS AKIAIOSFODNN7EXAMPLE:0RQf4/cRonhpaBX5sCYVf1bNRuU=
Content-Length: 124

<AccessControlPolicy>
  <Owner>
    <ID>75cc57f09aa0c8caeab4f8c24e99d10f8e7faeefbf76c078efc7c6caea54ba06a</ID>
    <DisplayName>mtd@amazon.com</DisplayName>
  </Owner>
  <AccessControlList>
    <Grant>
      <Grantee xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:type="CanonicalUser">
        <ID>a9a7b886d6fd24a52fe8ca5bef65f89a64e0193f23000e241bf9b1c61be666e9</ID>
        <DisplayName>BucketOwner@amazon.com</DisplayName>
      </Grantee>
      <Permission>FULL_CONTROL</Permission>
    </Grant>
  </AccessControlList>
</AccessControlPolicy>
```

Likewise, to get the permissions of a specific object version, you must specify its version ID in a `GET Object versionId acl` request. You need to include the version ID because, by default, `GET Object acl` returns the permissions of the current version of the object.

Example — Retrieving the permissions for a specified object version

In the following example, Amazon S3 returns the permissions for the key, `my-image.jpg`, version ID, `DVBH40Nr8X8gUMLUo`.

```
GET /my-image.jpg?versionId=DVBH40Nr8XgUMLUo&acl HTTP/1.1
Host: bucket.s3.amazonaws.com
Date: Wed, 28 Oct 2009 22:32:00 GMT
Authorization: AWS AKIAIOSFODNN7EXAMPLE:0RQf4/cRonhpaBX5sCYVf1bNRuU
```

For more information, see [GetObjectAcl](#) in the *Amazon Simple Storage Service API Reference*.

Working with objects in a versioning-suspended bucket

In Amazon S3, you can suspend versioning to stop accruing new versions of the same object in a bucket. You might do this because you only want a single version of an object in a bucket. Or, you might not want to accrue charges for multiple versions.

When you suspend versioning, existing objects in your bucket do not change. What changes is how Amazon S3 handles objects in future requests. The topics in this section explain various object operations in a versioning-suspended bucket, including adding, retrieving, and deleting objects.

For more information about S3 Versioning, see [Using versioning in S3 buckets \(p. 638\)](#). For more information about retrieving object versions, see [Retrieving object versions from a versioning-enabled bucket \(p. 655\)](#).

Topics

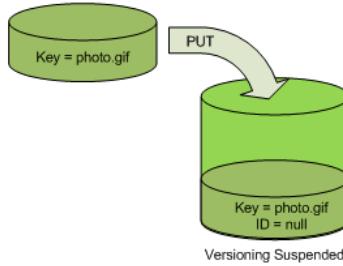
- [Adding objects to versioning-suspended buckets \(p. 667\)](#)
- [Retrieving objects from versioning-suspended buckets \(p. 668\)](#)
- [Deleting objects from versioning-suspended buckets \(p. 668\)](#)

Adding objects to versioning-suspended buckets

You can add objects to versioning-suspended buckets in Amazon S3 to create the object with a null version ID or overwrite any object version with a matching version ID.

After you suspend versioning on a bucket, Amazon S3 automatically adds a null version ID to every subsequent object stored thereafter (using `PUT`, `POST`, or `COPY`) in that bucket.

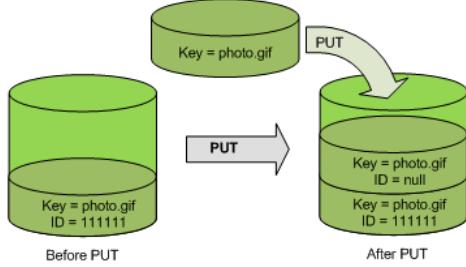
The following figure shows how Amazon S3 adds the version ID of `null` to an object when it is added to a version-suspended bucket.



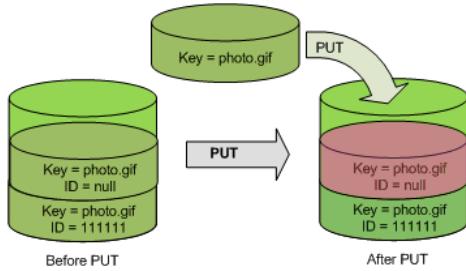
If a null version is already in the bucket and you add another object with the same key, the added object overwrites the original null version.

If there are versioned objects in the bucket, the version you `PUT` becomes the current version of the object. The following figure shows how adding an object to a bucket that contains versioned objects does not overwrite the object already in the bucket.

In this case, version 111111 was already in the bucket. Amazon S3 attaches a version ID of null to the object being added and stores it in the bucket. Version 111111 is not overwritten.



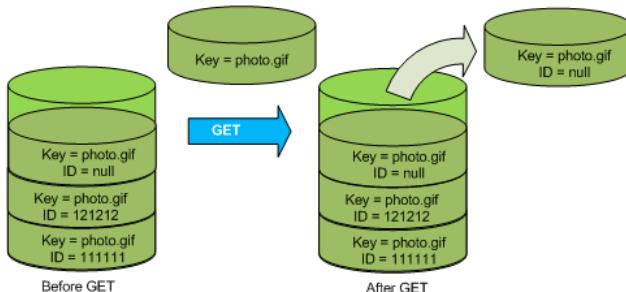
If a null version already exists in a bucket, the null version is overwritten, as shown in the following figure.



Although the key and version ID (null) of the null version are the same before and after the `PUT`, the contents of the null version originally stored in the bucket are replaced by the contents of the object `PUT` into the bucket.

Retrieving objects from versioning-suspended buckets

A `GET` Object request returns the current version of an object whether you've enabled versioning on a bucket or not. The following figure shows how a simple `GET` returns the current version of an object.



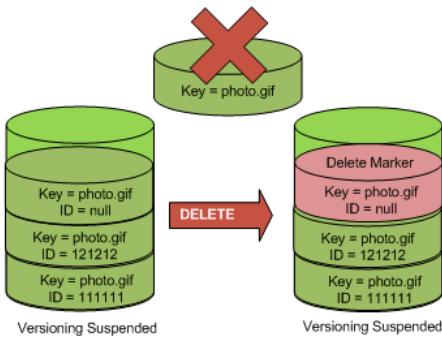
Deleting objects from versioning-suspended buckets

You can delete objects from versioning-suspended buckets to remove an object with a null version ID.

If versioning is suspended for a bucket, a `DELETE` request:

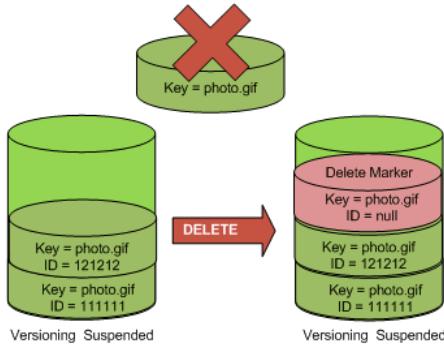
- Can only remove an object whose version ID is `null`.
- Doesn't remove anything if there isn't a null version of the object in the bucket.
- Inserts a delete marker into the bucket.

The following figure shows how a simple `DELETE` removes a null version. Amazon S3 inserts a delete marker in its place with a version ID of `null`.

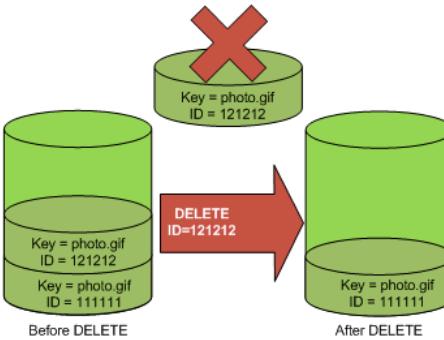


Remember that a delete marker doesn't have content, so you lose the content of the null version when a delete marker replaces it.

The following figure shows a bucket that doesn't have a null version. In this case, the `DELETE` removes nothing; Amazon S3 just inserts a delete marker.



Even in a versioning-suspended bucket, the bucket owner can permanently delete a specified version by including the version ID in the `DELETE` request. The following figure shows that deleting a specified object version permanently removes that version of the object. Only the bucket owner can delete a specified object version.



Using AWS Backup for Amazon S3

Amazon S3 is natively integrated with AWS Backup, a fully managed, policy-based service that you can use to centrally define backup policies to protect your data in Amazon S3. After you define your backup policies and assign Amazon S3 resources to the policies, AWS Backup automates the creation of Amazon S3 backups and securely stores the backups in an encrypted backup vault that you designate in your backup plan.

When using AWS Backup for Amazon S3, you can perform the following actions:

- Create continuous backups and periodic backups. Continuous backups are useful for point-in-time restore, and periodic backups are useful to meet your long-term data-retention needs.
- Automate backup scheduling and retention by centrally configuring backup policies.
- Restore backups of Amazon S3 data to a point in time that you specify.

Along with AWS Backup, you can use S3 Versioning and Amazon S3 Replication to help recover from accidental deletions and perform your own self-recovery operations.

Prerequisites

You must activate [S3 Versioning](#) on your bucket before AWS Backup can back it up.

Note

We recommend that you [set a lifecycle expiration rule for versioning-enabled buckets](#) that are being backed up. If you do not set a lifecycle expiration period, your Amazon S3 storage costs might increase because AWS Backup retains all versions of your Amazon S3 data.

Getting started

To get started with AWS Backup for Amazon S3, see [Creating Amazon S3 backups](#) in the *AWS Backup Developer Guide*.

Restrictions and limitations

To learn about the limitations, see [Creating Amazon S3 backups](#) in the *AWS Backup Developer Guide*.

Working with archived objects

When you archive Amazon S3 objects to the S3 Glacier Flexible Retrieval or S3 Glacier Deep Archive storage class, or when objects are archived to the S3 Intelligent-Tiering Archive Access or Deep Archive Access tiers, the objects are not accessible in real time. To restore the objects, you must do the following:

- For objects in the Archive Access and Deep Archive Access tiers, you must initiate the restore request and wait until the object is moved into the Frequent Access tier.
- For objects in the S3 Glacier Flexible Retrieval and S3 Glacier Deep Archive storage classes, you must initiate the restore request and wait until a temporary copy of the object is available.

For more information about how all Amazon S3 storage classes compare, see [Using Amazon S3 storage classes \(p. 688\)](#).

When you are restoring from the S3 Intelligent-Tiering Archive Access tier or S3 Intelligent-Tiering Deep Archive Access tier, the object moves back into the S3 Intelligent-Tiering Frequent Access tier. Afterwards, if the object is not accessed after 30 consecutive days, it automatically moves into the Infrequent Access tier. It moves into the S3 Intelligent-Tiering Archive Access tier after a minimum of 90 consecutive days of no access, and it moves into the Deep Archive Access tier after a minimum of 180 consecutive days of no access.

Note

Unlike in the S3 Glacier Flexible Retrieval and S3 Glacier Deep Archive storage classes, restore requests for S3 Intelligent-Tiering objects don't accept the days value.

When you use S3 Glacier Flexible Retrieval or S3 Glacier Deep Archive, Amazon S3 restores a temporary copy of the object only for the specified duration. After that, it deletes the restored object copy. You can modify the expiration period of a restored copy by reissuing a restore. In this case, Amazon S3 updates the expiration period relative to the current time.

Note

When you restore an archive from S3 Glacier Flexible Retrieval or S3 Glacier Deep Archive, you pay for both the archived object and a copy that you restored temporarily. For information about pricing, see [Amazon S3 pricing](#).

Amazon S3 calculates the expiration time of the restored object copy by adding the number of days specified in the restoration request to the time when the requested restoring is completed. It then rounds the resulting time to the next day at midnight Universal Coordinated Time (UTC). For example, suppose that a restored object copy was created on October 15, 2012 10:30 AM UTC, and the restoration period was specified as 3 days. In this case, the restored copy expires on October 19, 2012 00:00 UTC, at which time Amazon S3 deletes the object copy.

If a temporary copy of the restored object is created, the object's storage class remains the same. (A [HEAD Object](#) or [GetObject](#) API operation request returns S3 Glacier Flexible Retrieval or S3 Glacier Deep Archive as the storage class.)

The time it takes a restore job to finish depends on which archive storage class or storage tier you use and which retrieval option you specify: **Expedited** (only available for S3 Glacier Flexible Retrieval and S3 Intelligent-Tiering Archive Access), **Standard**, or **Bulk**. For more information, see [Archive retrieval options \(p. 671\)](#).

You can be notified when your restore is complete using Amazon S3 Event Notifications. For more information, see [Amazon S3 Event Notifications \(p. 1017\)](#).

When required, you can restore large segments of the data stored for a secondary copy. However, keep in mind that the S3 Glacier Flexible Retrieval and S3 Glacier Deep Archive storage classes and the Archive Access and Deep Archive Access tiers are designed for 35 random restore requests per pebibyte (PiB) stored per day.

Using Batch Operations with restore requests

To restore more than one Amazon S3 object with a single request, you can use S3 Batch Operations. You provide S3 Batch Operations with a list of objects to operate on. S3 Batch Operations calls the respective API to perform the specified operation. A single Batch Operations job can perform the specified operation on billions of objects containing exabytes of data.

The S3 Batch Operations feature tracks progress, sends notifications, and stores a detailed completion report of all actions, providing a fully managed, auditable, serverless experience. You can use S3 Batch Operations through the AWS Management Console, AWS CLI, AWS SDKs, or REST API. For more information, see [the section called "Batch Operations basics" \(p. 881\)](#).

The following sections provide more information about restoring archived objects.

Topics

- [Archive retrieval options \(p. 671\)](#)
- [Restoring an archived object \(p. 673\)](#)
- [Querying archived objects \(p. 677\)](#)

Archive retrieval options

The following are the available retrieval options when restoring an archived object in Amazon S3:

- **Expedited** - Quickly access your data stored in the S3 Glacier Flexible Retrieval storage class or S3 Intelligent-Tiering Archive Access tier when occasional urgent requests for a subset of archives are required. For all but the largest archived objects (250 MB+), data that is accessed using expedited retrievals is typically made available within 1–5 minutes.

Provisioned capacity helps ensure that retrieval capacity for expedited retrievals from S3 Glacier Flexible Retrieval is available when you need it. For more information, see [Provisioned capacity \(p. 672\)](#).

- **Standard** - Access any of your archived objects within several hours. This is the default option for retrieval requests that do not specify the retrieval option. Standard retrievals typically finish within 3–5 hours for objects stored in the S3 Glacier Flexible Retrieval storage class or S3 Intelligent-Tiering Archive Access tier. They typically finish within 12 hours for objects stored in the S3 Glacier Deep Archive or S3 Intelligent-Tiering Deep Archive Access storage class. Standard retrievals are free for objects stored in S3 Intelligent-Tiering.
- **Bulk** - The lowest-cost retrieval option in Amazon S3 Glacier, enabling you to retrieve large amounts, even petabytes, of data inexpensively. Bulk retrievals typically finish within 5–12 hours for objects stored in the S3 Glacier Flexible Retrieval storage class or S3 Intelligent-Tiering Archive Access tier. They typically finish within 48 hours for objects stored in the S3 Glacier Deep Archive storage class or S3 Intelligent-Tiering Deep Archive Access tier. Bulk retrievals are free for objects stored in S3 Glacier Flexible Retrieval or S3 Intelligent-Tiering.

The following table summarizes the archival retrieval options. For complete information about pricing, see [Amazon S3 pricing](#).

To make an Expedited, Standard, or Bulk retrieval, set the `Tier` request element in the [POST Object restore](#) REST API request to the option you want, or the equivalent in the AWS CLI or AWS SDKs. If you purchased provisioned capacity, all expedited retrievals are automatically served through your provisioned capacity.

You can restore an archived object programmatically or by using the Amazon S3 console. Amazon S3 processes only one restore request at a time per object. You can use both the console and the Amazon S3 API to check the restoration status and to find out when Amazon S3 will delete the restored copy.

For more information, see [Restoring an archived object \(p. 673\)](#).

Provisioned capacity

Provisioned capacity helps ensure that your retrieval capacity for expedited retrievals from S3 Glacier Flexible Retrieval is available when you need it. Each unit of capacity provides that at least three expedited retrievals can be performed every 5 minutes, and it provides up to 150 MB/s of retrieval throughput.

If your workload requires highly reliable and predictable access to a subset of your data in minutes, you should purchase provisioned retrieval capacity. Without provisioned capacity, expedited retrievals might not be accepted during periods of high demand. If you require access to expedited retrievals under all circumstances, we recommend that you purchase provisioned retrieval capacity.

You can purchase provisioned capacity using the Amazon S3 console, the Amazon S3 Glacier console, the [Purchase Provisioned Capacity](#) REST API, the AWS SDKs, or the AWS CLI. For provisioned capacity pricing information, see [Amazon S3 pricing](#).

Upgrading the speed of an in-progress restore

Using Amazon S3 restore speed upgrade, you can change the restore speed to a faster speed while the restore is in progress. A restore speed upgrade overrides an in-progress restore with a faster restore tier. You cannot slow down an in-progress restore.

To upgrade the speed of an in-progress restoration, issue another restore request to the same object that sets a new `Tier` request element in the [POST Object restore](#) REST API, or the equivalent in the AWS CLI or AWS SDKs. When issuing a request to upgrade the restore tier, you must choose a tier that is faster than the tier that the in-progress restore is using. You must not change any other parameters, such as the `Days` request element.

Note

Standard and bulk restores for S3 Intelligent-Tiering are free of charge. However, subsequent restore requests called on an object that is already being restored are billed as a GET request.

You can be notified when your restore is complete using Amazon S3 Event Notifications. For more information, see [Amazon S3 Event Notifications \(p. 1017\)](#). Restores are charged at the price of the upgraded tier. For information about restore pricing, see [Amazon S3 pricing](#).

Restoring an archived object

Amazon S3 objects that are stored in the S3 Glacier Flexible Retrieval or S3 Glacier Deep Archive storage classes are not immediately accessible. To access an object in these storage classes, you must restore a temporary copy of it to its S3 bucket for a specified duration (number of days). For information about using these storage classes, see [Using Amazon S3 storage classes \(p. 688\)](#) and [Managing your storage lifecycle \(p. 701\)](#).

Restored objects from S3 Glacier Flexible Retrieval or S3 Glacier Deep Archive are stored only for the number of days that you specify. If you want a permanent copy of the object, create a copy of it in your Amazon S3 bucket. Unless you make a copy, the object will still be stored in the S3 Glacier Flexible Retrieval or S3 Glacier Deep Archive storage classes.

To calculate the expiry date, Amazon S3 adds the number of days that you specify to the time you request to restore the object, and then rounds to the next day at midnight UTC. This calculation applies to the initial restoration of the object and to any extensions to availability that you request. For example, if an object was restored on Oct 15, 2012 10:30 AM UTC, and the number of days that you specified is 3, the object is available until Oct 19, 2012 00:00 UTC. If, on Oct 16, 2012 11:00 AM UTC, you change the number of days that you want it to be accessible to 1, Amazon S3 makes the restored object available until Oct 18, 2012 00:00 UTC.

When you restore an archived object, you are paying for both the archive and a copy that you restored temporarily. For information about pricing, see [Amazon S3 pricing](#).

You can restore an archived object using the Amazon S3 console, the REST API, the AWS SDKs, and the AWS Command Line Interface (AWS CLI).

Using the S3 console

Use the following steps to restore an object that has been archived to the S3 Glacier Flexible Retrieval or S3 Glacier Deep Archive storage classes, to check the status, and to upgrade an in-progress restore. (The console uses the names **S3 Glacier Flexible Retrieval** and **Glacier Deep Archive** for these storage classes.)

To restore an archived 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 **Buckets** list, choose the name of the bucket that contains the objects that you want to restore.
3. In the **Objects** list, select the object or objects that you want to restore, choose **Actions**, and then choose **Initiate restore**.
4. If you're restoring from S3 Glacier Flexible Retrieval or S3 Glacier Deep Archive, enter the number of days that you want your archived data to be accessible in the **Initiate restore** dialog box.
5. In **Retrieval options**, do one of the following:
 - Choose **Bulk retrieval** or **Standard retrieval**, and then choose **Restore**.
 - Choose **Expedited retrieval** (available only for S3 Glacier Flexible Retrieval or S3 Intelligent-Tiering Archive Access).

6. Provisioned capacity is only available for objects in S3 Glacier Flexible Retrieval. If you have provisioned capacity, choose **Restore** to start a provisioned retrieval.

If you have provisioned capacity, all of your expedited retrievals are served by your provisioned capacity. For more information, see [Provisioned capacity \(p. 672\)](#).

- If you don't have provisioned capacity and you don't want to buy it, choose **Restore**.
- If you don't have provisioned capacity, but you want to buy it, choose **Add capacity unit**, and then choose **Buy**. When you get the **Purchase succeeded** message, choose **Restore** to start provisioned retrieval.

You can upgrade the speed of your restoration while it is in progress.

To upgrade an in-progress restore to a faster tier

1. Open the Amazon S3 console at <https://console.aws.amazon.com/s3/>.
2. In the **Bucket name** list, choose the name of the bucket that contains the objects that you want to restore.
3. In the **Objects** list, select one or more of the objects that you are restoring, choose **Actions**, and then choose **Restore from S3 Glacier Flexible Retrieval**. For information about checking the restoration status of an object, see [Checking restore status and expiration date \(p. 674\)](#).
4. Choose the tier that you want to upgrade to, and then choose **Restore**.

For information about upgrading to a faster restore tier, see [Upgrading the speed of an in-progress restore \(p. 672\)](#).

Note

Standard and bulk restores for S3 Intelligent-Tiering are free of charge. However, subsequent restore requests called on an object that is already being restored are billed as a GET request.

[Checking restore status and expiration date](#)

You can check the progress of the restoration on the **Object overview** page. For more information, see [Viewing an object overview in the Amazon S3 console \(p. 256\)](#). This page will show that the restoration is **In progress**.

If you're restoring from S3 Glacier Flexible Retrieval or S3 Glacier Deep Archive, the temporary copy of the **Object overview** shows the **Restoration expiry date**. Amazon S3 will remove the restored copy of your archive on this date.

Restored objects from S3 Glacier Flexible Retrieval or S3 Glacier Deep Archive are stored only for the number of days that you specify. If you want a permanent copy of the object, create a copy of it in your Amazon S3 bucket.

After restoring an object, you can download it from the **Overview** page. For more information, see [Viewing an object overview in the Amazon S3 console \(p. 256\)](#).

Using the AWS SDKs

Java

The following example restores a copy of an object that has been archived using the AWS SDK for Java. The example initiates a restoration request for the specified archived object and checks its restoration status.

For instructions on creating and testing a working sample, see [Testing the Amazon S3 Java Code Examples \(p. 1192\)](#).

```
import com.amazonaws.AmazonServiceException;
import com.amazonaws.SdkClientException;
import com.amazonaws.auth.profile.ProfileCredentialsProvider;
import com.amazonaws.regions.Regions;
import com.amazonaws.services.s3.AmazonS3;
import com.amazonaws.services.s3.AmazonS3ClientBuilder;
import com.amazonaws.services.s3.model.ObjectMetadata;
import com.amazonaws.services.s3.model.RestoreObjectRequest;

import java.io.IOException;

public class RestoreArchivedObject {

    public static void main(String[] args) throws IOException {
        Regions clientRegion = Regions.DEFAULT_REGION;
        String bucketName = "*** Bucket name ***";
        String keyName = "*** Object key ***";

        try {
            AmazonS3 s3Client = AmazonS3ClientBuilder.standard()
                .withCredentials(new ProfileCredentialsProvider())
                .withRegion(clientRegion)
                .build();

            // Create and submit a request to restore an object from Glacier for two
            days.
            RestoreObjectRequest requestRestore = new RestoreObjectRequest(bucketName,
keyName, 2);
            s3Client.restoreObjectV2(requestRestore);

            // Check the restoration status of the object.
            ObjectMetadata response = s3Client.getObjectMetadata(bucketName, keyName);
            Boolean restoreFlag = response.getOngoingRestore();
            System.out.format("Restoration status: %s.\n",
                restoreFlag ? "in progress" : "not in progress (finished or
failed)");
        } catch (AmazonServiceException e) {
            // The call was transmitted successfully, but Amazon S3 couldn't process
            // it, so it returned an error response.
            e.printStackTrace();
        } catch (SdkClientException e) {
            // Amazon S3 couldn't be contacted for a response, or the client
            // couldn't parse the response from Amazon S3.
            e.printStackTrace();
        }
    }
}
```

.NET

The following C# example initiates a request to restore an archived object for 2 days. Amazon S3 maintains the restoration status in the object metadata. After initiating the request, the example retrieves the object metadata and checks the value of the `RestoreInProgress` property.

For instructions on creating and testing a working sample, see [Running the Amazon S3 .NET Code Examples \(p. 1193\)](#).

```
using Amazon;
using Amazon.S3;
using Amazon.S3.Model;
using System;
using System.Threading.Tasks;
```

```
namespace Amazon.DocSamples.S3
{
    class RestoreArchivedObjectTest
    {
        private const string bucketName = "*** bucket name ***";
        private const string objectKey = "** archived object key name **";
        // Specify your bucket region (an example region is shown).
        private static readonly RegionEndpoint bucketRegion = RegionEndpoint.USWest2;
        private static IAmazonS3 client;

        public static void Main()
        {
            client = new AmazonS3Client(bucketRegion);
            RestoreObjectAsync(client, bucketName, objectKey).Wait();
        }

        static async Task RestoreObjectAsync(IAmazonS3 client, string bucketName,
string objectKey)
        {
            try
            {
                var restoreRequest = new RestoreObjectRequest
                {
                    BucketName = bucketName,
                    Key = objectKey,
                    Days = 2
                };
                RestoreObjectResponse response = await
client.RestoreObjectAsync(restoreRequest);

                // Check the status of the restoration.
                await CheckRestorationStatusAsync(client, bucketName, objectKey);
            }
            catch (AmazonS3Exception amazonS3Exception)
            {
                Console.WriteLine("An AmazonS3Exception was thrown. Exception: " +
amazonS3Exception.ToString());
            }
            catch (Exception e)
            {
                Console.WriteLine("Exception: " + e.ToString());
            }
        }

        static async Task CheckRestorationStatusAsync(IAmazonS3 client, string
bucketName, string objectKey)
        {
            GetObjectMetadataRequest metadataRequest = new GetObjectMetadataRequest
            {
                BucketName = bucketName,
                Key = objectKey
            };
            GetObjectMetadataResponse response = await
client.GetObjectMetadataAsync(metadataRequest);
            Console.WriteLine("restoration status: {0}", response.RestoreInProgress ?
"in-progress" : "finished or failed");
        }
    }
}
```

Using the REST API

Amazon S3 provides an API for you to initiate an archive restoration. For more information, see [RestoreObject](#) in the *Amazon Simple Storage Service API Reference*.

Using the AWS CLI

Use the `restore-object` command to restore objects from S3 Glacier Flexible Retrieval.

The following example restores object `dir1/example.obj` in `awsexamplebucket` for 25 days.

```
aws s3api restore-object --bucket awsexamplebucket --key dir1/example.obj --restore-request '{"Days":25,"GlacierJobParameters":{"Tier":"Standard"}}'
```

If the JSON syntax used in the example results in an error on a Windows client, replace the restore request with the following syntax:

```
--restore-request Days=25,GlacierJobParameters={"Tier":"Standard"}
```

You can use the following command to monitor the status of your `restore-object` request:

```
aws s3api head-object --bucket awsexamplebucket --key dir1/example.obj
```

For more information, see [restore-object](#) in the AWS CLI Command Reference.

Querying archived objects

With the select type of [POST Object restore](#), you can perform filtering operations using simple Structured Query Language (SQL) statements directly on your data that is archived by Amazon S3 to S3 Glacier. When you provide an SQL query for an archived object, `select` runs the query in place and writes the output results to an S3 bucket. You can run queries and custom analytics on your data that is stored in S3 Glacier, without having to restore your entire object to Amazon S3.

When you perform select queries, S3 Glacier provides three data access tiers—*expedited*, *standard*, and *bulk*. All of these tiers provide different data access times and costs, and you can choose any one of them depending on how quickly you want your data to be available. For more information, see [Data access tiers \(p. 679\)](#).

You can use the select type of restore with the AWS SDKs, the S3 Glacier REST API, and the AWS Command Line Interface (AWS CLI).

Topics

- [Requirements and limits when using select \(p. 677\)](#)
- [Querying data using select \(p. 678\)](#)
- [Error handling \(p. 679\)](#)
- [Data access tiers \(p. 679\)](#)

Requirements and limits when using select

The following are requirements for using select:

- Archive objects that are queried by select must be formatted as uncompressed comma-separated values (CSV).

- You need an S3 bucket for output. The AWS account that you use to initiate an S3 Glacier select job must have write permissions for the S3 bucket. The bucket must be in the same AWS Region as the bucket that contains the archived object that is being queried.
- The requesting AWS account must have permissions to perform the `s3:RestoreObject` and `s3:GetObject` actions. For more information about these permissions, see [Example — Bucket subresource operations \(p. 416\)](#).
- The archive must not be encrypted with SSE-C or client-side encryption.

The following limits apply when using select:

- There are no limits on the number of records that select can process. An input or output record must not exceed 1 MB; otherwise, the query fails. There is a limit of 1,048,576 columns per record.
- There is no limit on the size of your final result. However, your results are broken into multiple parts.
- An SQL expression is limited to 128 KB.

Querying data using select

Using select, you can use SQL commands to query S3 Glacier archive objects that are in encrypted uncompressed CSV format. With this restriction, you can perform simple query operations on your text-based data in S3 Glacier. For example, you might look for a specific name or ID among a set of archived text files.

To query your S3 Glacier data, create a select request using the [POST Object restore](#) operation. When performing a select request, you provide the SQL expression, the archive to query, and the location to store the results.

The following example expression returns all records from the archived object specified in [POST Object restore](#).

```
SELECT * FROM object
```

S3 Glacier Select supports a subset of the ANSI SQL language. It supports common filtering SQL clauses like `SELECT`, `FROM`, and `WHERE`. It does not support `SUM`, `COUNT`, `GROUP BY`, `JOINS`, `DISTINCT`, `UNION`, `ORDER BY`, and `LIMIT`. For more information about support for SQL, see [SQL reference for Amazon S3 Select and S3 Glacier Select \(p. 856\)](#).

Select output

When you initiate a select request, you define an output location for the results of your select query. This location must be an S3 bucket in the same AWS Region as the bucket that contains the archived object that is being queried. The AWS account that initiates the job must have permissions to write to the bucket.

You can specify the Amazon S3 storage class and encryption for the output objects stored in Amazon S3. Select supports AWS Key Management Service (SSE-KMS) and Amazon S3 (SSE-S3) encryption. Select doesn't support SSE-C and client-side encryption. For more information about Amazon S3 storage classes and encryption, see [Using Amazon S3 storage classes \(p. 688\)](#) and [Protecting data using server-side encryption \(p. 338\)](#).

S3 Glacier Select results are stored in the S3 bucket using the prefix provided in the output location specified in [POST Object restore](#). From this information, select creates a unique prefix referring to the job ID. (Prefixes are used to group Amazon S3 objects together by beginning object names with a common string.) Under this unique prefix, there are two new prefixes created, `results` for results and `errors` for logs and errors. When the job is completed, a result manifest is written that contains the location of all results.

There is also a placeholder file named `job.txt` that is written to the output location. After it is written, it is never updated. The placeholder file is used for the following:

- Validation of the write permission and majority of SQL syntax errors synchronously.
- Providing a static output about your select request that you can easily reference whenever you want.

For example, suppose that you make a select request with the output location for the results specified as `s3://example-bucket/my-prefix`, and the job response returns the job ID as `examplekne1209ualkdjh812elkassdu9012e`. After the select job finishes, you can see the following Amazon S3 objects in your bucket.

```
s3://example-bucket/my-prefix/examplekne1209ualkdjh812elkassdu9012e/job.txt
s3://example-bucket/my-prefix/examplekne1209ualkdjh812elkassdu9012e/results/abc
s3://example-bucket/my-prefix/examplekne1209ualkdjh812elkassdu9012e/results/def
s3://example-bucket/my-prefix/examplekne1209ualkdjh812elkassdu9012e/results/ghi
s3://example-bucket/my-prefix/examplekne1209ualkdjh812elkassdu9012e/result_manifest.txt
```

The select query results are broken into multiple parts. In the example, select uses the prefix that you specified when setting the output location and appends the job ID and the `results` prefix. It then writes the results in three parts, with the object names ending in `abc`, `def`, and `ghi`. The result manifest contains all three files to allow programmatic retrieval. If the job fails with any error, then a file is visible under the error prefix and an `error_manifest.txt` is produced.

Presence of a `result_manifest.txt` file along with the absence of `error_manifest.txt` guarantees that the job finished successfully. There is no guarantee provided on how results are ordered.

Note

The length of an Amazon S3 object name, also referred to as the `key`, can be no more than 1,024 bytes. S3 Glacier Select reserves 128 bytes for prefixes. And, the length of your Amazon S3 location path cannot be more than 512 bytes. A request with a length greater than 512 bytes returns an exception, and the request is not accepted.

Error handling

Select notifies you of two kinds of errors. The first set of errors is sent to you synchronously when you submit the query in [POST Object restore](#). These errors are sent to you as part of the HTTP response. Another set of errors can occur after the query has been accepted successfully, but they happen during query execution. In this case, the errors are written to the specified output location under the `errors` prefix.

Select stops running the query after encountering an error. To run the query successfully, you must resolve all errors. You can check the logs to identify which records caused a failure.

Because queries run in parallel across multiple compute nodes, the errors that you get are not in sequential order. For example, if your query fails with an error in row 6,234, it does not mean that all rows before row 6,234 were successfully processed. The next run of the query might show an error in a different row.

Data access tiers

You can specify one of the following data access tiers when querying an archived object:

- **Expedited** – Allows you to quickly access your data when occasional urgent requests for a subset of archives are required. For all but the largest archived object (250 MB+), data accessed using Expedited retrievals are typically made available within 1–5 minutes. There are two types of Expedited data access: On-Demand and Provisioned. On-Demand requests are similar to EC2 On-Demand instances and are available most of the time. Provisioned requests are guaranteed to be available when you need them. For more information, see [Provisioned capacity \(p. 680\)](#).

- **Standard** – Allows you to access any of your archived objects within several hours. Standard retrievals typically finish within 3–5 hours. This is the default tier.
- **Bulk** – The lowest-cost data access option in S3 Glacier, enabling you to retrieve large amounts, even petabytes, of data inexpensively in a day. Bulk access typically finishes within 5–12 hours.

To make an Expedited, Standard, or Bulk request, set the `Tier` request element in the [POST Object restore](#) REST API request to the option you want, or the equivalent in the AWS CLI or AWS SDKs. For Expedited access, there is no need to designate whether an expedited retrieval is On-Demand or Provisioned. If you purchased provisioned capacity, all Expedited retrievals are automatically served through your provisioned capacity. For information about tier pricing, see [S3 Glacier pricing](#).

Provisioned capacity

Provisioned capacity helps ensure that your retrieval capacity for expedited retrievals is available when you need it. Each unit of capacity ensures that at least three expedited retrievals can be performed every 5 minutes and provides up to 150 MB/s of retrieval throughput. For more information, see [the section called "Provisioned capacity" \(p. 672\)](#).

Using S3 Object Lock

With S3 Object Lock, you can store objects using a *write-once-read-many* (WORM) model. Object Lock can help prevent objects from being deleted or overwritten for a fixed amount of time or indefinitely. You can use Object Lock to help meet regulatory requirements that require WORM storage, or to simply add another layer of protection against object changes and deletion.

S3 Object Lock has been assessed by Cohasset Associates for use in environments that are subject to SEC 17a-4, CFTC, and FINRA regulations. For more information about how Object Lock relates to these regulations, see the [Cohasset Associates Compliance Assessment](#).

Object Lock provides two ways to manage object retention: *retention periods* and *legal holds*.

- **Retention period** — Specifies a fixed period of time during which an object remains locked. During this period, your object is WORM-protected and can't be overwritten or deleted. For more information, see [Retention periods \(p. 682\)](#)
- **Legal hold** — Provides the same protection as a retention period, but it has no expiration date. Instead, a legal hold remains in place until you explicitly remove it. Legal holds are independent from retention periods. For more information, see [Legal holds \(p. 682\)](#).

An object version can have both a retention period and a legal hold, one but not the other, or neither. For more information, see [How S3 Object Lock works \(p. 681\)](#).

Object Lock works only in versioned buckets, and retention periods and legal holds apply to individual object versions. When you lock an object version, Amazon S3 stores the lock information in the metadata for that object version. Placing a retention period or legal hold on an object protects only the version specified in the request. It doesn't prevent new versions of the object from being created.

If you put an object into a bucket that has the same key name as an existing protected object, Amazon S3 creates a new version of that object, stores it in the bucket as requested, and reports the request as completed successfully. The existing protected version of the object remains locked according to its retention configuration.

To use S3 Object Lock, you follow these basic steps:

1. Create a new bucket with Object Lock enabled.

2. (Optional) Configure a default retention period for objects placed in the bucket.
3. Place the objects that you want to lock in the bucket.
4. Apply a retention period, a legal hold, or both, to the objects that you want to protect.

For information about configuring and managing S3 Object Lock, see the following sections:

Topics

- [How S3 Object Lock works \(p. 681\)](#)
- [Configuring S3 Object Lock using the console \(p. 684\)](#)
- [Managing Object Lock \(p. 685\)](#)

How S3 Object Lock works

You can use S3 Object Lock to store objects using a *write-once-read-many* (WORM) model. Object Lock can help prevent objects from being deleted or overwritten for a fixed amount of time or indefinitely. You can use S3 Object Lock to meet regulatory requirements that require WORM storage, or add an extra layer of protection against object changes and deletion.

For information about managing the lock status of your Amazon S3 objects, see [the section called "Managing Object Lock" \(p. 685\)](#).

Note

S3 buckets with S3 Object Lock can't be used as destination buckets for server access logs. For more information, see [the section called "Logging server access" \(p. 978\)](#).

The following sections describe the main features of S3 Object Lock.

Topics

- [Retention modes \(p. 681\)](#)
- [Retention periods \(p. 682\)](#)
- [Legal holds \(p. 682\)](#)
- [Bucket configuration \(p. 683\)](#)
- [Required permissions \(p. 684\)](#)

Retention modes

S3 Object Lock provides two *retention modes*:

- Governance mode
- Compliance mode

These retention modes apply different levels of protection to your objects. You can apply either retention mode to any object version that is protected by Object Lock.

In *governance mode*, users can't overwrite or delete an object version or alter its lock settings unless they have special permissions. With governance mode, you protect objects against being deleted by most users, but you can still grant some users permission to alter the retention settings or delete the object if necessary. You can also use governance mode to test retention-period settings before creating a compliance-mode retention period.

To override or remove governance-mode retention settings, a user must have the `s3:BypassGovernanceRetention` permission and must explicitly include `x-amz-bypass-`

`governance-retention:true` as a request header with any request that requires overriding governance mode.

Note

The Amazon S3 console by default includes the `x-amz-bypass-governance-retention:true` header. If you try to delete objects protected by *governance* mode and have `s3:BypassGovernanceRetention` permissions, the operation will succeed.

In *compliance* mode, a protected object version can't be overwritten or deleted by any user, including the root user in your AWS account. When an object is locked in compliance mode, its retention mode can't be changed, and its retention period can't be shortened. Compliance mode helps ensure that an object version can't be overwritten or deleted for the duration of the retention period.

Note

Updating an object version's metadata, as occurs when you place or alter an Object Lock, doesn't overwrite the object version or reset its `Last-Modified` timestamp.

Retention periods

A *retention period* protects an object version for a fixed amount of time. When you place a retention period on an object version, Amazon S3 stores a timestamp in the object version's metadata to indicate when the retention period expires. After the retention period expires, the object version can be overwritten or deleted unless you also placed a legal hold on the object version.

You can place a retention period on an object version either explicitly or through a bucket default setting. When you apply a retention period to an object version explicitly, you specify a *Retain Until Date* for the object version. Amazon S3 stores the `Retain Until Date` setting in the object version's metadata and protects the object version until the retention period expires.

When you use bucket default settings, you don't specify a `Retain Until Date`. Instead, you specify a duration, in either days or years, for which every object version placed in the bucket should be protected. When you place an object in the bucket, Amazon S3 calculates a `Retain Until Date` for the object version by adding the specified duration to the object version's creation timestamp. It stores the `Retain Until Date` in the object version's metadata. The object version is then protected exactly as though you explicitly placed a lock with that retention period on the object version.

Note

If your request to place an object version in a bucket contains an explicit retention mode and period, those settings override any bucket default settings for that object version.

Like all other Object Lock settings, retention periods apply to individual object versions. Different versions of a single object can have different retention modes and periods.

For example, suppose that you have an object that is 15 days into a 30-day retention period, and you `PUT` an object into Amazon S3 with the same name and a 60-day retention period. In this case, your `PUT` succeeds, and Amazon S3 creates a new version of the object with a 60-day retention period. The older version maintains its original retention period and becomes deletable in 15 days.

You can extend a retention period after you've applied a retention setting to an object version. To do this, submit a new lock request for the object version with a `Retain Until Date` that is later than the one currently configured for the object version. Amazon S3 replaces the existing retention period with the new, longer period. Any user with permissions to place an object retention period can extend a retention period for an object version locked in either mode.

Legal holds

With Object Lock you can also place a *legal hold* on an object version. Like a retention period, a legal hold prevents an object version from being overwritten or deleted. However, a legal hold doesn't have an associated retention period and remains in effect until removed. Legal holds can be freely placed and

removed by any user who has the `s3:PutObjectLegalHold` permission. For a complete list of Amazon S3 permissions, see [Actions, resources, and condition keys for Amazon S3 \(p. 431\)](#).

Legal holds are independent from retention periods. As long as the bucket that contains the object has Object Lock enabled, you can place and remove legal holds regardless of whether the specified object version has a retention period set. Placing a legal hold on an object version doesn't affect the retention mode or retention period for that object version.

For example, suppose that you place a legal hold on an object version while the object version is also protected by a retention period. If the retention period expires, the object doesn't lose its WORM protection. Rather, the legal hold continues to protect the object until an authorized user explicitly removes it. Similarly, if you remove a legal hold while an object version has a retention period in effect, the object version remains protected until the retention period expires.

To use Object Lock, you must enable it for a bucket. You can also optionally configure a default retention mode and period that applies to new objects that are placed in the bucket. For more information, see [Configuring S3 Object Lock using the console \(p. 684\)](#).

Bucket configuration

To use Object Lock, you must enable it for a bucket. You can also optionally configure a default retention mode and period that applies to new objects that are placed in the bucket.

Note

When using S3 Object Lock, make sure to take your encryption technique into consideration. For example, if you are using server-side encryption with AWS KMS keys, consider how the possible deletion of the key might interact with S3 Object Lock. It might be important to consider protection for the key as well.

Enabling S3 Object Lock

Before you can lock any objects, you have to configure a bucket to use S3 Object Lock. To do this, you specify when you create the bucket that you want to enable Object Lock. After you configure a bucket for Object Lock, you can lock objects in that bucket using retention periods, legal holds, or both.

Note

- You can only enable Object Lock for new buckets. If you want to turn on Object Lock for an existing bucket, contact AWS Support.
- When you create a bucket with Object Lock enabled, Amazon S3 automatically enables versioning for the bucket.
- If you create a bucket with Object Lock enabled, you can't disable Object Lock or suspend versioning for the bucket.

For information about enabling Object Lock on the console, see [Configuring S3 Object Lock using the console \(p. 684\)](#).

Default retention settings

When you turn on Object Lock for a bucket, the bucket can store protected objects. However, the setting doesn't automatically protect objects that you put into the bucket. If you want to automatically protect object versions that are placed in the bucket, you can configure a default retention period. Default settings apply to all new objects that are placed in the bucket, unless you explicitly specify a different retention mode and period for an object when you create it.

Tip

If you want to enforce the bucket default retention mode and period for all new object versions placed in a bucket, set the bucket defaults and deny users permission to configure object

retention settings. Amazon S3 then applies the default retention mode and period to new object versions placed in the bucket, and rejects any request to put an object that includes a retention mode and setting.

Bucket default settings require both a mode and a period. A bucket default mode is either *governance* or *compliance*. For more information, see [Retention modes \(p. 681\)](#).

A default retention period is described not as a timestamp, but as a period either in days or in years. When you place an object version in a bucket with a default retention period, Object Lock calculates a *Retain Until Date*. It does this by adding the default retention period to the creation timestamp for the object version. Amazon S3 stores the resulting timestamp as the object version's *Retain Until Date*, as if you had calculated the timestamp manually and placed it on the object version yourself.

Default settings apply only to new objects that are placed in the bucket. Placing a default retention setting on a bucket doesn't place any retention settings on objects that already exist in the bucket.

Important

Object locks apply to individual object versions only. If you place an object in a bucket that has a default retention period, and you don't explicitly specify a retention period for that object, Amazon S3 creates the object with a retention period that matches the bucket default. After the object is created, its retention period is independent from the bucket's default retention period. Changing a bucket's default retention period doesn't change the existing retention period for any objects in that bucket.

Note

If you configure a default retention period on a bucket, requests to upload objects in such a bucket must include the `Content-MD5` header. For more information, see [Put Object](#) in the *Amazon Simple Storage Service API Reference*.

Required permissions

Object Lock operations require specific permissions. Depending on the exact operation you are attempting, you might need any of the following permissions:

- `s3:BypassGovernanceRetention`
- `s3:GetBucketObjectLockConfiguration`
- `s3:GetObjectLegalHold`
- `s3:GetObjectRetention`
- `s3:PutBucketObjectLockConfiguration`
- `s3:PutObjectLegalHold`
- `s3:PutObjectRetention`

For information about using conditions with permissions, see [Amazon S3 condition key examples \(p. 420\)](#).

Configuring S3 Object Lock using the console

With S3 Object Lock, you can store objects in Amazon S3 using a *write-once-read-many* (WORM) model. You can use S3 Object Lock to prevent an object from being deleted or overwritten for a fixed amount of time or indefinitely. For more information about S3 Object Lock capabilities, see [How S3 Object Lock works \(p. 681\)](#).

To use S3 Object Lock, follow these basic steps:

1. Create a new bucket with Object Lock enabled.
2. (Optional) Configure a default retention period for objects placed in the bucket.

3. Place the objects that you want to lock in the bucket.
4. Apply a retention period, a legal hold, or both, to the objects that you want to protect.

Before you lock any objects, you have to enable a bucket to use S3 Object Lock. You enable Object Lock when you create a bucket. After you enable Object Lock on a bucket, you can lock objects in that bucket. When you create a bucket with Object Lock enabled, you can't disable Object Lock or suspend versioning for that bucket.

For information about creating a bucket with S3 Object Lock enabled, see [Creating a bucket \(p. 119\)](#).

To enable Object Lock legal hold

1. Sign in to the AWS Management Console and open the Amazon S3 console at <https://console.aws.amazon.com/s3/>.
2. In the **Buckets** list, choose the name of the bucket that you want.
3. In the **Objects** list, choose the name of the object that you want to enable or disable legal hold for.
The **Object overview** opens, displaying the properties for your object.
4. Under **Object Lock legal hold**, choose **Edit**.
5. Under **Legal hold**, choose **Enable** or **Disable**.
6. Choose **Save changes**.

To edit Object Lock retention settings

1. In the **Objects** list, choose the name of the object that you want to edit Object Lock retention settings for.
The **Object overview** opens, displaying the properties for your object.
2. Under **Object Lock retention**, choose **Edit**.
3. Under **Retention**, choose **Enable** or **Disable**.
4. Under **Retention mode**, choose **Governance mode** or **Compliance mode**.
5. In the **Retain until date** box, enter the date when the object is no longer protected by the chosen retention mode.
6. Choose **Save changes**.

For more information about legal hold and retention settings, see [How S3 Object Lock works \(p. 681\)](#).

For information about managing Object Lock using the AWS CLI, AWS SDKs, and the Amazon S3 REST APIs, see [\(p. 685\)](#).

Managing Object Lock

You can use the AWS CLI, AWS SDKs, and the Amazon S3 REST APIs to configure and view lock information, set retention limits, manage deletes and lifecycles, and more.

Topics

- [Viewing the lock information for an object \(p. 686\)](#)
- [Bypassing governance mode \(p. 686\)](#)
- [Configuring events and notifications \(p. 686\)](#)
- [Setting retention limits \(p. 686\)](#)
- [Managing delete markers and object lifecycles \(p. 687\)](#)

- [Using S3 Object Lock with replication \(p. 687\)](#)

Viewing the lock information for an object

You can view the Object Lock status of an Amazon S3 object version using the `GET Object` or `HEAD Object` commands. Both commands return the retention mode, `Retain Until Date`, and the legal hold status for the specified object version.

To view an object version's retention mode and retention period, you must have the `s3:GetObjectRetention` permission. To view an object version's legal hold status, you must have the `s3:GetObjectLegalHold` permission. If you `GET` or `HEAD` an object version but don't have the necessary permissions to view its lock status, the request succeeds. However, it doesn't return information that you don't have permission to view.

To view a bucket's default retention configuration (if it has one), request the bucket's Object Lock configuration. To do this, you must have the `s3:GetBucketObjectLockConfiguration` permission. If you make a request for an Object Lock configuration against a bucket that doesn't have S3 Object Lock enabled, Amazon S3 returns an error. For more information about permissions, see [Example — Object operations \(p. 415\)](#).

You can configure Amazon S3 Inventory reports on your buckets to include the `Retain Until Date`, `object lock Mode`, and `Legal Hold Status` for all objects in a bucket. For more information, see [Amazon S3 Inventory \(p. 739\)](#).

Bypassing governance mode

You can perform operations on object versions that are locked in governance mode as if they were unprotected if you have the `s3:BypassGovernanceRetention` permission. These operations include deleting an object version, shortening the retention period, or removing the object lock by placing a new lock with empty parameters.

To bypass governance mode, you must explicitly indicate in your request that you want to bypass this mode. To do this, include the `x-amz-bypass-governance-retention:true` header with your request, or use the equivalent parameter with requests made through the AWS CLI, or AWS SDKs. The AWS Management Console automatically applies this header for requests made through the console if you have the permission required to bypass governance mode.

Note

Bypassing governance mode doesn't affect an object version's legal hold status. If an object version has a legal hold enabled, the legal hold remains in force and prevents requests to overwrite or delete the object version.

Configuring events and notifications

You can use Amazon S3 Event Notifications to track access and changes to your Object Lock configurations and data using AWS CloudTrail. For information about CloudTrail, see the [AWS CloudTrail documentation](#).

You can also use Amazon CloudWatch to generate alerts based on this data. For information about CloudWatch, see the [Amazon CloudWatch documentation](#).

Setting retention limits

You can set minimum and maximum allowable retention periods for a bucket using a bucket policy. You do this using the `s3:object-lock-remaining-retention-days` condition key. The maximum retention period is 100 years.

The following example shows a bucket policy that uses the `s3:object-lock-remaining-retention-days` condition key to set a maximum retention period of 10 days.

```
{  
    "Version": "2012-10-17",  
    "Id": "<SetRetentionLimits",  
    "Statement": [  
        {  
            "Sid": "<SetRetentionPeriod",  
            "Effect": "Deny",  
            "Principal": "*",  
            "Action": [  
                "s3:PutObjectRetention"  
            ],  
            "Resource": "arn:aws:s3:::<awsexamplebucket1>/*",  
            "Condition": {  
                "NumericGreaterThan": {  
                    "s3:object-lock-remaining-retention-days": "10"  
                }  
            }  
        }  
    ]  
}
```

Note

If your bucket is the destination bucket for a replication policy and you want to set up minimum and maximum allowable retention periods for object replicas that are created using replication, you must include the `s3:ReplicateObject` action in your bucket policy.

For more information, see the following topics:

- [Actions, resources, and condition keys for Amazon S3 \(p. 431\)](#)
- [Example — Object operations \(p. 415\)](#)
- [Amazon S3 condition key examples \(p. 420\)](#)

Managing delete markers and object lifecycles

Although you can't delete a protected object version, you can still create a delete marker for that object. Placing a delete marker on an object doesn't delete the object or its object versions. However, it makes Amazon S3 behave in most ways as though the object has been deleted. For more information, see [Working with delete markers \(p. 662\)](#).

Note

Delete markers are not WORM-protected, regardless of any retention period or legal hold in place on the underlying object.

Object lifecycle management configurations continue to function normally on protected objects, including placing delete markers. However, protected object versions remain safe from being deleted or overwritten by a lifecycle configuration. For more information about managing object lifecycles, see [Managing your storage lifecycle \(p. 701\)](#).

Using S3 Object Lock with replication

You can use S3 Object Lock with replication to enable automatic, asynchronous copying of locked objects and their retention metadata, across S3 buckets in different or the same AWS Regions. When you use replication, objects in a *source bucket* are replicated to a *destination bucket*. For more information, see [Replicating objects \(p. 753\)](#).

To set up S3 Object Lock with replication, you can choose one of the following options.

Option 1: Enable Object Lock first

1. Enable Object Lock on the destination bucket, or on both the source and the destination bucket.
2. Set up replication between the source and the destination buckets.

Option 2: Set up replication first

1. Set up replication between the source and destination buckets.
2. Enable Object Lock on just the destination bucket, or on both the source and destination buckets.

When enabling Object Lock in the preceding options, this must either be done at the time of bucket creation or you must contact AWS Support if using an existing bucket. This is required to make sure that replication is configured correctly.

Before you contact AWS Support, review the following requirements for setting up Object Lock with replication:

- The Amazon S3 destination bucket must have Object Lock enabled on it.
- You must grant two new permissions on the source S3 bucket in the AWS Identity and Access Management (IAM) role that you use to set up replication. The two new permissions are `s3:GetObjectRetention` and `s3:GetObjectLegalHold`. If the role has an `s3:Get*` permission, it satisfies the requirement. For more information, see [Setting up permissions \(p. 769\)](#).

For more information about S3 Object Lock, see [How S3 Object Lock works \(p. 681\)](#).

Using Amazon S3 storage classes

Each object in Amazon S3 has a storage class associated with it. For example, if you list the objects in an S3 bucket, the console shows the storage class for all the objects in the list. Amazon S3 offers a range of storage classes for the objects that you store. You choose a class depending on your use case scenario and performance access requirements. All of these storage classes offer high durability.

The following sections provide details of the various storage classes and how to set the storage class for your objects.

Topics

- [Storage classes for frequently accessed objects \(p. 688\)](#)
- [Storage class for automatically optimizing data with changing or unknown access patterns \(p. 689\)](#)
- [Storage classes for infrequently accessed objects \(p. 689\)](#)
- [Storage classes for archiving objects \(p. 690\)](#)
- [Storage class for Amazon S3 on Outposts \(p. 691\)](#)
- [Comparing the Amazon S3 storage classes \(p. 692\)](#)
- [Setting the storage class of an object \(p. 692\)](#)

Storage classes for frequently accessed objects

For performance-sensitive use cases (those that require millisecond access time) and frequently accessed data, Amazon S3 provides the following storage classes:

- **S3 Standard** — The default storage class. If you don't specify the storage class when you upload an object, Amazon S3 assigns the S3 Standard storage class.
- **Reduced Redundancy** — The Reduced Redundancy Storage (RRS) storage class is designed for noncritical, reproducible data that can be stored with less redundancy than the S3 Standard storage class.

Important

We recommend that you not use this storage class. The S3 Standard storage class is more cost effective.

For durability, RRS objects have an average annual expected loss of 0.01 percent of objects. If an RRS object is lost, when requests are made to that object, Amazon S3 returns a 405 error.

Storage class for automatically optimizing data with changing or unknown access patterns

S3 Intelligent-Tiering is an Amazon S3 storage class designed to optimize storage costs by automatically moving data to the most cost-effective access tier, without performance impact or operational overhead. It is the only cloud storage that delivers automatic cost savings by moving data on a granular object level between access tiers when access patterns change. S3 Intelligent-Tiering is the perfect storage class when you want to optimize storage costs for data that has unknown or changing access patterns. There are no retrieval fees for S3 Intelligent-Tiering.

For a small monthly object monitoring and automation fee, S3 Intelligent-Tiering monitors access patterns and automatically moves objects that have not been accessed to lower cost access tiers. S3 Intelligent-Tiering delivers automatic storage cost savings in two low latency and high throughput access tiers. For data that can be accessed asynchronously, customers can choose to activate automatic archiving capabilities within the S3 Intelligent-Tiering storage class. S3 Intelligent-Tiering is designed for 99.9% availability and 99.999999999% durability.

S3 Intelligent-Tiering automatically stores objects in three access tiers: a *Frequent Access* tier, an *Infrequent Access* tier, and a *Archive Instant Access* tier. Objects that are uploaded or transitioned to S3 Intelligent-Tiering are automatically stored in the *Frequent Access* tier. S3 Intelligent-Tiering works by monitoring access patterns and then moving the objects that have not been accessed in 30 consecutive days to the *Infrequent Access* tier. With S3 Intelligent-Tiering, any existing objects that have not been accessed for 90 consecutive days will automatically move to the *Archive Instant Access* tier. You can configure S3 Intelligent-Tiering as your default storage class for newly created data, or you can choose to activate one or both of the archive access tiers using the API with [PutBucketIntelligentTieringConfiguration](#), the CLI, or the Amazon S3 console. After you activate one or both of the archive access tiers, S3 Intelligent-Tiering automatically moves objects that haven't been accessed for 90 consecutive days to the *Archive Access* tier, and after 180 consecutive days of no access, to the *Deep Archive Access* tier. For information on using S3 Intelligent-Tiering, see [Using S3 Intelligent-Tiering \(p. 695\)](#)

In order to access archived objects later, you first need to restore them. For more information, see [Restoring objects from the S3 Intelligent-Tiering Archive Access and Deep Archive Access tiers \(p. 700\)](#).

Note

If the size of an object is less than 128 KB, it is not monitored and not eligible for auto-tiering. Smaller objects are always stored in the Frequent Access tier. For information on S3 Intelligent-Tiering, see [S3 Intelligent-Tiering access tiers \(p. 694\)](#)

Storage classes for infrequently accessed objects

The **S3 Standard-IA** and **S3 One Zone-IA** storage classes are designed for long-lived and infrequently accessed data. (IA stands for *infrequent access*.) S3 Standard-IA and S3 One Zone-IA objects are available

for millisecond access (similar to the S3 Standard storage class). Amazon S3 charges a retrieval fee for these objects, so they are most suitable for infrequently accessed data. For pricing information, see [Amazon S3 pricing](#).

For example, you might choose the S3 Standard-IA and S3 One Zone-IA storage classes to do the following:

- For storing backups.
- For older data that is accessed infrequently, but that still requires millisecond access. For example, when you upload data, you might choose the S3 Standard storage class, and use lifecycle configuration to tell Amazon S3 to transition the objects to the S3 Standard-IA or S3 One Zone-IA class.

For more information about lifecycle management, see [Managing your storage lifecycle \(p. 701\)](#).

Note

The S3 Standard-IA and S3 One Zone-IA storage classes are suitable for objects larger than 128 KB that you plan to store for at least 30 days. If an object is less than 128 KB, Amazon S3 charges you for 128 KB. If you delete an object before the end of the 30-day minimum storage duration period, you are charged for 30 days. For pricing information, see [Amazon S3 pricing](#).

These storage classes differ as follows:

- **S3 Standard-IA** — Amazon S3 stores the object data redundantly across multiple geographically separated Availability Zones (similar to the S3 Standard storage class). S3 Standard-IA objects are resilient to the loss of an Availability Zone. This storage class offers greater availability and resiliency than the S3 One Zone-IA class.
- **S3 One Zone-IA** — Amazon S3 stores the object data in only one Availability Zone, which makes it less expensive than S3 Standard-IA. However, the data is not resilient to the physical loss of the Availability Zone resulting from disasters, such as earthquakes and floods. The S3 One Zone-IA storage class is as durable as Standard-IA, but it is less available and less resilient. For a comparison of storage class durability and availability, see [Comparing the Amazon S3 storage classes \(p. 692\)](#) at the end of this section. For pricing information, see [Amazon S3 pricing](#).

We recommend the following:

- S3 Standard-IA — Use for your primary or only copy of data that can't be re-created.
- S3 One Zone-IA — Use if you can re-create the data if the Availability Zone fails, and for object replicas when setting S3 Cross-Region Replication (CRR).

Storage classes for archiving objects

The **S3 Glacier Instant Retrieval**, **S3 Glacier Flexible Retrieval**, and **S3 Glacier Deep Archive** storage classes are designed for low-cost data archiving. These storage classes offer the same durability and resiliency as the S3 Standard and S3 Standard-IA storage classes. For a comparison of storage class durability and availability, see [Comparing the Amazon S3 storage classes \(p. 692\)](#).

These storage classes differ as follows:

- **S3 Glacier Instant Retrieval** — Use for archiving data that is rarely accessed and requires milliseconds retrieval. Data stored in the S3 Glacier Instant Retrieval storage class offers a cost savings compared to the S3 Standard-IA storage class, with the same latency and throughput performance as the S3 Standard-IA storage class. S3 Glacier Instant Retrieval has higher data access costs than S3 Standard-IA. For pricing information, see [Amazon S3 pricing](#).
- **S3 Glacier Flexible Retrieval** — Use for archives where portions of the data might need to be retrieved in minutes. Data stored in the S3 Glacier Flexible Retrieval storage class has a minimum storage

duration period of 90 days and can be accessed in as little as 1–5 minutes using expedited retrieval. The retrieval time is flexible, and you can request free bulk retrievals in up to 5–12 hours. If you have deleted, overwritten, or transitioned to a different storage class an object before the 90-day minimum, you are charged for 90 days. For pricing information, see [Amazon S3 pricing](#).

- **S3 Glacier Deep Archive** — Use for archiving data that rarely needs to be accessed. Data stored in the S3 Glacier Deep Archive storage class has a minimum storage duration period of 180 days and a default retrieval time of 12 hours. If you have deleted, overwritten, or transitioned to a different storage class an object before the 180-day minimum, you are charged for 180 days. For pricing information, see [Amazon S3 pricing](#).

S3 Glacier Deep Archive is the lowest cost storage option in AWS. Storage costs for S3 Glacier Deep Archive are less expensive than using the S3 Glacier Flexible Retrieval storage class. You can reduce S3 Glacier Deep Archive retrieval costs by using bulk retrieval, which returns data within 48 hours.

Retrieving archived objects

You can set the storage class of an object to S3 Glacier Flexible Retrieval or S3 Glacier Deep Archive in the same ways that you do for the other storage classes as described in the section [Setting the storage class of an object \(p. 692\)](#). However, the S3 Glacier Flexible Retrieval and S3 Glacier Deep Archive objects are not available for real-time access. You must first restore the S3 Glacier Flexible Retrieval and S3 Glacier Deep Archive objects before you can access them. (S3 Standard, RRS, S3 Standard-IA, S3 One Zone-IA, S3 Glacier Instant Retrieval, and S3 Intelligent-Tiering objects are available for anytime access.) For more information about retrieving archived objects, see [Restoring an archived object \(p. 673\)](#).

Important

When you choose the S3 Glacier Flexible Retrieval or S3 Glacier Deep Archive storage class, your objects remain in Amazon S3. You can't access them directly through the separate Amazon S3 Glacier service.

To learn more about the Amazon S3 Glacier service, see the [Amazon S3 Glacier Developer Guide](#).

Storage class for Amazon S3 on Outposts

With Amazon S3 on Outposts, you can create S3 buckets on your AWS Outposts resources and store and retrieve objects on-premises for applications that require local data access, local data processing, and data residency. You can use the same APIs and features on AWS Outposts as you do on Amazon S3, including access policies, encryption, and tagging. You can use S3 on Outposts through the AWS Management Console, AWS CLI, AWS SDKs, or REST API.

S3 on Outposts provides a new storage class, S3 Outposts (OUTPOSTS). The S3 Outposts storage class is available only for objects stored in buckets on Outposts. If you try to use this storage class with an S3 bucket in an AWS Region, an `InvalidStorageClass` error occurs. In addition, if you try to use other S3 storage classes with objects stored in S3 on Outposts buckets, the same error occurs.

Objects stored in the S3 Outposts (OUTPOSTS) storage class are always encrypted by using server-side encryption with Amazon S3 managed encryption keys (SSE-S3). For more information, see [Protecting data using server-side encryption with Amazon S3-managed encryption keys \(SSE-S3\) \(p. 355\)](#).

You can also explicitly choose to encrypt objects stored in the S3 Outposts storage class by using server-side encryption with customer-provided encryption keys (SSE-C). For more information, see [Protecting data using server-side encryption with customer-provided encryption keys \(SSE-C\) \(p. 366\)](#).

Note

S3 on Outposts doesn't support server-side encryption with AWS Key Management Service (AWS KMS) keys (SSE-KMS).

For more information about S3 on Outposts, see [What is Amazon S3 on Outposts? \(p. 1241\)](#)

Comparing the Amazon S3 storage classes

The following table compares the storage classes, including their availability, durability, minimum storage duration, and other considerations.

Storage Class	Designed for	Durability (designed for)	Availability (designed for)	Availability Zones	Min storage duration	Min billable object size	Other Considerations
STANDARD	Frequently accessed data	99.999999999%	99.99%	>= 3	None	None	None
STANDARD_IA	Long-lived, infrequently accessed data	99.999999999%	99.9%	>= 3	30 days	128 KB	Per GB retrieval fees apply.
INTELLIGENT_TIERING	Long-lived data with changing or unknown access patterns	99.999999999%	99.9%	>= 3	30 days	None	Monitoring and automation fees per object apply. No retrieval fees.
ONEZONE_IA	Long-lived, infrequently accessed, non-critical data	99.999999999%	99.5%	1	30 days	128 KB	Per GB retrieval fees apply. Not resilient to the loss of the Availability Zone.
GLACIER	Long-term data archiving with retrieval times ranging from minutes to hours	99.999999999%	99.99% (after you restore objects)	>= 3	90 days	None	Per GB retrieval fees apply. You must first restore archived objects before you can access them. For more information, see Restoring Archived Objects .
DEEP_ARCHIVE	Archiving rarely accessed data with a default retrieval time of 12 hours	99.999999999%	99.99% (after you restore objects)	>= 3	180 days	None	Per GB retrieval fees apply. You must first restore archived objects before you can access them. For more information, see Restoring Archived Objects .
RRS (Not recommended)	Frequently accessed, non-critical data	99.99%	99.99%	>= 3	None	None	None

All of the storage classes except for S3 One Zone-IA are designed to be resilient to the physical loss of an Availability Zone resulting from disasters.

In addition to the performance requirements of your application scenario, consider costs. For storage class pricing, see [Amazon S3 pricing](#).

Setting the storage class of an object

To set and update object storage classes, you can use the Amazon S3 console, AWS SDKs, or the AWS Command Line Interface (AWS CLI). Each uses the Amazon S3 APIs to send requests to Amazon S3.

Amazon S3 APIs support setting (or updating) the storage class of objects as follows:

- When creating a new object, you can specify its storage class. For example, when creating objects using the [PUT Object](#), [POST Object](#), and [Initiate Multipart Upload](#) APIs, you add the `x-amz-storage-class` request header to specify a storage class. If you don't add this header, Amazon S3 uses Standard, the default storage class.
- You can also change the storage class of an object that is already stored in Amazon S3 to any other storage class by making a copy of the object using the [PUT Object - Copy](#) API. However, you can't use [PUT Object - Copy](#) to copy objects that are stored in the S3 Glacier Flexible Retrieval or S3 Glacier Deep Archive storage classes. You also can't transition from S3 One Zone-IA to S3 Glacier Instant Retrieval.

You copy the object in the same bucket using the same key name and specify request headers as follows:

- Set the `x-amz-metadata-directive` header to COPY.
- Set the `x-amz-storage-class` to the storage class that you want to use.

In a versioning-enabled bucket, you can't change the storage class of a specific version of an object. When you copy it, Amazon S3 gives it a new version ID.

- You can direct Amazon S3 to change the storage class of objects by adding an S3 Lifecycle configuration to a bucket. For more information, see [Managing your storage lifecycle \(p. 701\)](#).
- When setting up a replication configuration, you can set the storage class for replicated objects to any other storage class. However, you can't replicate objects that are stored in the S3 Glacier

Flexible Retrieval or S3 Glacier Deep Archive storage classes. For more information, see [Replication configuration \(p. 759\)](#).

Restricting access policy permissions to a specific storage class

When you grant access policy permissions for Amazon S3 operations, you can use the `s3:x-amz-storage-class` condition key to restrict which storage class to use when storing uploaded objects. For example, when you grant `s3:PUTObject` permission, you can restrict object uploads to a specific storage class. For an example policy, see [Example 5: Restricting object uploads to objects with a specific storage class \(p. 425\)](#).

For more information about using conditions in policies and a complete list of Amazon S3 condition keys, see the following:

- [Actions, resources, and condition keys for Amazon S3 \(p. 431\)](#)
- [Amazon S3 condition key examples \(p. 420\)](#)

Amazon S3 Intelligent-Tiering

The S3 Intelligent-Tiering storage class is designed to optimize storage costs by automatically moving data to the most cost-effective access tier when access patterns change, without operational overhead or impact on performance. For a small monthly object monitoring and automation charge, S3 Intelligent-Tiering monitors access patterns and automatically moves objects that have not been accessed to lower-cost access tiers.

S3 Intelligent-Tiering delivers automatic storage cost savings in three low latency and high throughput access tiers. For data that can be accessed asynchronously, you can choose to activate automatic archiving capabilities within the S3 Intelligent-Tiering storage class. There are no retrieval charges in S3 Intelligent-Tiering. If an object in the Infrequent Access tier or Archive Instant Access tier is accessed later, it is automatically moved back to the Frequent Access tier. No additional tiering charges apply when objects are moved between access tiers within the S3 Intelligent-Tiering storage class.

S3 Intelligent-Tiering is the recommended storage class for data with unknown, changing, or unpredictable access patterns, independent of object size or retention period, such as data lakes, data analytics, and new applications.

For information about using S3 Intelligent-Tiering, see the following sections:

Topics

- [How S3 Intelligent-Tiering works \(p. 693\)](#)
- [Using S3 Intelligent-Tiering \(p. 695\)](#)
- [Managing S3 Intelligent-Tiering \(p. 698\)](#)

How S3 Intelligent-Tiering works

The S3 Intelligent-Tiering storage class automatically stores objects in three access tiers. One tier is optimized for frequent access, one lower-cost tier is optimized for infrequent access, and another very low-cost tier is optimized for rarely accessed data. For a low monthly object monitoring and automation charge, S3 Intelligent-Tiering monitors access patterns and automatically moves objects to the Infrequent Access tier when they have not been accessed for 30 consecutive days. After 90 days of no access, the objects are moved to the Archive Instant Access tier without performance impact or operational overhead.

To get the lowest storage cost on data that can be accessed in minutes to hours, you can choose to activate additional archiving capabilities. After you activate the optional archive capabilities, S3 Intelligent-Tiering moves objects that have not been accessed for 90 consecutive days to the Archive Access tier. After 180 consecutive days of no access, the objects are moved to the Deep Archive Access tier.

There are no retrieval charges in S3 Intelligent-Tiering. If an object in the Infrequent Access tier or Archive Instant Access tier is accessed later, it is automatically moved back to the Frequent Access tier. Downloading or copying an object through the AWS Management Console or running [GetObject](#) or [CopyObject](#) will initiate access. Running a HEAD, LIST, GET object tags, or PUT object tags operation will not constitute access.

You can configure S3 Intelligent-Tiering as your default storage class for newly created data by specifying INTELLIGENT-TIERING in your [S3 PUT API](#) request header. S3 Intelligent-Tiering is designed for 99.9% availability and 99.999999999% durability.

Note

If the size of an object is less than 128 KB, it is not monitored and is not eligible for automatic tiering. Smaller objects are always stored in the Frequent Access tier.

S3 Intelligent-Tiering access tiers

Frequent Access tier (automatic)

This is the default access tier that any object created or transitioned to S3 Intelligent-Tiering begins its lifecycle in. An object remains in this tier as long as it is being accessed. The Frequent Access tier provides low latency and high throughput performance.

Infrequent Access tier (automatic)

If an object is not accessed for 30 consecutive days, the object moves to the Infrequent Access tier. The Infrequent Access tier provides low latency and high throughput performance.

Archive Instant Access tier (automatic)

If an object is not accessed for 90 consecutive days, the object moves to the Archive Instant Access tier. The Archive Instant Access tier provides low latency and high throughput performance.

Archive Access tier (optional)

S3 Intelligent-Tiering provides you with the option to activate the Archive Access tier for data that can be accessed asynchronously. After activation, the Archive Access tier automatically archives objects that have not been accessed for a minimum of 90 consecutive days. You can extend the last access time for archiving to a maximum of 730 days. The Archive Access tier has the same performance as the [S3 Glacier Flexible Retrieval](#) storage class. Standard retrieval times for this access tier can range from 3-5 hours.

Note

Only activate the Archive Access tier for 90 days if you want to bypass the Archive Instant Access tier. The Archive Access tier delivers slightly lower storage cost with minute to hour retrieval times. The Archive Instant Access tier delivers millisecond access and high throughput performance.

Deep Archive Access tier (optional)

S3 Intelligent-Tiering provides you with the option to activate the Deep Archive Access tier for data that can be accessed asynchronously. After activation, the Deep Archive Access tier automatically archives objects that have not been accessed for a minimum of 180 consecutive days. You can extend the last access time for archiving to a maximum of 730 days. The Deep Archive Access tier has the same performance as the [S3 Glacier Deep Archive](#) storage class. Standard retrieval of objects in this access tier occurs within 12 hours.

Note

Activate the Archive Access and Deep Archive Access tiers only if your objects can be accessed asynchronously by your application. If the object that you are retrieving is stored in the Archive Access or Deep Archive Access tiers, first restore the object using [RestoreObject](#). For more information, see [Restoring objects from the S3 Intelligent-Tiering Archive Access and Deep Archive Access tiers \(p. 700\)](#)

Using S3 Intelligent-Tiering

You can use the S3 Intelligent-Tiering storage class to automatically optimize storage costs. S3 Intelligent-Tiering delivers automatic cost savings by moving data on a granular object level between access tiers when access patterns change. For data that can be accessed asynchronously, you can choose to enable automatic archiving within the S3 Intelligent-Tiering storage class using the AWS Management Console, AWS CLI, or Amazon S3 API.

Moving data to S3 Intelligent-Tiering

There are two ways to move data into S3 Intelligent-Tiering. You can directly [PUT](#) data into S3 Intelligent-Tiering by specifying `INTELLIGENT_TIERING` in the `x-amz-storage-class` header or configure S3 Lifecycle policies to transition objects from S3 Standard or S3 Standard-Infrequent Access to S3 Intelligent-Tiering.

Uploading data to S3 Intelligent-Tiering using Direct PUT

When you upload an object to the S3 Intelligent-Tiering storage class using the [PUT](#) API operation, you specify S3 Intelligent-Tiering in the `x-amz-storage-class` request header.

The following request stores the image, `my-image.jpg`, in the `myBucket` bucket. The request uses the `x-amz-storage-class` header to request that the object is stored using the S3 Intelligent-Tiering storage class.

Example

```
PUT /my-image.jpg HTTP/1.1
Host: myBucket.s3.<Region>.amazonaws.com (http://amazonaws.com/)
Date: Wed, 1 Sep 2021 17:50:00 GMT
Authorization: authorization string
Content-Type: image/jpeg
Content-Length: 11434
Expect: 100-continue
x-amz-storage-class: INTELLIGENT_TIERING
```

Transitioning data to S3 Intelligent-Tiering from S3 Standard or S3 Standard-Infrequent Access using S3 Lifecycle

You can add rules to an S3 Lifecycle configuration to tell Amazon S3 to transition objects from one storage class to another. For information on supported transitions and related constraints, see [Transitioning objects using S3 Lifecycle](#).

You can specify S3 Lifecycle policies at the bucket or prefix level. In this S3 Lifecycle configuration rule, the filter specifies a key prefix (`documents/`). Therefore, the rule applies to objects with key name `prefix documents/`, such as `documents/doc1.txt` and `documents/doc2.txt`. The rule specifies a Transition action directing Amazon S3 to transition objects to the S3 Intelligent-Tiering storage class 0 days after creation. In this case, objects are eligible for transition to S3 Intelligent-Tiering at midnight UTC following creation.

Example

```
<LifecycleConfiguration>
  <Rule>
    <ID>ExampleRule</ID>
    <Filter>
      <Prefix>documents/</Prefix>
    </Filter>
    <Status>Enabled</Status>
    <Transition>
      <Days>0</Days>
      <StorageClass>INTELLIGENT_TIERING</StorageClass>
    </Transition>
  </Rule>
</LifecycleConfiguration>
```

Enabling S3 Intelligent-Tiering Archive Access and Deep Archive Access tiers

To get the lowest storage cost on data that can be accessed in minutes to hours, you can activate one or both of the archive access tiers by creating a bucket, prefix, or object tag level configuration using the AWS Management Console, AWS CLI, or Amazon S3 API.

Using the S3 console

To enable S3 Intelligent-Tiering automatic archiving

1. Sign in to the AWS Management Console and open the Amazon S3 console at <https://console.aws.amazon.com/s3/>.
2. In the **Buckets** list, choose the name of the bucket that you want.
3. Choose **Properties**.
4. Navigate to the **S3 Intelligent-Tiering Archive configurations** section and choose **Create configuration**.
5. In the **Archive configuration settings** section, specify a descriptive configuration name for your S3 Intelligent-Tiering Archive configuration.
6. Under **Choose a configuration scope**, choose a configuration scope to use. Optionally, you can limit the configuration scope to specified objects within a bucket using a shared prefix, object tag, or combination of the two.
 - a. To limit the scope of the configuration, select **Limit the scope of this configuration using one or more filters**.
 - b. To limit the scope of the configuration using a single prefix, enter the prefix under **Prefix**.
 - c. To limit the scope of the configuration using object tags, select **Add tag** and enter a value for **Key**.
7. Under **Status**, select **Enable**.
8. In the **Archive settings** section, select one or both of the Archive Access tiers to enable.
9. Choose **Create**.

Using the AWS CLI

You can use the following AWS CLI commands to manage S3 Intelligent-Tiering configurations:

- `delete-bucket-intelligent-tiering-configuration`
- `get-bucket-intelligent-tiering-configuration`

- [list-bucket-intelligent-tiering-configurations](#)
- [put-bucket-intelligent-tiering-configuration](#)

For instructions on setting up the AWS CLI, see [Developing with Amazon S3 using the AWS CLI \(p. 1183\)](#).

When using the AWS CLI, you cannot specify the configuration as an XML file. You must specify the JSON instead. The following is an example XML S3 Intelligent-Tiering configuration and equivalent JSON that you can specify in an AWS CLI command.

The following example puts an S3 Intelligent-Tiering configuration to the specified bucket.

Example [put-bucket-intelligent-tiering-configuration](#)

JSON

```
{  
    "Id": "string",  
    "Filter": {  
        "Prefix": "string",  
        "Tag": {  
            "Key": "string",  
            "Value": "string"  
        },  
        "And": {  
            "Prefix": "string",  
            "Tags": [  
                {  
                    "Key": "string",  
                    "Value": "string"  
                }  
                ...  
            ]  
        },  
        "Status": "Enabled"|"Disabled",  
        "Tierings": [  
            {  
                "Days": integer,  
                "AccessTier": "ARCHIVE_ACCESS"|"DEEP_ARCHIVE_ACCESS"  
            }  
            ...  
        ]  
    }  
}
```

XML

```
PUT /?intelligent-tiering&id=Id HTTP/1.1  
Host: Bucket.s3.amazonaws.com  
<?xml version="1.0" encoding="UTF-8"?>  
<IntelligentTieringConfiguration xmlns="http://s3.amazonaws.com/doc/2006-03-01/">  
    <Id>string</Id>  
    <Filter>  
        <And>  
            <Prefix>string</Prefix>  
            <Tag>  
                <Key>string</Key>  
                <Value>string</Value>  
            </Tag>  
            ...  
        </And>  
        <Prefix>string</Prefix>  
        <Tag>
```

```
<Key>string</Key>
  <Value>string</Value>
</Tag>
</Filter>
<Status>string</Status>
<Tiering>
  <AccessTier>string</AccessTier>
  <Days>integer</Days>
</Tiering>
...
</IntelligentTieringConfiguration>
```

Using the PUT API operation

You can use the [PutBucketIntelligentTieringConfiguration](#) operation for a specified bucket and up to 1,000 S3 Intelligent-Tiering configurations per bucket. You can define which objects within a bucket are eligible for the archive access tiers using a shared prefix or object tag. Using a shared prefix or object tag allows you to align to specific business applications, workflows, or internal organizations. You also have the flexibility to activate the Archive Access tier, the Deep Archive Access tier, or both.

Managing S3 Intelligent-Tiering

The S3 Intelligent-Tiering storage class delivers automatic storage cost savings in three low latency and high throughput access tiers. It also offers optional archive capabilities to help you get the lowest storage costs in the cloud for data that can be accessed in minutes to hours. The S3 Intelligent-Tiering storage class supports all Amazon S3 features, including the following:

- S3 Inventory, for verifying the access tier of objects
- S3 Replication, for replicating data to any AWS Region
- S3 Storage Lens, for viewing storage usage and activity metrics
- Server-Side Encryption, for object data
- S3 Object Lock, for preventing accidental deletion
- AWS PrivateLink, for accessing Amazon S3 through a private endpoint in a VPC

Identifying which S3 Intelligent-Tiering access tier objects are stored in

You can use [Amazon S3 Inventory](#) to get a list of your objects and their corresponding metadata, including their S3 Intelligent-Tiering access tier. Amazon S3 Inventory provides CSV, ORC, or Parquet output files that list your objects and their corresponding metadata on either a daily or weekly basis for an Amazon S3 bucket or a shared prefix. (Shared prefix refers to objects that have names that begin with a common string.)

Viewing the archive status of an object within S3 Intelligent-Tiering

You can set up an Amazon S3 event notification to receive notice when an object within the S3 Intelligent-Tiering storage class has moved to either the Archive Access tier or the Deep Archive Access tier. For more information, see [Enabling event notifications](#).

Amazon S3 can publish event notifications to an Amazon Simple Notification Service (Amazon SNS) topic, an Amazon Simple Queue Service (Amazon SQS) queue, or an AWS Lambda function. For more information, see [Amazon S3 Event Notifications \(p. 1017\)](#).

The following is an example of a message that Amazon S3 sends to publish an s3:IntelligentTiering event. For more information, see [Event message structure](#).

```
{
    "Records": [
        {
            "eventVersion": "2.3",
            "eventSource": "aws:s3",
            "awsRegion": "us-west-2",
            "eventTime": "1970-01-01T00:00:00.000Z",
            "eventName": "IntelligentTiering",
            "userIdentity": {
                "principalId": "s3.amazonaws.com"
            },
            "requestParameters": {
                "sourceIPAddress": "s3.amazonaws.com"
            },
            "responseElements": {
                "x-amz-request-id": "C3D13FE58DE4C810",
                "x-amz-id-2": "FMyUVURIY8/IgAtTv8xRjskZQpcIZ9KG4V5Wp6S7S/JRWeUWerMUE5JgHvANOjpD"
            },
            "s3": {
                "s3SchemaVersion": "1.0",
                "configurationId": "testConfigRule",
                "bucket": {
                    "name": "mybucket",
                    "ownerIdentity": {
                        "principalId": "A3NL1KOZZKExample"
                    },
                    "arn": "arn:aws:s3:::mybucket"
                },
                "object": {
                    "key": "HappyFace.jpg",
                    "size": 1024,
                    "eTag": "d41d8cd98f00b204e9800998ecf8427e",
                }
            },
            "intelligentTieringEventData": {
                "destinationAccessTier": "ARCHIVE_ACCESS"
            }
        }
    ]
}
```

You can also use a [HEAD object request](#) to view an object's archive status. If an object is stored using the S3 Intelligent-Tiering storage class and is in one of the archive tiers, the HEAD object response shows the current archive tier. It does this using the [x-amz-archive-status](#) header.

The following HEAD object request returns the metadata of an object.

Example

```
HEAD /my-image.jpg HTTP/1.1
Host: bucket.s3.<Region>.amazonaws.com
Date: Wed, 28 Oct 2009 22:32:00 GMT
Authorization: AWS AKIAIOSFODNN7EXAMPLE:02236Q3V0RonhpaBX5sCYVf1bNRuU=
```

HEAD object requests can also be used to monitor the status of a [restore-object](#) request. If the archive restoration is in progress, the HEAD object response includes the [x-amz-restore](#) header.

The following is a sample HEAD object response showing an object archived using S3 Intelligent-Tiering with a restore request in progress.

Example

```
HTTP/1.1 200 OK
x-amz-id-2: FSVaTMjrmBp3Izs1NnwBZeu7M19iI8UbxMbi0A8AirHANJBo+hEftBuiESACOMJp
x-amz-request-id: E5CEFCB143EB505A
Date: Fri, 13 Nov 2020 00:28:38 GMT
Last-Modified: Mon, 15 Oct 2012 21:58:07 GMT
ETag: "1acccb31fcf202eba0c0f41fa2f09b4d7"
x-amz-storage-class: 'INTELLIGENT_TIERING'
x-amz-archive-status: 'ARCHIVE_ACCESS'
x-amz-restore: 'ongoing-request="true"'
x-amz-restore-request-date: 'Fri, 13 Nov 2020 00:20:00 GMT'
Accept-Ranges: bytes
Content-Type: binary/octet-stream
Content-Length: 300
Server: AmazonS3
```

Restoring objects from the S3 Intelligent-Tiering Archive Access and Deep Archive Access tiers

To access objects in the S3 Intelligent-Tiering Archive Access and Deep Archive Access tiers, you must initiate the [restore request](#) and wait until the object is moved into the Frequent Access tier. For information on archived objects, see [Working with archived objects](#).

When you restore an object from the Archive Access tier or Deep Archive Access tier, the object moves back into the Frequent Access tier. Afterwards, if the object isn't accessed for 30 consecutive days, it automatically moves into the Infrequent Access tier. Then, it moves into the Archive Access tier after a minimum of 90 consecutive days of no access. It moves into the Deep Archive Access tier after a minimum of 180 consecutive days of no access.

There are no retrieval charges in S3 Intelligent-Tiering. [Standard and Bulk](#) data retrievals and restore requests are free of charge for both the Archive Access and Deep Archive Access tiers. Subsequent restore requests called on archived objects that have already been restored are billed as a GET request.

Note

When restoring an object in the S3 Intelligent-Tiering archive access tiers, the restore request uses Standard retrieval as the default retrieval option. You can specify Standard or Bulk retrieval within `GlacierJobParameters`. You can also specify Expedited retrieval from the Archive Access tier, which is charged at the Expedited request and retrieval rate.

You can restore an archived object using the Amazon S3 console, the REST API, and the AWS Command Line Interface (AWS CLI).

Using the S3 console

To restore an object using the Amazon S3 console.

1. Sign in to the AWS Management Console and open the Amazon S3 console at <https://console.aws.amazon.com/s3/>.
2. In the **Bucket name** list, choose the name of the bucket that contains the objects that you want to restore.
3. In the **Objects** list, select one or more of the objects that you are restoring, choose **Actions**, and then choose **Restore from S3 Intelligent-Tiering Archive or Deep Archive Access**.
4. Choose **Restore**.

Note

Unlike in S3 Glacier Flexible Retrieval and S3 Glacier Deep Archive restore requests, you do not need to choose the tier you want to upgrade to. Objects from the S3 Intelligent-Tiering

Archive Access and Deep Archive Access tiers automatically restore to the Frequent Access tier.

Using the REST API

Amazon S3 provides an API operation for you to initiate an archive restoration. For more information, see [RestoreObject](#) in the *Amazon Simple Storage Service API Reference*.

Using the AWS CLI

Use the `restore-object` command to restore objects from the S3 Intelligent-Tiering Archive Access or Deep Archive Access tiers.

The following example restores object `dir1/example.obj` in `awsexamplebucket`.

```
aws s3api restore-object --bucket awsexamplebucket --key dir1/example.obj --restore-request '{}'
```

You can use the following command to monitor the status of your `restore-object` request.

```
aws s3api head-object --bucket awsexamplebucket --key dir1/example.obj
```

For more information, see [restore-object](#) in the AWS CLI Command Reference.

Note

Unlike in the S3 Glacier Flexible Retrieval and S3 Glacier Deep Archive storage classes, restore requests for S3 Intelligent-Tiering objects don't accept the days value.

Checking the restore status of an object

You can check the progress of your object's restoration on the [Object overview](#) page on the Amazon S3 console. For more information, see [Viewing an object overview in the Amazon S3 console \(p. 256\)](#). This page will show that the restoration is **In progress**. You can use request to be notified of object restoration completion by using `s3:ObjectRestore:Completed` with the [Amazon S3 Event Notifications](#) feature.

The following table summarizes archived object retrieval speeds.

Note

[Expedited retrievals](#) are a premium feature available for the S3 Intelligent-Tiering Archive Access tier and are charged at the Expedited request and retrieval rate.

For information about paying for Amazon S3, see [Amazon S3 Pricing](#).

Managing your storage lifecycle

To manage your objects so that they are stored cost effectively throughout their lifecycle, configure their [Amazon S3 Lifecycle](#). An [S3 Lifecycle configuration](#) is a set of rules that define actions that Amazon S3 applies to a group of objects. There are two types of actions:

- **Transition actions** – These actions define when objects transition to another storage class. For example, you might choose to transition objects to the S3 Standard-IA storage class 30 days after creating them, or archive objects to the S3 Glacier Flexible Retrieval storage class one year after creating them. For more information, see [Using Amazon S3 storage classes \(p. 688\)](#).

There are costs associated with lifecycle transition requests. For pricing information, see [Amazon S3 pricing](#).

- **Expiration actions** – These actions define when objects expire. Amazon S3 deletes expired objects on your behalf.

Lifecycle expiration costs depend on when you choose to expire objects. For more information, see [Expiring objects \(p. 707\)](#).

If there is any delay between when an object becomes eligible for a lifecycle action and when Amazon S3 transfers or expires your object, billing changes are applied as soon as the object becomes eligible for the lifecycle action. For example, if an object is scheduled to expire and Amazon S3 does not immediately expire the object, you won't be charged for storage after the expiration time. The one exception to this behavior is if you have a lifecycle rule to transition to the S3 Intelligent-Tiering storage class. In that case, billing changes do not occur until the object has transitioned to S3 Intelligent-Tiering.

For more information about S3 Lifecycle rules, see [Lifecycle configuration elements \(p. 721\)](#).

Managing object lifecycle

Define S3 Lifecycle configuration rules for objects that have a well-defined lifecycle. For example:

- If you upload periodic logs to a bucket, your application might need them for a week or a month. After that, you might want to delete them.
- Some documents are frequently accessed for a limited period of time. After that, they are infrequently accessed. At some point, you might not need real-time access to them, but your organization or regulations might require you to archive them for a specific period. After that, you can delete them.
- You might upload some types of data to Amazon S3 primarily for archival purposes. For example, you might archive digital media, financial and healthcare records, raw genomics sequence data, long-term database backups, and data that must be retained for regulatory compliance.

With S3 Lifecycle configuration rules, you can tell Amazon S3 to transition objects to less-expensive storage classes, or archive or delete them.

Creating a lifecycle configuration

An S3 Lifecycle configuration is an XML file that consists of a set of rules with predefined actions that you want Amazon S3 to perform on objects during their lifetime.

You can also configure the lifecycle by using the Amazon S3 console, REST API, AWS SDKs, and the AWS Command Line Interface (AWS CLI). For more information, see [Setting lifecycle configuration on a bucket \(p. 708\)](#).

Amazon S3 provides a set of REST API operations for managing lifecycle configuration on a bucket. Amazon S3 stores the configuration as a *lifecycle subresource* that is attached to your bucket. For details, see the following:

[PUT Bucket lifecycle](#)

[GET Bucket lifecycle](#)

[DELETE Bucket lifecycle](#)

For more information about creating a lifecycle configuration, see the following topics:

Topics

- [Transitioning objects using Amazon S3 Lifecycle \(p. 703\)](#)
- [Expiring objects \(p. 707\)](#)

- [Setting lifecycle configuration on a bucket \(p. 708\)](#)
- [Lifecycle and other bucket configurations \(p. 719\)](#)
- [Configuring Lifecycle event notifications \(p. 720\)](#)
- [Lifecycle configuration elements \(p. 721\)](#)
- [Examples of S3 Lifecycle configuration \(p. 728\)](#)

Transitioning objects using Amazon S3 Lifecycle

You can add rules in an S3 Lifecycle configuration to tell Amazon S3 to transition objects to another Amazon S3 storage class. For more information about storage classes, see [Using Amazon S3 storage classes \(p. 688\)](#). Some examples of when you might use S3 Lifecycle configurations in this way include the following:

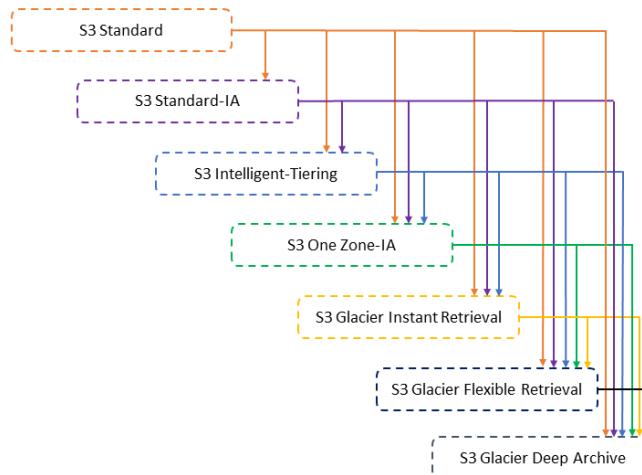
- When you know that objects are infrequently accessed, you might transition them to the S3 Standard-IA storage class.
- You might want to archive objects that you don't need to access in real time to the S3 Glacier Flexible Retrieval storage class.

The following sections describe supported transitions, related constraints, and transitioning to the S3 Glacier Flexible Retrieval storage class.

Supported transitions and related constraints

In an S3 Lifecycle configuration, you can define rules to transition objects from one storage class to another to save on storage costs. When you don't know the access patterns of your objects, or if your access patterns are changing over time, you can transition the objects to the S3 Intelligent-Tiering storage class for automatic cost savings. For information about storage classes, see [Using Amazon S3 storage classes \(p. 688\)](#).

Amazon S3 supports a waterfall model for transitioning between storage classes, as shown in the following diagram.



Supported lifecycle transitions

Amazon S3 supports the following lifecycle transitions between storage classes using an S3 Lifecycle configuration.

You *can transition* from the following:

- The S3 Standard storage class to any other storage class.
- The S3 Standard-IA storage class to the S3 Intelligent-Tiering, S3 One Zone-IA, S3 Glacier Instant Retrieval, S3 Glacier Flexible Retrieval, or S3 Glacier Deep Archive storage classes.
- The S3 Intelligent-Tiering storage class to the S3 One Zone-IA, S3 Glacier Instant Retrieval, S3 Glacier Flexible Retrieval, or S3 Glacier Deep Archive storage classes.
- The S3 One Zone-IA storage class to the S3 Glacier Flexible Retrieval or S3 Glacier Deep Archive storage classes.
- The S3 Glacier Instant Retrieval storage class to the S3 Glacier Flexible Retrieval or S3 Glacier Deep Archive storage classes.
- The S3 Glacier Flexible Retrieval storage class to the S3 Glacier Deep Archive storage class.
- Any storage class to the S3 Glacier Deep Archive storage class.

Unsupported lifecycle transitions

Amazon S3 does not support any of the following lifecycle transitions.

You *can't transition* from the following:

- Any storage class to the S3 Standard storage class.
- Any storage class to the Reduced Redundancy Storage (RRS) class.
- The S3 Intelligent-Tiering storage class to the S3 Standard-IA storage class.
- The S3 One Zone-IA storage class to the S3 Intelligent-Tiering, S3 Standard-IA, or S3 Glacier Instant Retrieval storage classes.

Constraints

Lifecycle storage class transitions have the following constraints:

Object Size and Transitions from S3 Standard or S3 Standard-IA to S3 Intelligent-Tiering, S3 Standard-IA, or S3 One Zone-IA

When you transition objects from the S3 Standard or S3 Standard-IA storage classes to S3 Intelligent-Tiering, S3 Standard-IA, or S3 One Zone-IA, the following object size constraints apply:

- **Larger objects** – For the following transitions, there is a cost benefit to transitioning larger objects:
 - From the S3 Standard or S3 Standard-IA storage classes to S3 Intelligent-Tiering.
 - From the S3 Standard storage class to S3 Standard-IA or S3 One Zone-IA.
- **Objects smaller than 128 KB** – For the following transitions, Amazon S3 does not transition objects that are smaller than 128 KB:
 - From the S3 Standard or S3 Standard-IA storage classes to S3 Intelligent-Tiering or S3 Glacier Instant Retrieval.
 - From the S3 Standard storage class to S3 Standard-IA or S3 One Zone-IA.

Note

You can filter lifecycle rules based on object size.

Minimum Days for Transition from S3 Standard or S3 Standard-IA to S3 Standard-IA or S3 One Zone-IA

Before you transition objects from the S3 Standard or S3 Standard-IA storage classes to S3 Standard-IA or S3 One Zone-IA, you must store them at least 30 days in the S3 Standard storage class. For example, you cannot create a Lifecycle rule to transition objects to the S3 Standard-IA storage class one day after you create them. Amazon S3 doesn't transition objects within the first 30 days because newer objects are often accessed more frequently or deleted sooner than is suitable for S3 Standard-IA or S3 One Zone-IA storage.

Similarly, if you are transitioning noncurrent objects (in versioned buckets), you can transition only objects that are at least 30 days noncurrent to S3 Standard-IA or S3 One Zone-IA storage.

Minimum 30-Day Storage Charge for S3 Standard-IA and S3 One Zone-IA

The S3 Standard-IA and S3 One Zone-IA storage classes have a minimum 30-day storage charge. Therefore, you can't specify a single Lifecycle rule for both an S3 Standard-IA or S3 One Zone-IA transition and an S3 Glacier Flexible Retrieval or S3 Glacier Deep Archive transition when the S3 Glacier Flexible Retrieval or S3 Glacier Deep Archive transition occurs less than 30 days after the S3 Standard-IA or S3 One Zone-IA transition.

The same 30-day minimum applies when you specify a transition from S3 Standard-IA storage to S3 One Zone-IA. You can specify two rules to accomplish this, but you pay minimum storage charges. For more information about cost considerations, see [Amazon S3 pricing](#).

Manage an object's complete lifecycle

You can combine these S3 Lifecycle actions to manage an object's complete lifecycle. For example, suppose that the objects you create have a well-defined lifecycle. Initially, the objects are frequently accessed for a period of 30 days. Then, objects are infrequently accessed for up to 90 days. After that, the objects are no longer needed, so you might choose to archive or delete them.

In this scenario, you can create an S3 Lifecycle rule in which you specify the initial transition action to S3 Intelligent-Tiering, S3 Standard-IA, or S3 One Zone-IA storage, another transition action to S3 Glacier Flexible Retrieval storage for archiving, and an expiration action. As you move the objects from one storage class to another, you save on storage cost. For more information about cost considerations, see [Amazon S3 pricing](#).

Transitioning to the S3 Glacier Flexible Retrieval and S3 Glacier Deep Archive storage classes (object archival)

Using S3 Lifecycle configuration, you can transition objects to the S3 Glacier Flexible Retrieval or S3 Glacier Deep Archive storage classes for archiving. When you choose the S3 Glacier Flexible Retrieval or S3 Glacier Deep Archive storage class, your objects remain in Amazon S3. You cannot access them directly through the separate Amazon S3 Glacier service. For more general information about S3 Glacier see, [What is Amazon S3 Glacier in the Amazon S3 Glacier Developer Guide](#).

Before you archive objects, review the following sections for relevant considerations.

General considerations

The following are the general considerations for you to consider before you archive objects:

- Encrypted objects remain encrypted throughout the storage class transition process.
- Objects that are stored in the S3 Glacier Flexible Retrieval or S3 Glacier Deep Archive storage classes are not available in real time.

Archived objects are Amazon S3 objects, but before you can access an archived object, you must first restore a temporary copy of it. The restored object copy is available only for the duration you specify

in the restore request. After that, Amazon S3 deletes the temporary copy, and the object remains archived in S3 Glacier Flexible Retrieval.

You can restore an object by using the Amazon S3 console or programmatically by using the AWS SDK wrapper libraries or the Amazon S3 REST API in your code. For more information, see [Restoring an archived object \(p. 673\)](#).

- Objects that are stored in the S3 Glacier Flexible Retrieval storage class can only be transitioned to the S3 Glacier Deep Archive storage class.

You can use an S3 Lifecycle configuration rule to convert the storage class of an object from S3 Glacier Flexible Retrieval to the S3 Glacier Deep Archive storage class only. If you want to change the storage class of an object that is stored in S3 Glacier Flexible Retrieval to a storage class other than S3 Glacier Deep Archive, you must use the restore operation to make a temporary copy of the object first. Then use the copy operation to overwrite the object specifying S3 Standard, S3 Intelligent-Tiering, S3 Standard-IA, S3 One Zone-IA, or Reduced Redundancy as the storage class.

- The transition of objects to the S3 Glacier Deep Archive storage class can go only one way.

You cannot use an S3 Lifecycle configuration rule to convert the storage class of an object from S3 Glacier Deep Archive to any other storage class. If you want to change the storage class of an archived object to another storage class, you must use the restore operation to make a temporary copy of the object first. Then use the copy operation to overwrite the object specifying S3 Standard, S3 Intelligent-Tiering, S3 Standard-IA, S3 One Zone-IA, S3 Glacier Instant Retrieval, S3 Glacier Flexible Retrieval, or Reduced Redundancy Storage as the storage class.

- The objects that are stored in the S3 Glacier Flexible Retrieval and S3 Glacier Deep Archive storage classes are visible and available only through Amazon S3. They are not available through the separate Amazon S3 Glacier service.

These are Amazon S3 objects, and you can access them only by using the Amazon S3 console or the Amazon S3 API. You cannot access the archived objects through the separate Amazon S3 Glacier console or the Amazon S3 Glacier API.

Cost considerations

If you are planning to archive infrequently accessed data for a period of months or years, the S3 Glacier Flexible Retrieval and S3 Glacier Deep Archive storage classes can reduce your storage costs. However, to ensure that the S3 Glacier Flexible Retrieval or S3 Glacier Deep Archive storage class is appropriate for you, consider the following:

- **Storage overhead charges** – When you transition objects to the S3 Glacier Flexible Retrieval or S3 Glacier Deep Archive storage class, a fixed amount of storage is added to each object to accommodate metadata for managing the object.
 - For each object archived to S3 Glacier Flexible Retrieval or S3 Glacier Deep Archive, Amazon S3 uses 8 KB of storage for the name of the object and other metadata. Amazon S3 stores this metadata so that you can get a real-time list of your archived objects by using the Amazon S3 API. For more information, see [Get Bucket \(List Objects\)](#). You are charged S3 Standard rates for this additional storage.
 - For each object that is archived to S3 Glacier Flexible Retrieval or S3 Glacier Deep Archive, Amazon S3 adds 32 KB of storage for index and related metadata. This extra data is necessary to identify and restore your object. You are charged S3 Glacier Flexible Retrieval or S3 Glacier Deep Archive rates for this additional storage.

If you are archiving small objects, consider these storage charges. Also consider aggregating many small objects into a smaller number of large objects to reduce overhead costs.

- **Number of days you plan to keep objects archived** – S3 Glacier Flexible Retrieval and S3 Glacier Deep Archive are long-term archival solutions. The minimal storage duration period is 90 days for the S3 Glacier Flexible Retrieval storage class and 180 days for S3 Glacier Deep Archive. Deleting data that

is archived to Amazon S3 Glacier doesn't incur charges if the objects you delete are archived for more than the minimal storage duration period. If you delete or overwrite an archived object within the minimal duration period, Amazon S3 charges a prorated early deletion fee. For information about the early deletion fee, see the "How am I charged for deleting objects from Amazon S3 Glacier that are less than 90 days old?" question on the [Amazon S3 FAQ](#).

- **S3 Glacier Flexible Retrieval and S3 Glacier Deep Archive transition request charges** – Each object that you transition to the S3 Glacier Flexible Retrieval or S3 Glacier Deep Archive storage class constitutes one transition request. There is a cost for each such request. If you plan to transition a large number of objects, consider the request costs. If you are archiving small objects, consider aggregating many small objects into a smaller number of large objects to reduce transition request costs.
- **S3 Glacier Flexible Retrieval and S3 Glacier Deep Archive data restore charges** – S3 Glacier Flexible Retrieval and S3 Glacier Deep Archive are designed for long-term archival of data that you access infrequently. For information about data restoration charges, see the "How much does it cost to retrieve data from Amazon S3 Glacier?" question on the [Amazon S3 FAQ](#). For information about how to restore data from Amazon S3 Glacier, see [Restoring an archived object \(p. 673\)](#).

When you archive objects to Amazon S3 Glacier by using S3 Lifecycle management, Amazon S3 transitions these objects asynchronously. There might be a delay between the transition date in the S3 Lifecycle configuration rule and the date of the physical transition. You are charged Amazon S3 Glacier prices based on the transition date specified in the rule. For more information, see the Amazon S3 Glacier section of the [Amazon S3 FAQ](#).

The Amazon S3 product detail page provides pricing information and example calculations for archiving Amazon S3 objects. For more information, see the following topics:

- "How is my storage charge calculated for Amazon S3 objects archived to Amazon S3 Glacier?" on the [Amazon S3 FAQ](#).
- "How am I charged for deleting objects from Amazon S3 Glacier that are less than 90 days old?" on the [Amazon S3 FAQ](#).
- "How much does it cost to retrieve data from Amazon S3 Glacier?" on the [Amazon S3 FAQ](#).
- [Amazon S3 pricing](#) for storage costs for the different storage classes.

Restoring archived objects

Archived objects are not accessible in real time. You must first initiate a restore request and then wait until a temporary copy of the object is available for the duration that you specify in the request. After you receive a temporary copy of the restored object, the object's storage class remains S3 Glacier Flexible Retrieval or S3 Glacier Deep Archive. (A [HEAD Object](#) or [GET Object](#) API operation request will return S3 Glacier Flexible Retrieval or S3 Glacier Deep Archive as the storage class.)

Note

When you restore an archive, you are paying for both the archive (S3 Glacier Flexible Retrieval or S3 Glacier Deep Archive rate) and a copy that you restored temporarily (Reduced Redundancy Storage rate). For information about pricing, see [Amazon S3 pricing](#).

You can restore an object copy programmatically or by using the Amazon S3 console. Amazon S3 processes only one restore request at a time per object. For more information, see [Restoring an archived object \(p. 673\)](#).

Expiring objects

When an object reaches the end of its lifetime based on its lifecycle policy, Amazon S3 queues it for removal and removes it asynchronously. There might be a delay between the expiration date and the

date at which Amazon S3 removes an object. You are not charged for expiration or the storage time associated with an object that has expired.

When dealing with buckets that have versioning enabled, realize that you will need to create a transition for noncurrent versions as well to avoid being charged for objects that have noncurrent versions. For more information about noncurrent version transitions, see [Lifecycle configuration elements \(p. 721\)](#).

To find when an object is scheduled to expire, use the [HEAD Object](#) or the [GET Object](#) API operations. These API operations return response headers that provide this information.

If you create an S3 Lifecycle expiration rule that causes objects that have been in S3 Standard-IA or S3 One Zone-IA storage for less than 30 days to expire, you are charged for 30 days. If you create a Lifecycle expiration rule that causes objects that have been in S3 Glacier Flexible Retrieval storage for less than 90 days to expire, you are charged for 90 days. If you create a Lifecycle expiration rule that causes objects that have been in S3 Glacier Deep Archive storage for less than 180 days to expire, you are charged for 180 days. For more information, see [Amazon S3 pricing](#) and [Using the S3 console \(p. 709\)](#).

Setting lifecycle configuration on a bucket

This section explains how you can set a S3 Lifecycle configuration on a bucket using AWS SDKs, the AWS CLI, or the Amazon S3 console. For information about S3 Lifecycle configuration, see [Managing your storage lifecycle \(p. 701\)](#).

You can use lifecycle rules to define actions that you want Amazon S3 to take during an object's lifetime (for example, transition objects to another storage class, archive them, or delete them after a specified period of time).

Before you set a lifecycle configuration, note the following:

Propagation delay

When you add an S3 Lifecycle configuration to a bucket, there is usually some lag before a new or updated Lifecycle configuration is fully propagated to all the Amazon S3 systems. Expect a delay of a few minutes before the configuration fully takes effect. This delay can also occur when you delete an S3 Lifecycle configuration.

Disabling or deleting Lifecycle rules

When you disable or delete Lifecycle rules, Amazon S3 stops scheduling new objects for deletion or transition after a small delay. Any objects that were already scheduled are unscheduled and are not deleted or transitioned.

Existing and new objects

When you add a Lifecycle configuration to a bucket, the configuration rules apply to both existing objects and objects that you add later. For example, if you add a Lifecycle configuration rule today with an expiration action that causes objects with a specific prefix to expire 30 days after creation, Amazon S3 will queue for removal any existing objects that are more than 30 days old.

Changes in billing

There may be a lag between when the Lifecycle configuration rules are satisfied and when the action triggered by satisfying the rule is taken. However, changes in billing happen as soon as the Lifecycle configuration rule is satisfied, even if the action is not yet taken.

For example, after the object expiration time, you are not charged for storage, even if the object is not deleted immediately. Another example, as soon as the object transition time elapses, you are charged

S3 Glacier Flexible Retrieval storage rates, even if the object is not immediately transitioned to the S3 Glacier Flexible Retrieval storage class. Lifecycle transitions to the S3 Intelligent-Tiering storage class are the exception. Changes in billing do not happen until the object has transitioned into the S3 Intelligent-Tiering storage class.

Using the S3 console

You can define a lifecycle rules for all objects or a subset of objects in the bucket by using a shared prefix (objects names that begin with a common string) or a tag. Using a lifecycle rule you can define actions specific to current and non-current object versions. For more information, see the following:

- [Managing your storage lifecycle \(p. 701\)](#)
- [Using versioning in S3 buckets \(p. 638\)](#)

To create a lifecycle rule

1. Sign in to the AWS Management Console and open the Amazon S3 console at <https://console.aws.amazon.com/s3/>.
2. In the **Buckets** list, choose the name of the bucket that you want to create a lifecycle rule for.
3. Choose the **Management** tab, and choose **Create lifecycle rule**.
4. In **Lifecycle rule name**, enter a name for your rule.

The name must be unique within the bucket.
5. Choose the scope of the lifecycle rule:
 - To apply this lifecycle rule to *all objects with a specific prefix or tag*, choose **Limit the scope to specific prefixes or tags**.
 - To limit the scope by prefix, in **Prefix**, enter the prefix.
 - To limit the scope by tag, choose **Add tag**, and enter the tag key and value.
- For more information about object name prefixes, see [Creating object key names \(p. 150\)](#). For more information about object tags, see [Categorizing your storage using tags \(p. 825\)](#).
- To apply this lifecycle rule to *all objects in the bucket*, choose **This rule applies to all objects in the bucket**, and choose **I acknowledge that this rule applies to all objects in the bucket**.
6. To filter a rule by object size, you can check **Specify minimum object size**, **Specify maximum object size**, or both options.
 - When you're specifying a **minimum object size** or **maximum object size**, the value must be larger than 0 bytes and up to 5TB. You can specify this value in bytes, KB, MB, or GB.
 - When you're specifying both, the maximum object size must be larger than the minimum object size.
7. Under **Lifecycle rule actions**, choose the actions that you want your lifecycle rule to perform:
 - Transition *current* versions of objects between storage classes
 - Transition *previous* versions of objects between storage classes
 - Expire *current* versions of objects
 - Permanently delete *previous* versions of objects
 - Delete expired delete markers or incomplete multipart uploads
- Depending on the actions that you choose, different options appear.
8. To transition *current* versions of objects between storage classes, under **Transition current versions of objects between storage classes**:

- a. In **Storage class transitions**, choose the storage class to transition to:
 - Standard-IA
 - Intelligent-Tiering
 - One Zone-IA
 - S3 Glacier Flexible Retrieval
 - Glacier Deep Archive
- b. In **Days after object creation**, enter the number of days after creation to transition the object.

For more information about storage classes, see [Using Amazon S3 storage classes \(p. 688\)](#). You can define transitions for current or previous object versions or for both current and previous versions. Versioning enables you to keep multiple versions of an object in one bucket. For more information about versioning, see [Using the S3 console \(p. 644\)](#).

Important

When you choose the S3 Glacier Flexible Retrieval or Glacier Deep Archive storage class, your objects remain in Amazon S3. You cannot access them directly through the separate Amazon S3 Glacier service. For more information, see [Transitioning objects using Amazon S3 Lifecycle \(p. 703\)](#).

9. To transition *non-current* versions of objects between storage classes, under **Transition non-current versions of objects between storage classes**:
 - a. In **Storage class transitions**, choose the storage class to transition to:
 - Standard-IA
 - Intelligent-Tiering
 - One Zone-IA
 - S3 Glacier Flexible Retrieval
 - Glacier Deep Archive
 - b. In **Days after object becomes non-current**, enter the number of days after creation to transition the object.
10. To expire *current* versions of objects, under **Expire previous versions of objects**, in **Number of days after object creation**, enter the number of days.

Important

In a non-versioned bucket the expiration action results in Amazon S3 permanently removing the object. For more information about lifecycle actions, see [Elements to describe lifecycle actions \(p. 725\)](#).

11. To permanently delete previous versions of objects, under **Permanently delete noncurrent versions of objects**, in **Days after objects become noncurrent**, enter the number of days. You can optionally specify the number of newer versions to retain by entering a value under **Number of newer versions to retain**.
12. Under **Delete expired delete markers or incomplete multipart uploads**, choose **Delete expired object delete markers** and **Delete incomplete multipart uploads**. Then, enter the number of days after the multipart upload initiation that you want to end and clean up incomplete multipart uploads.

For more information about multipart uploads, see [Uploading and copying objects using multipart upload \(p. 167\)](#).

13. Choose **Create rule**.

If the rule does not contain any errors, Amazon S3 enables it, and you can see it on the **Management** tab under **Lifecycle rules**.

For information about CloudFormation templates and examples, see [Working with AWS CloudFormation templates](#) and [AWS::S3::Bucket](#) in the *AWS CloudFormation User Guide*.

Using the AWS CLI

You can use the following AWS CLI commands to manage S3 Lifecycle configurations:

- `put-bucket-lifecycle-configuration`
- `get-bucket-lifecycle-configuration`
- `delete-bucket-lifecycle`

For instructions on setting up the AWS CLI, see [Developing with Amazon S3 using the AWS CLI \(p. 1183\)](#).

The Amazon S3 Lifecycle configuration is an XML file. But when using the AWS CLI, you cannot specify the XML. You must specify the JSON instead. The following are example XML Lifecycle configurations and equivalent JSON that you can specify in an AWS CLI command.

Consider the following example S3 Lifecycle configuration.

Example Example 1

JSON

```
{  
    "Rules": [  
        {  
            "Filter": {  
                "Prefix": "documents/"  
            },  
            "Status": "Enabled",  
            "Transitions": [  
                {  
                    "Days": 365,  
                    "StorageClass": "GLACIER"  
                }  
            ],  
            "Expiration": {  
                "Days": 3650  
            },  
            "ID": "ExampleRule"  
        }  
    ]  
}
```

XML

```
<LifecycleConfiguration>  
    <Rule>  
        <ID>ExampleRule</ID>  
        <Filter>  
            <Prefix>documents/</Prefix>  
        </Filter>  
        <Status>Enabled</Status>  
        <Transition>  
            <Days>365</Days>  
            <StorageClass>GLACIER</StorageClass>  
        </Transition>  
        <Expiration>  
            <Days>3650</Days>  
        </Expiration>  
    </Rule>  
</LifecycleConfiguration>
```

```
</Expiration>
</Rule>
</LifecycleConfiguration>
```

Example Example 2

JSON

```
{
    "Rules": [
        {
            "ID": "id-1",
            "Filter": {
                "And": [
                    {
                        "Prefix": "myprefix",
                        "Tags": [
                            {
                                "Value": "mytagvalue1",
                                "Key": "mytagkey1"
                            },
                            {
                                "Value": "mytagvalue2",
                                "Key": "mytagkey2"
                            }
                        ]
                    }
                ],
                "Status": "Enabled",
                "Expiration": {
                    "Days": 1
                }
            }
        ]
    ]
}
```

XML

```
<LifecycleConfiguration xmlns="http://s3.amazonaws.com/doc/2006-03-01/">
    <Rule>
        <ID>id-1</ID>
        <Expiration>
            <Days>1</Days>
        </Expiration>
        <Filter>
            <And>
                <Prefix>myprefix</Prefix>
                <Tag>
                    <Key>mytagkey1</Key>
                    <Value>mytagvalue1</Value>
                </Tag>
                <Tag>
                    <Key>mytagkey2</Key>
                    <Value>mytagvalue2</Value>
                </Tag>
            </And>
        </Filter>
        <Status>Enabled</Status>
    </Rule>
</LifecycleConfiguration>
```

You can test the put-bucket-lifecycle-configuration as follows.

To test the configuration

1. Save the JSON Lifecycle configuration in a file (`lifecycle.json`).
2. Run the following AWS CLI command to set the Lifecycle configuration on your bucket.

```
$ aws s3api put-bucket-lifecycle-configuration \
--bucket bucketname \
--lifecycle-configuration file://lifecycle.json
```

3. To verify, retrieve the S3 Lifecycle configuration using the `get-bucket-lifecycle-configuration` AWS CLI command as follows.

```
$ aws s3api get-bucket-lifecycle-configuration \
--bucket bucketname
```

4. To delete the S3 Lifecycle configuration use the `delete-bucket-lifecycle` AWS CLI command as follows.

```
aws s3api delete-bucket-lifecycle \
--bucket bucketname
```

Using the AWS SDKs

Java

You can use the AWS SDK for Java to manage the S3 Lifecycle configuration of a bucket. For more information about managing S3 Lifecycle configuration, see [Managing your storage lifecycle \(p. 701\)](#).

Note

When you add S3 Lifecycle configuration to a bucket, Amazon S3 replaces the bucket's current Lifecycle configuration, if there is one. To update a configuration, you retrieve it, make the desired changes, and then add the revised configuration to the bucket.

The following example shows how to use the AWS SDK for Java to add, update, and delete the Lifecycle configuration of a bucket. The example does the following:

- Adds a Lifecycle configuration to a bucket.
- Retrieves the Lifecycle configuration and updates it by adding another rule.
- Adds the modified Lifecycle configuration to the bucket. Amazon S3 replaces the existing configuration.
- Retrieves the configuration again and verifies that it has the right number of rules by printing the number of rules.
- Deletes the Lifecycle configuration and verifies that it has been deleted by attempting to retrieve it again.

For instructions on creating and testing a working sample, see [Testing the Amazon S3 Java Code Examples \(p. 1192\)](#).

```
import com.amazonaws.AmazonServiceException;
import com.amazonaws.SdkClientException;
import com.amazonaws.auth.profile.ProfileCredentialsProvider;
import com.amazonaws.regions.Regions;
import com.amazonaws.services.s3.AmazonS3;
```

```
import com.amazonaws.services.s3.AmazonS3ClientBuilder;
import com.amazonaws.services.s3.model.BucketLifecycleConfiguration;
import com.amazonaws.services.s3.model.BucketLifecycleConfiguration.Transition;
import com.amazonaws.services.s3.model.StorageClass;
import com.amazonaws.services.s3.model.Tag;
import com.amazonaws.services.s3.model.lifecycle.LifecycleAndOperator;
import com.amazonaws.services.s3.model.lifecycle.LifecycleFilter;
import com.amazonaws.services.s3.model.lifecycle.LifecyclePrefixPredicate;
import com.amazonaws.services.s3.model.lifecycle.LifecycleTagPredicate;

import java.io.IOException;
import java.util.Arrays;

public class LifecycleConfiguration {

    public static void main(String[] args) throws IOException {
        Regions clientRegion = Regions.DEFAULT_REGION;
        String bucketName = "*** Bucket name ***";

        // Create a rule to archive objects with the "glacierobjects/" prefix to
        // Glacier immediately.
        BucketLifecycleConfiguration.Rule rule1 = new
        BucketLifecycleConfiguration.Rule()
            .withId("Archive immediately rule")
            .withFilter(new LifecycleFilter(new
        LifecyclePrefixPredicate("glacierobjects/")))
            .addTransition(new
        Transition().withDays(0).withStorageClass(StorageClass.Glacier))
            .withStatus(BucketLifecycleConfiguration.ENABLED);

        // Create a rule to transition objects to the Standard-Infrequent Access
        // storage class
        // after 30 days, then to Glacier after 365 days. Amazon S3 will delete the
        // objects after 3650 days.
        // The rule applies to all objects with the tag "archive" set to "true".
        BucketLifecycleConfiguration.Rule rule2 = new
        BucketLifecycleConfiguration.Rule()
            .withId("Archive and then delete rule")
            .withFilter(new LifecycleFilter(new LifecycleTagPredicate(new
        Tag("archive", "true"))))
            .addTransition(new
        Transition().withDays(30).withStorageClass(StorageClass.StandardInfrequentAccess))
            .addTransition(new
        Transition().withDays(365).withStorageClass(StorageClass.Glacier))
            .withExpirationInDays(3650)
            .withStatus(BucketLifecycleConfiguration.ENABLED);

        // Add the rules to a new BucketLifecycleConfiguration.
        BucketLifecycleConfiguration configuration = new BucketLifecycleConfiguration()
            .withRules(Arrays.asList(rule1, rule2));

        try {
            AmazonS3 s3Client = AmazonS3ClientBuilder.standard()
                .withCredentials(new ProfileCredentialsProvider())
                .withRegion(clientRegion)
                .build();

            // Save the configuration.
            s3Client.setBucketLifecycleConfiguration(bucketName, configuration);

            // Retrieve the configuration.
            configuration = s3Client.getBucketLifecycleConfiguration(bucketName);

            // Add a new rule with both a prefix predicate and a tag predicate.
            configuration.getRules().add(new
            BucketLifecycleConfiguration.Rule().withId("NewRule"))
        }
    }
}
```

```
        .withFilter(new LifecycleFilter(new LifecycleAndOperator(
            Arrays.asList(new
                LifecyclePrefixPredicate("YearlyDocuments/"),
                new LifecycleTagPredicate(new Tag("expire_after",
                    "ten_years"))))))
            .withExpirationInDays(3650)
            .withStatus(BucketLifecycleConfiguration.ENABLED));

        // Save the configuration.
        s3Client.setBucketLifecycleConfiguration(bucketName, configuration);

        // Retrieve the configuration.
        configuration = s3Client.getBucketLifecycleConfiguration(bucketName);

        // Verify that the configuration now has three rules.
        configuration = s3Client.getBucketLifecycleConfiguration(bucketName);
        System.out.println("Expected # of rules = 3; found: " +
            configuration.getRules().size());

        // Delete the configuration.
        s3Client.deleteBucketLifecycleConfiguration(bucketName);

        // Verify that the configuration has been deleted by attempting to retrieve
        it.
        configuration = s3Client.getBucketLifecycleConfiguration(bucketName);
        String s = (configuration == null) ? "No configuration found." :
        "Configuration found.";
        System.out.println(s);
    } catch (AmazonServiceException e) {
        // The call was transmitted successfully, but Amazon S3 couldn't process
        // it, so it returned an error response.
        e.printStackTrace();
    } catch (SdkClientException e) {
        // Amazon S3 couldn't be contacted for a response, or the client
        // couldn't parse the response from Amazon S3.
        e.printStackTrace();
    }
}
}
```

.NET

You can use the AWS SDK for .NET to manage the S3 Lifecycle configuration on a bucket. For more information about managing Lifecycle configuration, see [Managing your storage lifecycle \(p. 701\)](#).

Note

When you add a Lifecycle configuration, Amazon S3 replaces the existing configuration on the specified bucket. To update a configuration, you must first retrieve the Lifecycle configuration, make the changes, and then add the revised Lifecycle configuration to the bucket.

The following example shows how to use the AWS SDK for .NET to add, update, and delete a bucket's Lifecycle configuration. The code example does the following:

- Adds a Lifecycle configuration to a bucket.
- Retrieves the Lifecycle configuration and updates it by adding another rule.
- Adds the modified Lifecycle configuration to the bucket. Amazon S3 replaces the existing Lifecycle configuration.
- Retrieves the configuration again and verifies it by printing the number of rules in the configuration.
- Deletes the Lifecycle configuration and verifies the deletion.

For instructions on how to create and test a working sample, see [Running the Amazon S3 .NET Code Examples \(p. 1193\)](#).

```
using Amazon;
using Amazon.S3;
using Amazon.S3.Model;
using System;
using System.Collections.Generic;
using System.Threading.Tasks;

namespace Amazon.DocSamples.S3
{
    class LifecycleTest
    {
        private const string bucketName = "*** bucket name ***";
        // Specify your bucket region (an example region is shown).
        private static readonly RegionEndpoint bucketRegion = RegionEndpoint.USWest2;
        private static IAmazonS3 client;
        public static void Main()
        {
            client = new AmazonS3Client(bucketRegion);
            AddUpdateDeleteLifecycleConfigAsync().Wait();
        }

        private static async Task AddUpdateDeleteLifecycleConfigAsync()
        {
            try
            {
                var lifeCycleConfiguration = new LifecycleConfiguration()
                {
                    Rules = new List<LifecycleRule>
                    {
                        new LifecycleRule
                        {
                            Id = "Archive immedately rule",
                            Filter = new LifecycleFilter()
                            {
                                LifecycleFilterPredicate = new
LifecyclePrefixPredicate()
                            },
                            Status = LifecycleRuleStatus.Enabled,
                            Transitions = new List<LifecycleTransition>
                            {
                                new LifecycleTransition
                                {
                                    Days = 0,
                                    StorageClass = S3StorageClass.Glacier
                                }
                            },
                            new LifecycleRule
                            {
                                Id = "Archive and then delete rule",
                                Filter = new LifecycleFilter()
                                {
                                    LifecycleFilterPredicate = new
LifecyclePrefixPredicate()
                                },
                                Prefix = "projectdocs/"
                            },
                            Status = LifecycleRuleStatus.Enabled,
                        }
                    }
                };
                await client.PutBucketLifecycleAsync(bucketName, lifeCycleConfiguration);
            }
            catch (AmazonS3Exception e)
            {
                Console.WriteLine("Caught exception: {0}", e.Message);
            }
        }
    }
}
```

```
        Transitions = new List<LifecycleTransition>
        {
            new LifecycleTransition
            {
                Days = 30,
                StorageClass =
S3StorageClass.StandardInfrequentAccess
            },
            new LifecycleTransition
            {
                Days = 365,
                StorageClass = S3StorageClass.Glacier
            }
        },
        Expiration = new LifecycleRuleExpiration()
        {
            Days = 3650
        }
    }
};

// Add the configuration to the bucket.
await AddExampleLifecycleConfigAsync(client, lifeCycleConfiguration);

// Retrieve an existing configuration.
lifeCycleConfiguration = await RetrieveLifecycleConfigAsync(client);

// Add a new rule.
lifeCycleConfiguration.Rules.Add(new LifecycleRule
{
    Id = "NewRule",
    Filter = new LifecycleFilter()
    {
        LifecycleFilterPredicate = new LifecyclePrefixPredicate()
        {
            Prefix = "YearlyDocuments/"
        }
    },
    Expiration = new LifecycleRuleExpiration()
    {
        Days = 3650
    }
});

// Add the configuration to the bucket.
await AddExampleLifecycleConfigAsync(client, lifeCycleConfiguration);

// Verify that there are now three rules.
lifeCycleConfiguration = await RetrieveLifecycleConfigAsync(client);
Console.WriteLine("Expected # of rules=3; found:{0}",
lifeCycleConfiguration.Rules.Count);

// Delete the configuration.
await RemoveLifecycleConfigAsync(client);

// Retrieve a nonexistent configuration.
lifeCycleConfiguration = await RetrieveLifecycleConfigAsync(client);

}

catch (AmazonS3Exception e)
{
    Console.WriteLine("Error encountered ***. Message:'{0}' when writing an
object", e.Message);
}
catch (Exception e)
```

```
        {
            Console.WriteLine("Unknown encountered on server. Message:{0}' when
writing an object", e.Message);
        }
    }

    static async Task AddExampleLifecycleConfigAsync(IAmazonS3 client,
LifecycleConfiguration configuration)
{
    PutLifecycleConfigurationRequest request = new
PutLifecycleConfigurationRequest
{
    BucketName = bucketName,
    Configuration = configuration
};
var response = await client.PutLifecycleConfigurationAsync(request);
}

static async Task<LifecycleConfiguration>
RetrieveLifecycleConfigAsync(IAmazonS3 client)
{
    GetLifecycleConfigurationRequest request = new
GetLifecycleConfigurationRequest
{
    BucketName = bucketName
};
var response = await client.GetLifecycleConfigurationAsync(request);
var configuration = response.Configuration;
return configuration;
}

static async Task RemoveLifecycleConfigAsync(IAmazonS3 client)
{
    DeleteLifecycleConfigurationRequest request = new
DeleteLifecycleConfigurationRequest
{
    BucketName = bucketName
};
await client.DeleteLifecycleConfigurationAsync(request);
}
}
```

Ruby

You can use the AWS SDK for Ruby to manage S3 Lifecycle configuration on a bucket by using the class [AWS::S3::BucketLifecycleConfiguration](#). For more information about using the AWS SDK for Ruby with Amazon S3, see [Using the AWS SDK for Ruby - Version 3 \(p. 1194\)](#). For more information about managing lifecycle configuration, see [Managing your storage lifecycle \(p. 701\)](#).

Using the REST API

The following sections in the *Amazon Simple Storage Service API Reference* describe the REST API related to the S3 Lifecycle configuration.

- [PUT Bucket lifecycle](#)
- [GET Bucket lifecycle](#)
- [DELETE Bucket lifecycle](#)

Lifecycle and other bucket configurations

In addition to S3 Lifecycle configurations, you can associate other configurations with your bucket. This section explains how S3 Lifecycle configuration relates to other bucket configurations.

Lifecycle and versioning

You can add S3 Lifecycle configurations to unversioned buckets and versioning-enabled buckets. For more information, see [Using versioning in S3 buckets \(p. 638\)](#).

A versioning-enabled bucket maintains one current object version, and zero or more noncurrent object versions. You can define separate Lifecycle rules for current and noncurrent object versions.

For more information, see [Lifecycle configuration elements \(p. 721\)](#).

Lifecycle configuration on MFA-enabled buckets

Lifecycle configuration on multi-factor authentication (MFA)-enabled buckets is not supported.

Lifecycle and logging

Amazon S3 Lifecycle actions are not captured by AWS CloudTrail object level logging. CloudTrail captures API requests made to external Amazon S3 endpoints, whereas S3 Lifecycle actions are performed using internal Amazon S3 endpoints. Amazon S3 server access logs can be enabled in an S3 bucket to capture S3 Lifecycle-related actions such as object transition to another storage class and object expiration resulting in permanent deletion or logical deletion. For more information, see [the section called "Logging server access" \(p. 978\)](#).

If you have logging enabled on your bucket, Amazon S3 server access logs report the results of the following operations.

Operation log	Description
S3.EXPIRE.OBJECT	Amazon S3 permanently deletes the object due to the Lifecycle expiration action.
S3.CREATE.DELETEMARKER	Amazon S3 logically deletes the current version and adds a delete marker in a Versioning enabled bucket.
S3.TRANSITION_SIA.OBJECT	Amazon S3 transitions the object to the S3 Standard-IA storage class.
S3.TRANSITION_ZIA.OBJECT	Amazon S3 transitions the object to the S3 One Zone-IA storage class.
S3.TRANSITION_INT.OBJECT	Amazon S3 transitions the object to the S3 Intelligent-Tiering storage class.
S3.TRANSITION_GIR.OBJECT	Amazon S3 initiates the transition of object to the S3 Glacier Instant Retrieval storage class.
S3.TRANSITION.OBJECT	Amazon S3 initiates the transition of object to the S3 Glacier Flexible Retrieval storage class.
S3.TRANSITION_GDA.OBJECT	Amazon S3 initiates the transition of object to the S3 Glacier Deep Archive storage class.

Operation log	Description
S3.DELETE.UPLOAD	Amazon S3 aborts incomplete multipart upload.

Note

Amazon S3 server access log records are generally delivered on a best-effort basis and cannot be used for complete accounting of all Amazon S3 requests.

More info

- [Lifecycle configuration elements \(p. 721\)](#)
- [Transitioning to the S3 Glacier Flexible Retrieval and S3 Glacier Deep Archive storage classes \(object archival\) \(p. 705\)](#)
- [Setting lifecycle configuration on a bucket \(p. 708\)](#)

Configuring Lifecycle event notifications

You can set up an Amazon S3 event notification to receive notice when Amazon S3 deletes an object or transitions it to another Amazon S3 storage class following an S3 Lifecycle rule.

By using the *LifecycleExpiration* event types you can receive notifications whenever Amazon S3 deletes an object based on your S3 Lifecycle configuration. The `s3:LifecycleExpiration:Delete` event type notifies you when an object in an unversioned bucket is deleted. It also notifies you when an object version is permanently deleted by an S3 Lifecycle configuration. The `s3:LifecycleExpiration:DeleteMarkerCreated` event type notifies you when S3 Lifecycle creates a delete marker when a current version of an object in versioned bucket is deleted. For more information, see [Delete object version](#).

By using the `s3:LifecycleTransition` event type, you can receive notification when an object is transitioned from one Amazon S3 storage class to another by an S3 Lifecycle configuration.

Amazon S3 can publish event notifications to an Amazon Simple Notification Service (Amazon SNS) topic, an Amazon Simple Queue Service (Amazon SQS) queue, or an AWS Lambda function. For more information, see [Amazon S3 Event Notifications \(p. 1017\)](#).

For instructions on how to configure Amazon S3 Event Notifications, see [Enabling event notifications](#).

The following is an example of a message Amazon S3 sends to publish an `s3:LifecycleExpiration:Delete` event. For more information, see [Event message structure](#).

```
{
  "Records": [
    {
      "eventVersion": "2.3",
      "eventSource": "aws:s3",
      "awsRegion": "us-west-2",
      "eventTime": "1970-01-01T00:00:00.000Z",
      "eventName": "LifecycleExpiration:Delete",
      "userIdentity": {
        "principalId": "s3.amazonaws.com"
      },
      "requestParameters": {
        "sourceIPAddress": "s3.amazonaws.com"
      },
      "responseElements": {
        "x-amz-request-id": "C3D13FE58DE4C810",
        "x-amz-id-2": "A9B1C2D3E4F5G6H7I8J9K0L1M2N3O4P5Q6R7S8T9U0V1W2X3Y4Z5"
      }
    }
  ]
}
```

```

        "x-amz-id-2":"FMyUVURIY8/IgAtTv8xRjskZQpcIZ9KG4V5Wp6S7S/JRWeUWerMUE5JgHvANOjpD"
    },
    "s3":{
        "s3SchemaVersion":"1.0",
        "configurationId":"testConfigRule",
        "bucket":{
            "name":"mybucket",
            "ownerIdentity":{
                "principalId":"A3NL1KOZZKExample"
            },
            "arn":"arn:aws:s3:::mybucket"
        },
        "object":{
            "key":"expiration/delete",
            "sequencer":"0055AED6DCD90281E5",
        }
    }
}
]
}
}

```

Messages that Amazon S3 sends to publish an *s3:LifecycleTransition* event also includes the following information.

```

"lifecycleEventData":{
    "transitionEventData": {
        "destinationStorageClass": the destination storage class for the object
    }
}

```

Lifecycle configuration elements

Topics

- [ID element \(p. 722\)](#)
- [Status element \(p. 722\)](#)
- [Filter element \(p. 722\)](#)
- [Elements to describe lifecycle actions \(p. 725\)](#)

You specify an S3 Lifecycle configuration as XML, consisting of one or more Lifecycle rules.

```

<LifecycleConfiguration>
    <Rule>
        ...
    </Rule>
    <Rule>
        ...
    </Rule>
</LifecycleConfiguration>

```

Each rule consists of the following:

- Rule metadata that include a rule ID, and status indicating whether the rule is enabled or disabled. If a rule is disabled, Amazon S3 doesn't perform any actions specified in the rule.
- Filter identifying objects to which the rule applies. You can specify a filter by using object size, object key prefix, one or more object tags, or a combination of filters.
- One or more transition or expiration actions with a date or a time period in the object's lifetime when you want Amazon S3 to perform the specified action.

The following sections describe the XML elements in an S3 Lifecycle configuration. For example configurations, see [Examples of S3 Lifecycle configuration \(p. 728\)](#).

ID element

An S3 Lifecycle configuration can have up to 1,000 rules. This limit is not adjustable. The <ID> element uniquely identifies a rule. ID length is limited to 255 characters.

Status element

The <Status> element value can be either Enabled or Disabled. If a rule is disabled, Amazon S3 doesn't perform any of the actions defined in the rule.

Filter element

A Lifecycle rule can apply to all or a subset of objects in a bucket based on the <Filter> element that you specify in the Lifecycle rule.

You can filter objects by key prefix, object tags, or a combination of both (in which case Amazon S3 uses a logical AND to combine the filters). Consider the following examples:

- **Specifying a filter using key prefixes** – This example shows an S3 Lifecycle rule that applies to a subset of objects based on the key name prefix (logs/). For example, the Lifecycle rule applies to objects logs/mylog.txt, logs/temp1.txt, and logs/test.txt. The rule does not apply to the object example.jpg.

```
<LifecycleConfiguration>
  <Rule>
    <Filter>
      <Prefix>logs/</Prefix>
    </Filter>
    transition/expiration actions.
    ...
  </Rule>
  ...
</LifecycleConfiguration>
```

If you want to apply a Lifecycle action to a subset of objects based on different key name prefixes, specify separate rules. In each rule, specify a prefix-based filter. For example, to describe a Lifecycle action for objects with key prefixes projectA/ and projectB/, you specify two rules as shown following.

```
<LifecycleConfiguration>
  <Rule>
    <Filter>
      <Prefix>projectA/</Prefix>
    </Filter>
    transition/expiration actions.
    ...
  </Rule>

  <Rule>
    <Filter>
      <Prefix>projectB/</Prefix>
    </Filter>
    transition/expiration actions.
    ...
  </Rule>
</LifecycleConfiguration>
```

For more information about object keys, see [Creating object key names \(p. 150\)](#).

- **Specifying a filter based on object tags** – In the following example, the Lifecycle rule specifies a filter based on a tag (`key`) and value (`value`). The rule then applies only to a subset of objects with the specific tag.

```
<LifecycleConfiguration>
    <Rule>
        <Filter>
            <Tag>
                <Key>key</Key>
                <Value>value</Value>
            </Tag>
        </Filter>
        transition/expiration actions.
        ...
    </Rule>
</LifecycleConfiguration>
```

You can specify a filter based on multiple tags. You must wrap the tags in the `<And>` element shown in the following example. The rule directs Amazon S3 to perform lifecycle actions on objects with two tags (with the specific tag key and value).

```
<LifecycleConfiguration>
    <Rule>
        <Filter>
            <And>
                <Tag>
                    <Key>key1</Key>
                    <Value>value1</Value>
                </Tag>
                <Tag>
                    <Key>key2</Key>
                    <Value>value2</Value>
                </Tag>
                ...
            </And>
        </Filter>
        transition/expiration actions.
    </Rule>
</Lifecycle>
```

The Lifecycle rule applies to objects that have both of the tags specified. Amazon S3 performs a logical AND. Note the following:

- Each tag must match both key and value exactly.
- The rule applies to a subset of objects that has all the tags specified in the rule. If an object has additional tags specified, the rule will still apply.

Note

When you specify multiple tags in a filter, each tag key must be unique.

- **Specifying a filter based on both prefix and one or more tags** – In a Lifecycle rule, you can specify a filter based on both the key prefix and one or more tags. Again, you must wrap all of these in the `<And>` element as shown following.

```
<LifecycleConfiguration>
    <Rule>
        <Filter>
            <And>
                <Prefix>key-prefix</Prefix>
```

```

<Tag>
  <Key>key1</Key>
  <Value>value1</Value>
</Tag>
<Tag>
  <Key>key2</Key>
  <Value>value2</Value>
</Tag>
...
</And>
</Filter>
<Status>Enabled</Status>
  transition/expiration actions.
</Rule>
</LifecycleConfiguration>

```

Amazon S3 combines these filters using a logical AND. That is, the rule applies to subset of objects with a specific key prefix and specific tags. A filter can have only one prefix, and zero or more tags.

- You can specify an **empty filter**, in which case the rule applies to all objects in the bucket.

```

<LifecycleConfiguration>
  <Rule>
    <Filter>
    </Filter>
    <Status>Enabled</Status>
    transition/expiration actions.
  </Rule>
</LifecycleConfiguration>

```

- To filter a rule by **object size**, you can specify a minimum size (`ObjectSizeGreater Than`) or a maximum size (`ObjectSizeLess Than`), or you can specify a range of object sizes.

Object size values are in bytes. Maximum filter size is 5TB. Some storage classes have minimum object size limitations, for more information, see [Comparing the Amazon S3 storage classes \(p. 692\)](#).

```

<LifecycleConfiguration>
  <Rule>
    <Filter>
      <ObjectSizeGreater Than>500</ObjectSizeGreater Than>
    </Filter>
    <Status>Enabled</Status>
    transition/expiration actions.
  </Rule>
</LifecycleConfiguration>

```

If you're specifying an object size range, the `ObjectSizeGreater Than` integer must be less than the `ObjectSizeLess Than` value. When using more than one filter, you must wrap the filters in an `<And>` element. The following example shows how to specify objects in a range between 500 and 64000 bytes.

```

<LifecycleConfiguration>
  <Rule>
    <Filter>
      <And>
        <Prefix>key-prefix</Prefix>
        <ObjectSizeGreater Than>500</ObjectSizeGreater Than>
        <ObjectSizeLess Than>64000</ObjectSizeLess Than>
      </And>
    </Filter>
    <Status>Enabled</Status>
    transition/expiration actions.
  </Rule>
</LifecycleConfiguration>

```

```
</Rule>  
</LifecycleConfiguration>
```

Elements to describe lifecycle actions

You can direct Amazon S3 to perform specific actions in an object's lifetime by specifying one or more of the following predefined actions in an S3 Lifecycle rule. The effect of these actions depends on the versioning state of your bucket.

- **Transition** action element – You specify the `Transition` action to transition objects from one storage class to another. For more information about transitioning objects, see [Supported transitions and related constraints \(p. 703\)](#). When a specified date or time period in the object's lifetime is reached, Amazon S3 performs the transition.

For a versioned bucket (versioning-enabled or versioning-suspended bucket), the `Transition` action applies to the current object version. To manage noncurrent versions, Amazon S3 defines the `NoncurrentVersionTransition` action (described later in this topic).

- **Expiration** action element – The `Expiration` action expires objects identified in the rule and applies to eligible objects in any of the Amazon S3 storage classes. For more information about storage classes, see [Using Amazon S3 storage classes \(p. 688\)](#). Amazon S3 makes all expired objects unavailable. Whether the objects are permanently removed depends on the versioning state of the bucket.

Important

Object expiration Lifecycle policies do not remove incomplete multipart uploads. To remove incomplete multipart uploads, you must use the `AbortIncompleteMultipartUpload` Lifecycle configuration action that is described later in this section.

- **Non-versioned bucket** – The `Expiration` action results in Amazon S3 permanently removing the object.
- **Versioned bucket** – For a versioned bucket (that is, versioning-enabled or versioning-suspended), there are several considerations that guide how Amazon S3 handles the `Expiration` action. For more information, see [Using versioning in S3 buckets \(p. 638\)](#). Regardless of the versioning state, the following applies:
 - The `Expiration` action applies only to the current version (it has no impact on noncurrent object versions).
 - Amazon S3 doesn't take any action if there are one or more object versions and the delete marker is the current version.
 - If the current object version is the only object version and it is also a delete marker (also referred as an *expired object delete marker*, where all object versions are deleted and you only have a delete marker remaining), Amazon S3 removes the expired object delete marker. You can also use the expiration action to direct Amazon S3 to remove any expired object delete markers. For an example, see [Example 7: Removing expired object delete markers \(p. 736\)](#).

Also consider the following when setting up Amazon S3 to manage expiration:

- **Versioning-enabled bucket**

If the current object version is not a delete marker, Amazon S3 adds a delete marker with a unique version ID. This makes the current version noncurrent, and the delete marker the current version.

- **Versioning-suspended bucket**

In a versioning-suspended bucket, the expiration action causes Amazon S3 to create a delete marker with null as the version ID. This delete marker replaces any object version with a null version ID in the version hierarchy, which effectively deletes the object.

In addition, Amazon S3 provides the following actions that you can use to manage noncurrent object versions in a versioned bucket (that is, versioning-enabled and versioning-suspended buckets).

- **NoncurrentVersionTransition** action element – Use this action to specify when to have Amazon S3 transition objects to the specified storage class. You can base this expiration on a certain number of days since the objects become noncurrent. In addition to the number of days, you can also provide a maximum number of noncurrent versions to retain. You also need to provide a `Filter` element in order to specify the maximum number of noncurrent versions. If you do not specify a `Filter` element, Amazon S3 will generate an `InvalidRequest` error when you provide a maximum number of noncurrent versions.

For more information about transitioning objects, see [Supported transitions and related constraints \(p. 703\)](#). For details about how Amazon S3 calculates the date when you specify the number of days in the `NoncurrentVersionTransition` action, see [Lifecycle rules: Based on an object's age \(p. 727\)](#).

- **NoncurrentVersionExpiration** action element – Use this action to specify when Amazon S3 permanently removes noncurrent objects. These deleted objects cannot be recovered. You can base this expiration on a certain number of days since the objects become noncurrent. In addition to the number of days, you can also provide a maximum number of noncurrent versions to retain. You also need to provide a `Filter` element in order to specify the maximum number of noncurrent versions. If you do not specify a `Filter` element, Amazon S3 will generate an `InvalidRequest` error when you provide a maximum number of noncurrent versions.

Delayed removal of noncurrent objects can be helpful when you need to correct any accidental deletes or overwrites. For example, you can configure an expiration rule to delete noncurrent versions five days after they become noncurrent. For example, suppose that on 1/1/2014 10:30 AM UTC, you create an object called `photo.gif` (version ID 111111). On 1/2/2014 11:30 AM UTC, you accidentally delete `photo.gif` (version ID 111111), which creates a delete marker with a new version ID (such as version ID 4857693). You now have five days to recover the original version of `photo.gif` (version ID 111111) before the deletion is permanent. On 1/8/2014 00:00 UTC, the Lifecycle rule for expiration executes and permanently deletes `photo.gif` (version ID 111111), five days after it became a noncurrent version.

For details about how Amazon S3 calculates the date when you specify the number of days in an `NoncurrentVersionExpiration` action, see [Lifecycle rules: Based on an object's age \(p. 727\)](#).

Important

Object expiration Lifecycle policies do not remove incomplete multipart uploads. To remove incomplete multipart uploads, you must use the **AbortIncompleteMultipartUpload** Lifecycle configuration action that is described later in this section.

In addition to the transition and expiration actions, you can use the following Lifecycle configuration action to direct Amazon S3 to stop incomplete multipart uploads.

- **AbortIncompleteMultipartUpload** action element – Use this element to set a maximum time (in days) that you want to allow multipart uploads to remain in progress. If the applicable multipart uploads (determined by the key name `prefix` specified in the Lifecycle rule) are not successfully completed within the predefined time period, Amazon S3 stops the incomplete multipart uploads. For more information, see [Aborting a multipart upload \(p. 193\)](#).

Note

You cannot specify this Lifecycle action in a rule that specifies a filter based on object tags.

- **ExpiredObjectDeleteMarker** action element – In a versioning-enabled bucket, a delete marker with zero noncurrent versions is referred to as the expired object delete marker. You can use this Lifecycle action to direct S3 to remove the expired object delete markers. For an example, see [Example 7: Removing expired object delete markers \(p. 736\)](#).

Note

You cannot specify this Lifecycle action in a rule that specifies a filter based on object tags.

How Amazon S3 calculates how long an object has been noncurrent

In a versioning-enabled bucket, you can have multiple versions of an object. There is always one current version, and zero or more noncurrent versions. Each time you upload an object, the current version is retained as the noncurrent version and the newly added version, the successor, becomes the current version. To determine the number of days an object is noncurrent, Amazon S3 looks at when its successor was created. Amazon S3 uses the number of days since its successor was created as the number of days an object is noncurrent.

Restoring previous versions of an object when using S3 Lifecycle configurations

As explained in detail in the topic [Restoring previous versions \(p. 657\)](#), you can use either of the following two methods to retrieve previous versions of an object:

1. By copying a noncurrent version of the object into the same bucket. The copied object becomes the current version of that object, and all object versions are preserved.
2. By permanently deleting the current version of the object. When you delete the current object version, you, in effect, turn the noncurrent version into the current version of that object.

When you are using S3 Lifecycle configuration rules with versioning-enabled buckets, we recommend as a best practice that you use the first method.

S3 Lifecycle operates under an eventually consistent model. A current version that you permanently deleted might not disappear until the changes propagate (Amazon S3 might be unaware of this deletion). In the meantime, the lifecycle rule that you configured to expire noncurrent objects might permanently remove noncurrent objects, including the one that you want to restore. So, copying the old version, as recommended in the first method, is the safer alternative.

Lifecycle rules: Based on an object's age

You can specify a time period, in number of days from the creation (or modification) of the objects, when Amazon S3 can take the action.

When you specify the number of days in the `Transition` and `Expiration` actions in an S3 Lifecycle configuration, note the following:

- It is the number of days since object creation when the action will occur.
- Amazon S3 calculates the time by adding the number of days specified in the rule to the object creation time and rounding the resulting time to the next day midnight UTC. For example, if an object was created at 1/15/2014 10:30 AM UTC and you specify 3 days in a transition rule, then the transition date of the object would be calculated as 1/19/2014 00:00 UTC.

Note

Amazon S3 maintains only the last modified date for each object. For example, the Amazon S3 console shows the **Last Modified** date in the object **Properties** pane. When you initially create a new object, this date reflects the date the object is created. If you replace the object, the date changes accordingly. So when we use the term *creation date*, it is synonymous with the term *last modified date*.

When specifying the number of days in the `NoncurrentVersionTransition` and `NoncurrentVersionExpiration` actions in a Lifecycle configuration, note the following:

- It is the number of days from when the version of the object becomes noncurrent (that is, when the object is overwritten or deleted) that Amazon S3 will perform the action on the specified object or objects.
- Amazon S3 calculates the time by adding the number of days specified in the rule to the time when the new successor version of the object is created and rounding the resulting time to the next day midnight UTC. For example, in your bucket, suppose that you have a current version of an object that was created at 1/1/2014 10:30 AM UTC. If the new version of the object that replaces the current version is created at 1/15/2014 10:30 AM UTC, and you specify 3 days in a transition rule, the transition date of the object is calculated as 1/19/2014 00:00 UTC.

Lifecycle rules: Based on a specific date

When specifying an action in an S3 Lifecycle rule, you can specify a date when you want Amazon S3 to take the action. When the specific date arrives, Amazon S3 applies the action to all qualified objects (based on the filter criteria).

If you specify an S3 Lifecycle action with a date that is in the past, all qualified objects become immediately eligible for that Lifecycle action.

Important

The date-based action is not a one-time action. Amazon S3 continues to apply the date-based action even after the date has passed, as long as the rule status is `Enabled`.

For example, suppose that you specify a date-based `Expiration` action to delete all objects (assume no filter specified in the rule). On the specified date, Amazon S3 expires all the objects in the bucket. S3 also continues to expire any new objects you create in the bucket. To stop the Lifecycle action, you must remove the action from the Lifecycle configuration, disable the rule, or delete the rule from the Lifecycle configuration.

The date value must conform to the ISO 8601 format. The time is always midnight UTC.

Note

You can't create the date-based Lifecycle rules using the Amazon S3 console, but you can view, disable, or delete such rules.

Examples of S3 Lifecycle configuration

This section provides examples of S3 Lifecycle configuration. Each example shows how you can specify the XML in each of the example scenarios.

Topics

- [Example 1: Specifying a filter \(p. 729\)](#)
- [Example 2: Disabling a Lifecycle rule \(p. 730\)](#)
- [Example 3: Tiering down storage class over an object's lifetime \(p. 731\)](#)
- [Example 4: Specifying multiple rules \(p. 732\)](#)
- [Example 5: Overlapping filters, conflicting lifecycle actions, and what Amazon S3 does with nonversioned buckets \(p. 732\)](#)
- [Example 6: Specifying a lifecycle rule for a versioning-enabled bucket \(p. 735\)](#)
- [Example 7: Removing expired object delete markers \(p. 736\)](#)
- [Example 8: Lifecycle configuration to abort multipart uploads \(p. 738\)](#)
- [Example 9: Lifecycle configuration using size-based rules \(p. 738\)](#)

Example 1: Specifying a filter

Each S3 Lifecycle rule includes a filter that you can use to identify a subset of objects in your bucket to which the S3 Lifecycle rule applies. The following S3 Lifecycle configurations show examples of how you can specify a filter.

- In this S3 Lifecycle configuration rule, the filter specifies a key prefix (`tax/`). Therefore, the rule applies to objects with the key name prefix `tax/`, such as `tax/doc1.txt` and `tax/doc2.txt`.

The rule specifies two actions that direct Amazon S3 to do the following:

- Transition objects to the S3 Glacier Flexible Retrieval storage class 365 days (one year) after creation.
- Delete objects (the `Expiration` action) 3,650 days (10 years) after creation.

```
<LifecycleConfiguration>
  <Rule>
    <ID>Transition and Expiration Rule</ID>
    <Filter>
      <Prefix>tax/</Prefix>
    </Filter>
    <Status>Enabled</Status>
    <Transition>
      <Days>365</Days>
      <StorageClass>S3 Glacier Flexible Retrieval</StorageClass>
    </Transition>
    <Expiration>
      <Days>3650</Days>
    </Expiration>
  </Rule>
</LifecycleConfiguration>
```

Instead of specifying object age in terms of days after creation, you can specify a date for each action. However, you can't use both `Date` and `Days` in the same rule.

- If you want the S3 Lifecycle rule to apply to all objects in the bucket, specify an empty prefix. In the following configuration, the rule specifies a `Transition` action that directs Amazon S3 to transition objects to the S3 Glacier Flexible Retrieval storage class 0 days after creation. This rule means that the objects are eligible for archival to Amazon S3 Glacier at midnight UTC following creation. For more information about lifecycle constraints, see [Constraints \(p. 704\)](#).

```
<LifecycleConfiguration>
  <Rule>
    <ID>Archive all object same-day upon creation</ID>
    <Filter>
      <Prefix></Prefix>
    </Filter>
    <Status>Enabled</Status>
    <Transition>
      <Days>0</Days>
      <StorageClass>S3 Glacier Flexible Retrieval</StorageClass>
    </Transition>
  </Rule>
</LifecycleConfiguration>
```

- You can specify zero or one key name prefix and zero or more object tags in a filter. The following example code applies the S3 Lifecycle rule to a subset of objects with the `tax/` key prefix and to objects that have two tags with specific key and value. When you specify more than one filter, you must include the `<And>` element as shown (Amazon S3 applies a logical AND to combine the specified filter conditions).

...

```
<Filter>
  <And>
    <Prefix>tax/</Prefix>
    <Tag>
      <Key>key1</Key>
      <Value>value1</Value>
    </Tag>
    <Tag>
      <Key>key2</Key>
      <Value>value2</Value>
    </Tag>
  </And>
</Filter>
...
```

- You can filter objects based only on tags. For example, the following S3 Lifecycle rule applies to objects that have the two specified tags (it does not specify any prefix).

```
...
<Filter>
  <And>
    <Tag>
      <Key>key1</Key>
      <Value>value1</Value>
    </Tag>
    <Tag>
      <Key>key2</Key>
      <Value>value2</Value>
    </Tag>
  </And>
</Filter>
...
```

Important

When you have multiple rules in an S3 Lifecycle configuration, an object can become eligible for multiple S3 Lifecycle actions. In such cases, Amazon S3 follows these general rules:

- Permanent deletion takes precedence over transition.
- Transition takes precedence over creation of delete markers.
- When an object is eligible for both a S3 Glacier Flexible Retrieval and S3 Standard-IA (or S3 One Zone-IA) transition, Amazon S3 chooses the S3 Glacier Flexible Retrieval transition.

For examples, see [Example 5: Overlapping filters, conflicting lifecycle actions, and what Amazon S3 does with nonversioned buckets \(p. 732\)](#).

Example 2: Disabling a Lifecycle rule

You can temporarily disable a S3 Lifecycle rule. The following S3 Lifecycle configuration specifies two rules:

- Rule 1 directs Amazon S3 to transition objects with the logs/ prefix to the S3 Glacier Flexible Retrieval storage class soon after creation.
- Rule 2 directs Amazon S3 to transition objects with the documents/ prefix to the S3 Glacier Flexible Retrieval storage class soon after creation.

In the policy, Rule 1 is enabled and Rule 2 is disabled. Amazon S3 ignores disabled rules.

```
<LifecycleConfiguration>
  <Rule>
    <ID>Rule1</ID>
    <Filter>
      <Prefix>logs/</Prefix>
    </Filter>
    <Status>Enabled</Status>
    <Transition>
      <Days>0</Days>
      <StorageClass>S3 Glacier Flexible Retrieval</StorageClass>
    </Transition>
  </Rule>
  <Rule>
    <ID>Rule2</ID>
    <Filter>
      <Prefix>documents/</Prefix>
    </Filter>
    <Status>Disabled</Status>
    <Transition>
      <Days>0</Days>
      <StorageClass>S3 Glacier Flexible Retrieval</StorageClass>
    </Transition>
  </Rule>
</LifecycleConfiguration>
```

Example 3: Tiering down storage class over an object's lifetime

In this example, you use S3 Lifecycle configuration to tier down the storage class of objects over their lifetime. Tiering down can help reduce storage costs. For more information about pricing, see [Amazon S3 pricing](#).

The following S3 Lifecycle configuration specifies a rule that applies to objects with the key name prefix logs/. The rule specifies the following actions:

- Two transition actions:
 - Transition objects to the S3 Standard-IA storage class 30 days after creation.
 - Transition objects to the S3 Glacier Flexible Retrieval storage class 90 days after creation.
- One expiration action that directs Amazon S3 to delete objects a year after creation.

```
<LifecycleConfiguration>
  <Rule>
    <ID>example-id</ID>
    <Filter>
      <Prefix>logs/</Prefix>
    </Filter>
    <Status>Enabled</Status>
    <Transition>
      <Days>30</Days>
      <StorageClass>STANDARD_IA</StorageClass>
    </Transition>
    <Transition>
      <Days>90</Days>
      <StorageClass>GLACIER</StorageClass>
    </Transition>
    <Expiration>
      <Days>365</Days>
    </Expiration>
  </Rule>
</LifecycleConfiguration>
```

```
</LifecycleConfiguration>
```

Note

You can use one rule to describe all S3 Lifecycle actions if all actions apply to the same set of objects (identified by the filter). Otherwise, you can add multiple rules with each specifying a different filter.

Example 4: Specifying multiple rules

You can specify multiple rules if you want different S3 Lifecycle actions of different objects. The following S3 Lifecycle configuration has two rules:

- Rule 1 applies to objects with the key name prefix `classA/`. It directs Amazon S3 to transition objects to the S3 Glacier Flexible Retrieval storage class one year after creation and expire these objects 10 years after creation.
- Rule 2 applies to objects with key name prefix `classB/`. It directs Amazon S3 to transition objects to the S3 Standard-IA storage class 90 days after creation and delete them one year after creation.

```
<LifecycleConfiguration>
  <Rule>
    <ID>ClassADocRule</ID>
    <Filter>
      <Prefix>classA/</Prefix>
    </Filter>
    <Status>Enabled</Status>
    <Transition>
      <Days>365</Days>
      <StorageClass>GLACIER</StorageClass>
    </Transition>
    <Expiration>
      <Days>3650</Days>
    </Expiration>
  </Rule>
  <Rule>
    <ID>ClassBDocRule</ID>
    <Filter>
      <Prefix>classB/</Prefix>
    </Filter>
    <Status>Enabled</Status>
    <Transition>
      <Days>90</Days>
      <StorageClass>STANDARD_IA</StorageClass>
    </Transition>
    <Expiration>
      <Days>365</Days>
    </Expiration>
  </Rule>
</LifecycleConfiguration>
```

Example 5: Overlapping filters, conflicting lifecycle actions, and what Amazon S3 does with nonversioned buckets

You might specify an S3 Lifecycle configuration in which you specify overlapping prefixes, or actions.

Generally, S3 Lifecycle optimizes for cost. For example, if two expiration policies overlap, the shorter expiration policy is honored so that data is not stored for longer than expected. Likewise, if two transition policies overlap, S3 Lifecycle transitions your objects to the lower-cost storage class.

In both cases, S3 Lifecycle tries to choose the path that is least expensive for you. An exception to this general rule is with the S3 Intelligent-Tiering storage class. S3 Intelligent-Tiering is favored by S3 Lifecycle over any storage class, aside from the S3 Glacier Flexible Retrieval and S3 Glacier Deep Archive storage classes.

The following examples show how Amazon S3 resolves potential conflicts.

Example 1: Overlapping prefixes (no conflict)

The following example configuration has two rules that specify overlapping prefixes as follows:

- The first rule specifies an empty filter, indicating all objects in the bucket.
- The second rule specifies a key name prefix (logs/), indicating only a subset of objects.

Rule 1 requests Amazon S3 to delete all objects one year after creation. Rule 2 requests Amazon S3 to transition a subset of objects to the S3 Standard-IA storage class 30 days after creation.

```
<LifecycleConfiguration>
  <Rule>
    <ID>Rule 1</ID>
    <Filter>
    </Filter>
    <Status>Enabled</Status>
    <Expiration>
      <Days>365</Days>
    </Expiration>
  </Rule>
  <Rule>
    <ID>Rule 2</ID>
    <Filter>
      <Prefix>logs/</Prefix>
    </Filter>
    <Status>Enabled</Status>
    <Transition>
      <StorageClass>STANDARD_IA<StorageClass>
      <Days>30</Days>
    </Transition>
  </Rule>
</LifecycleConfiguration>
```

Since there is no conflict in this case, Amazon S3 will transition the objects with the logs/ prefix to the S3 Standard-IA storage class 30 days after creation. When any object reaches one year after creation, it will be deleted.

Example 2: Conflicting lifecycle actions

In this example configuration, there are two rules that direct Amazon S3 to perform two different actions on the same set of objects at the same time in the objects' lifetime:

- Both rules specify the same key name prefix, so both rules apply to the same set of objects.
- Both rules specify the same 365 days after object creation when the rules apply.
- One rule directs Amazon S3 to transition objects to the S3 Standard-IA storage class and another rule wants Amazon S3 to expire the objects at the same time.

```
<LifecycleConfiguration>
  <Rule>
    <ID>Rule 1</ID>
```

```
<Filter>
  <Prefix>logs/</Prefix>
</Filter>
<Status>Enabled</Status>
<Expiration>
  <Days>365</Days>
</Expiration>
</Rule>
<Rule>
  <ID>Rule 2</ID>
<Filter>
  <Prefix>logs/</Prefix>
</Filter>
<Status>Enabled</Status>
<Transition>
  <StorageClass>STANDARD_IA<StorageClass>
  <Days>365</Days>
</Transition>
</Rule>
</LifecycleConfiguration>
```

In this case, because you want objects to expire (to be removed), there is no point in changing the storage class, so Amazon S3 chooses the expiration action on these objects.

Example 3: Overlapping prefixes resulting in conflicting lifecycle actions

In this example, the configuration has two rules, which specify overlapping prefixes as follows:

- Rule 1 specifies an empty prefix (indicating all objects).
- Rule 2 specifies a key name prefix (logs/) that identifies a subset of all objects.

For the subset of objects with the logs/ key name prefix, S3 Lifecycle actions in both rules apply. One rule directs Amazon S3 to transition objects 10 days after creation, and another rule directs Amazon S3 to transition objects 365 days after creation.

```
<LifecycleConfiguration>
<Rule>
  <ID>Rule 1</ID>
<Filter>
  <Prefix></Prefix>
</Filter>
<Status>Enabled</Status>
<Transition>
  <StorageClass>STANDARD_IA<StorageClass>
  <Days>10</Days>
</Transition>
</Rule>
<Rule>
  <ID>Rule 2</ID>
<Filter>
  <Prefix>logs/</Prefix>
</Filter>
<Status>Enabled</Status>
<Transition>
  <StorageClass>STANDARD_IA<StorageClass>
  <Days>365</Days>
</Transition>
</Rule>
</LifecycleConfiguration>
```

In this case, Amazon S3 chooses to transition them 10 days after creation.

Example 4: Tag-based filtering and resulting conflicting lifecycle actions

Suppose that you have the following S3 Lifecycle policy that has two rules, each specifying a tag filter:

- Rule 1 specifies a tag-based filter (`tag1/value1`). This rule directs Amazon S3 to transition objects to the S3 Glacier Flexible Retrieval storage class 365 days after creation.
- Rule 2 specifies a tag-based filter (`tag2/value2`). This rule directs Amazon S3 to expire objects 14 days after creation.

The S3 Lifecycle configuration is shown in following example.

```
<LifecycleConfiguration>
  <Rule>
    <ID>Rule 1</ID>
    <Filter>
      <Tag>
        <Key>tag1</Key>
        <Value>value1</Value>
      </Tag>
    </Filter>
    <Status>Enabled</Status>
    <Transition>
      <StorageClass>GLACIER<StorageClass>
      <Days>365</Days>
    </Transition>
  </Rule>
  <Rule>
    <ID>Rule 2</ID>
    <Filter>
      <Tag>
        <Key>tag2</Key>
        <Value>value2</Value>
      </Tag>
    </Filter>
    <Status>Enabled</Status>
    <Expiration>
      <Days>14</Days>
    </Expiration>
  </Rule>
</LifecycleConfiguration>
```

If an object has both tags, then Amazon S3 has to decide which rule to follow. In this case, Amazon S3 expires the object 14 days after creation. The object is removed, and therefore the transition action does not apply.

Example 6: Specifying a lifecycle rule for a versioning-enabled bucket

Suppose that you have a versioning-enabled bucket, which means that for each object, you have a current version and zero or more noncurrent versions. (For more information about S3 Versioning, see [Using versioning in S3 buckets \(p. 638\)](#).) In this example, you want to maintain one year's worth of history, and delete the noncurrent versions. S3 Lifecycle configurations supports keeping 1 to 100 versions of any object.

To save storage costs, you want to move noncurrent versions to S3 Glacier Flexible Retrieval 30 days after they become noncurrent (assuming that these noncurrent objects are cold data for which you don't need real-time access). In addition, you expect frequency of access of the current versions to diminish 90 days after creation, so you might choose to move these objects to the S3 Standard-IA storage class.

```
<LifecycleConfiguration>
  <Rule>
    <ID>sample-rule</ID>
    <Filter>
      <Prefix></Prefix>
    </Filter>
    <Status>Enabled</Status>
    <Transition>
      <Days>90</Days>
      <StorageClass>STANDARD_IA</StorageClass>
    </Transition>
    <NoncurrentVersionTransition>
      <NoncurrentDays>30</NoncurrentDays>
      <StorageClass>S3 Glacier Flexible Retrieval</StorageClass>
    </NoncurrentVersionTransition>
    <NoncurrentVersionExpiration>
      <NewerNoncurrentVersions>5</NewerNoncurrentVersions>
      <NoncurrentDays>365</NoncurrentDays>
    </NoncurrentVersionExpiration>
  </Rule>
</LifecycleConfiguration>
```

Example 7: Removing expired object delete markers

A versioning-enabled bucket has one current version and zero or more noncurrent versions for each object. When you delete an object, note the following:

- If you don't specify a version ID in your delete request, Amazon S3 adds a delete marker instead of deleting the object. The current object version becomes noncurrent, and the delete marker becomes the current version.
- If you specify a version ID in your delete request, Amazon S3 deletes the object version permanently (a delete marker is not created).
- A delete marker with zero noncurrent versions is referred to as an *expired object delete marker*.

This example shows a scenario that can create expired object delete markers in your bucket, and how you can use S3 Lifecycle configuration to direct Amazon S3 to remove the expired object delete markers.

Suppose that you write a S3 Lifecycle policy that uses the NoncurrentVersionExpiration action to remove the noncurrent versions 30 days after they become noncurrent and retains at most 10 noncurrent versions, as shown in the following example.

```
<LifecycleConfiguration>
  <Rule>
    ...
    <NoncurrentVersionExpiration>
      <NewerNoncurrentVersions>10</NewerNoncurrentVersions>
      <NoncurrentDays>30</NoncurrentDays>
    </NoncurrentVersionExpiration>
  </Rule>
</LifecycleConfiguration>
```

The NoncurrentVersionExpiration action does not apply to the current object versions. It removes only the noncurrent versions.

For current object versions, you have the following options to manage their lifetime, depending on whether the current object versions follow a well-defined lifecycle:

- **The current object versions follow a well-defined lifecycle.**

In this case, you can use an S3 Lifecycle policy with the `Expiration` action to direct Amazon S3 to remove the current versions, as shown in the following example.

```
<LifecycleConfiguration>
  <Rule>
    ...
    <Expiration>
      <Days>60</Days>
    </Expiration>
    <NoncurrentVersionExpiration>
      <NewerNoncurrentVersions>10</NewerNoncurrentVersions>
      <NoncurrentDays>30</NoncurrentDays>
    </NoncurrentVersionExpiration>
  </Rule>
</LifecycleConfiguration>
```

In this example, Amazon S3 removes current versions 60 days after they are created by adding a delete marker for each of the current object versions. This process makes the current version noncurrent, and the delete marker becomes the current version. For more information, see [Using versioning in S3 buckets \(p. 638\)](#).

Note

You cannot specify both a `Days` and an `ExpiredObjectDeleteMarker` tag on the same rule. When you specify the `Days` tag, Amazon S3 automatically performs `ExpiredObjectDeleteMarker` cleanup when the delete markers are old enough to satisfy the age criteria. To clean up delete markers as soon as they become the only version, create a separate rule with only the `ExpiredObjectDeleteMarker` tag.

The `NoncurrentVersionExpiration` action in the same S3 Lifecycle configuration removes noncurrent objects 30 days after they become noncurrent. Thus, in this example, all object versions are permanently removed 90 days after object creation. Although expired object delete markers are created during this process, Amazon S3 detects and removes the expired object delete markers for you.

- **The current object versions don't have a well-defined lifecycle.**

In this case, you might remove the objects manually when you don't need them, creating a delete marker with one or more noncurrent versions. If your S3 Lifecycle configuration with the `NoncurrentVersionExpiration` action removes all the noncurrent versions, you now have expired object delete markers.

Specifically for this scenario, S3 Lifecycle configuration provides an `Expiration` action that you can use to remove the expired object delete markers.

```
<LifecycleConfiguration>
  <Rule>
    <ID>Rule 1</ID>
    <Filter>
      <Prefix>logs/</Prefix>
    </Filter>
    <Status>Enabled</Status>
    <Expiration>
      <ExpiredObjectDeleteMarker>true</ExpiredObjectDeleteMarker>
    </Expiration>
    <NoncurrentVersionExpiration>
      <NewerNoncurrentVersions>10</NewerNoncurrentVersions>
      <NoncurrentDays>30</NoncurrentDays>
    </NoncurrentVersionExpiration>
  </Rule>
</LifecycleConfiguration>
```

By setting the `ExpiredObjectDeleteMarker` element to `true` in the `Expiration` action, you direct Amazon S3 to remove the expired object delete markers.

Note

When you use the `ExpiredObjectDeleteMarker` S3 Lifecycle action, the rule cannot specify a tag-based filter.

Example 8: Lifecycle configuration to abort multipart uploads

You can use the Amazon S3 multipart upload REST API operations to upload large objects in parts. For more information about multipart uploads, see [Uploading and copying objects using multipart upload \(p. 167\)](#).

Using S3 Lifecycle configuration, you can direct Amazon S3 to stop incomplete multipart uploads (identified by the key name prefix specified in the rule) if they aren't completed within a specified number of days after initiation. When Amazon S3 aborts a multipart upload, it deletes all the parts associated with the multipart upload. This process helps control your storage costs by ensuring that you don't have incomplete multipart uploads with parts that are stored in Amazon S3.

Note

When you use the `AbortIncompleteMultipartUpload` S3 Lifecycle action, the rule cannot specify a tag-based filter.

The following is an example S3 Lifecycle configuration that specifies a rule with the `AbortIncompleteMultipartUpload` action. This action directs Amazon S3 to stop incomplete multipart uploads seven days after initiation.

```
<LifecycleConfiguration>
  <Rule>
    <ID>sample-rule</ID>
    <Filter>
      <Prefix>SomeKeyPrefix/</Prefix>
    </Filter>
    <Status>rule-status</Status>
    <AbortIncompleteMultipartUpload>
      <DaysAfterInitiation>7</DaysAfterInitiation>
    </AbortIncompleteMultipartUpload>
  </Rule>
</LifecycleConfiguration>
```

Example 9: Lifecycle configuration using size-based rules

You can create rules that transition objects based only on their size. You can specify a minimum size (`ObjectSizeGreater Than`) or a maximum size (`ObjectSizeLess Than`), or you can specify a range of object sizes in bytes. When using more than one filter, such as a prefix and size rule, you must wrap the filters in an `<And>` element.

```
<LifecycleConfiguration>
  <Rule>
    <ID>Transition with a prefix and based on size</ID>
    <Filter>
      <And>
        <Prefix>tax/</Prefix>
        <ObjectSizeGreater Than>500</ObjectSizeGreater Than>
      </And>
    </Filter>
    <Status>Enabled</Status>
    <Transition>
      <Days>365</Days>
      <StorageClass>S3 Glacier Flexible Retrieval</StorageClass>
    </Transition>
  </Rule>
</LifecycleConfiguration>
```

```
</Rule>
</LifecycleConfiguration>
```

If you're specifying a range by using both the `ObjectSizeGreater Than` and `ObjectSizeLess Than` elements, the maximum object size must be larger than the minimum object size. When using more than one filter, you must wrap the filters in an `<And>` element. The following example shows how to specify objects in a range between 500 and 64000 bytes.

```
<LifecycleConfiguration>
  <Rule>
    ...
    <And>
      <ObjectSizeGreater Than>500</ObjectSizeGreater Than>
      <ObjectSizeLess Than>64000</ObjectSizeLess Than>
    </And>
  </Rule>
</LifecycleConfiguration>
```

Amazon S3 Inventory

Amazon S3 Inventory is one of the tools Amazon S3 provides to help manage your storage. You can use it to audit and report on the replication and encryption status of your objects for business, compliance, and regulatory needs. You can also simplify and speed up business workflows and big data jobs using Amazon S3 Inventory, which provides a scheduled alternative to the Amazon S3 synchronous `List` API operation. Amazon S3 Inventory does not use the `List` API to audit your objects and does not affect the request rate of your bucket.

Amazon S3 Inventory provides comma-separated values (CSV), [Apache optimized row columnar \(ORC\)](#) or [Apache Parquet](#) output files that list your objects and their corresponding metadata on a daily or weekly basis for an S3 bucket or a shared prefix (that is, objects that have names that begin with a common string). If weekly, a report is generated every Sunday (UTC) after the initial report. For information about Amazon S3 Inventory pricing, see [Amazon S3 pricing](#).

You can configure multiple inventory lists for a bucket. You can configure what object metadata to include in the inventory, whether to list all object versions or only current versions, where to store the inventory list file output, and whether to generate the inventory on a daily or weekly basis. You can also specify that the inventory list file be encrypted.

You can query Amazon S3 Inventory using standard SQL by using [Amazon Athena](#), Amazon Redshift Spectrum, and other tools such as [Presto](#), [Apache Hive](#), and [Apache Spark](#). You can use Athena to run queries on your inventory files. You can use it for Amazon S3 Inventory queries in all Regions where Athena is available.

Source and destination buckets

The bucket that the inventory lists the objects for is called the *source bucket*. The bucket where the inventory list file is stored is called the *destination bucket*.

Source bucket

The inventory lists the objects that are stored in the source bucket. You can get inventory lists for an entire bucket or filtered by (object key name) prefix.

The source bucket:

- Contains the objects that are listed in the inventory.
- Contains the configuration for the inventory.

Destination bucket

Amazon S3 Inventory list files are written to the destination bucket. To group all the inventory list files in a common location in the destination bucket, you can specify a destination prefix (object key name) in the inventory configuration.

The destination bucket:

- Contains the inventory file lists.
- Contains the manifest files that list all the file inventory lists that are stored in the destination bucket. For more information, see [Inventory manifest \(p. 747\)](#).
- Must have a bucket policy to give Amazon S3 permission to verify ownership of the bucket and permission to write files to the bucket.
- Must be in the same AWS Region as the source bucket.
- Can be the same as the source bucket.
- Can be owned by a different AWS account than the account that owns the source bucket.

Amazon S3 Inventory list

An inventory list file contains a list of the objects in the source bucket and metadata for each object. The inventory lists are stored in the destination bucket as a CSV file compressed with GZIP, as an Apache optimized row columnar (ORC) file compressed with ZLIB, or as an Apache Parquet file compressed with Snappy. Objects are sorted in ascending order based on the key names.

The inventory list contains a list of the objects in an S3 bucket and the following metadata for each listed object:

- **Bucket name** – The name of the bucket that the inventory is for.
- **Key name** – The object key name (or key) that uniquely identifies the object in the bucket. When using the CSV file format, the key name is URL-encoded and must be decoded before you can use it.
- **Version ID** – The object version ID. When you enable versioning on a bucket, Amazon S3 assigns a version number to objects that are added to the bucket. For more information, see [Using versioning in S3 buckets \(p. 638\)](#). (This field is not included if the list is only for the current version of objects.)
- **IsLatest** – Set to `True` if the object is the current version of the object. (This field is not included if the list is only for the current version of objects.)
- **Size** – The object size in bytes.
- **Last modified date** – The object creation date or the last modified date, whichever is the latest.
- **ETag** – The entity tag is a hash of the object. The ETag reflects changes only to the contents of an object, not its metadata. The ETag can be an MD5 digest of the object data. Whether it is depends on how the object was created and how it is encrypted.
- **Storage class** – The storage class used for storing the object. For more information, see [Using Amazon S3 storage classes \(p. 688\)](#).
- **Multipart upload flag** – Set to `True` if the object was uploaded as a multipart upload. For more information, see [Uploading and copying objects using multipart upload \(p. 167\)](#).
- **Delete marker** – Set to `True` if the object is a delete marker. For more information, see [Using versioning in S3 buckets \(p. 638\)](#). (This field is automatically added to your report if you've configured the report to include all versions of your objects.)
- **Replication status** – Set to `PENDING`, `COMPLETED`, `FAILED`, or `REPLICA`. For more information, see [Getting replication status information \(p. 820\)](#).
- **Encryption status** – Set to `SSE-S3`, `SSE-C`, `SSE-KMS`, or `NOT-SSE`. The server-side encryption status for SSE-S3, SSE-KMS, and SSE with customer-provided keys (SSE-C). A status of `NOT-SSE` means that the object is not encrypted with server-side encryption. For more information, see [Protecting data using encryption \(p. 337\)](#).

- **S3 Object Lock Retain until date** – The date until which the locked object cannot be deleted. For more information, see [Using S3 Object Lock \(p. 680\)](#).
- **S3 Object Lock Mode** – Set to Governance or Compliance for objects that are locked. For more information, see [Using S3 Object Lock \(p. 680\)](#).
- **S3 Object Lock Legal hold status** – Set to On if a legal hold has been applied to an object. Otherwise, it is set to Off. For more information, see [Using S3 Object Lock \(p. 680\)](#).
- **S3 Intelligent-Tiering access tier** – Access tier (frequent or infrequent) of the object if stored in S3 Intelligent-Tiering. For more information, see [Storage class for automatically optimizing data with changing or unknown access patterns \(p. 689\)](#).
- **S3 Bucket Key status** – Set to ENABLED or DISABLED. Indicates whether the object uses S3 Bucket Key for server-side encryption. For more information, see [Using Amazon S3 Bucket Keys \(p. 347\)](#).
- **Checksum Algorithm** – Indicates the algorithm used to create the checksum for the object.

We recommend that you create a lifecycle policy that deletes old inventory lists. For more information, see [Managing your storage lifecycle \(p. 701\)](#).

Inventory consistency

All of your objects might not appear in each inventory list. The inventory list provides eventual consistency for PUTs of both new objects and overwrites, and DELETEs. Inventory lists are a rolling snapshot of bucket items, which are eventually consistent (that is, the list might not include recently added or deleted objects).

To validate the state of the object before you take action on the object, we recommend that you perform a HEAD Object REST API request to retrieve metadata for the object, or check the object's properties in the Amazon S3 console. You can also check object metadata with the AWS CLI or the AWS SDKs. For more information, see [HEAD Object](#) in the *Amazon Simple Storage Service API Reference*.

For more information about working with Amazon S3 Inventory, see the following topics.

Topics

- [Configuring Amazon S3 Inventory \(p. 741\)](#)
- [Setting up Amazon S3 Event Notifications for inventory completion \(p. 745\)](#)
- [Locating your inventory list \(p. 746\)](#)
- [Querying Amazon S3 Inventory with Amazon Athena \(p. 749\)](#)
- [Converting empty version ID strings in Amazon S3 Inventory reports to null strings \(p. 751\)](#)

Configuring Amazon S3 Inventory

Amazon S3 Inventory provides a flat file list of your objects and metadata, which is a scheduled alternative to the Amazon S3 synchronous List API operation. Amazon S3 Inventory provides comma-separated values (CSV) or [Apache optimized row columnar \(ORC\)](#) or [Apache Parquet \(Parquet\)](#) output files that list your objects and their corresponding metadata on a daily or weekly basis for an S3 bucket or for objects that share a prefix (objects that have names that begin with the same string). For more information, see [Amazon S3 Inventory \(p. 739\)](#).

This section describes how to configure an inventory, including details about the inventory source and destination buckets.

Topics

- [Overview \(p. 742\)](#)
- [Creating a destination bucket policy \(p. 742\)](#)

- [Granting Amazon S3 permission to use your AWS KMS key for encryption \(p. 743\)](#)
- [Configuring inventory using the S3 console \(p. 744\)](#)

Overview

Amazon S3 Inventory helps you manage your storage by creating lists of the objects in an S3 bucket on a defined schedule. You can configure multiple inventory lists for a bucket. The inventory lists are published to CSV, ORC, or Parquet files in a destination bucket.

The easiest way to set up an inventory is by using the AWS Management Console, but you can also use the REST API, AWS CLI, or AWS SDKs. The console performs the first step of the following procedure for you: adding a bucket policy to the destination bucket.

To set up Amazon S3 Inventory for an S3 bucket

1. Add a bucket policy for the destination bucket.

You must create a bucket policy on the destination bucket to grant permissions to Amazon S3 to write objects to the bucket in the defined location. For an example policy, see [Granting permissions for Amazon S3 Inventory and Amazon S3 analytics \(p. 497\)](#).

2. Configure an inventory to list the objects in a source bucket and publish the list to a destination bucket.

When you configure an inventory list for a source bucket, you specify the destination bucket where you want the list to be stored, and whether you want to generate the list daily or weekly. You can also configure what object metadata to include and whether to list all object versions or only current versions.

You can specify that the inventory list file be encrypted by using an Amazon S3 managed key (SSE-S3) or an AWS Key Management Service (AWS KMS) customer managed key. For more information about SSE-S3 and SSE-KMS, see [Protecting data using server-side encryption \(p. 338\)](#). If you plan to use SSE-KMS encryption, see Step 3.

- For information about how to use the console to configure an inventory list, see [Configuring Amazon S3 Inventory \(p. 741\)](#).
- To use the Amazon S3 API to configure an inventory list, use the [PUT Bucket inventory configuration](#) REST API or the equivalent from the AWS CLI or AWS SDKs.

3. To encrypt the inventory list file with SSE-KMS, grant Amazon S3 permission to use the AWS KMS key.

You can configure encryption for the inventory list file by using the AWS Management Console, REST API, AWS CLI, or AWS SDKs. Whichever way you choose, you must grant Amazon S3 permission to use the customer managed key to encrypt the inventory file. You grant Amazon S3 permission by modifying the key policy for the customer managed key that you want to use to encrypt the inventory file. For more information, see the next section, [Granting Amazon S3 permission to use your AWS KMS key for encryption \(p. 743\)](#).

4. To configure inventory, see [Configuring inventory using the S3 console \(p. 744\)](#).

If you use encryption for cross-account operations of Amazon S3 inventory configuration in the destination bucket, you should use fully qualified KMS key ARN. For more information, see [Using encryption for cross-account operations \(p. 132\)](#) and [ServerSideEncryptionByDefault](#).

Creating a destination bucket policy

If you create the inventory configuration through the S3 console, Amazon S3 automatically creates a bucket policy on the destination bucket that grants Amazon S3 write permission. If you create the

inventory configuration through the AWS CLI, SDKs, or the REST API, you must manually add a bucket policy on the destination bucket. For more information, see [Granting permissions for Amazon S3 Inventory and Amazon S3 analytics \(p. 497\)](#). The Amazon S3 Inventory destination bucket policy allows Amazon S3 to write data for the inventory reports to the bucket.

If an error occurs when you try to create the bucket policy, you are given instructions on how to fix it. For example, if you choose a destination bucket in another AWS account and don't have permissions to read and write to the bucket policy, you see an error message.

In this case, the destination bucket owner must add the displayed bucket policy to the destination bucket. If the policy is not added to the destination bucket, you won't get an inventory report because Amazon S3 doesn't have permission to write to the destination bucket. If the source bucket is owned by a different account than that of the current user, the correct account ID of the source bucket must be substituted in the policy.

Granting Amazon S3 permission to use your AWS KMS key for encryption

To grant Amazon S3 permission to encrypt using a customer managed AWS Key Management Service (AWS KMS)key, you must use a key policy. To update your key policy so that you can use customer managed key, follow the steps below.

To grant permissions to encrypt using your KMS key

1. Using the AWS account that owns the customer managed key, sign into the AWS Management Console.
2. Open the AWS KMS console at <https://console.aws.amazon.com/kms>.
3. To change the AWS Region, use the Region selector in the upper-right corner of the page.
4. In the left navigation pane, choose **Customer managed keys**.
5. Under **Customer managed keys**, choose the customer managed key that you want to use to encrypt the inventory file.
6. Under **Key policy**, choose **Switch to policy view**.
7. To update the key policy, choose **Edit**.
8. Under **Edit key policy**, add the following key policy to the existing key policy.

```
{  
    "Sid": "Allow Amazon S3 use of the KMS key",  
    "Effect": "Allow",  
    "Principal": {  
        "Service": "s3.amazonaws.com"  
    },  
    "Action": [  
        "kms:GenerateDataKey"  
    ],  
    "Resource": "*",  
    "Condition":{  
        "StringEquals":{  
            "aws:SourceAccount":"source-account-id"  
        },  
        "ArnLike":{  
            "aws:SourceARN": "arn:aws:s3:::source-bucket-name"  
        }  
    }  
}
```

9. Choose **Save changes**.

For more information about creating customer managed keys and using key policies, see the following links in the *AWS Key Management Service Developer Guide*:

- [Getting Started](#)
- [Using Key Policies in AWS KMS](#)

Configuring inventory using the S3 console

Use these instructions to configure inventory using the S3 console.

Note

It may take up to 48 hours to deliver the first report.

1. Sign in to the AWS Management Console and open the Amazon S3 console at <https://console.aws.amazon.com/s3/>.
2. In the **Buckets** list, choose the name of the bucket for which you want to configure Amazon S3 Inventory.
3. Choose **Management**.
4. Under **Inventory configurations**, choose **Create inventory configuration**.
5. In **Inventory configuration name**, enter a name.
6. Set the **Inventory scope**:
 - Enter an optional prefix.
 - Choose object versions: **Current versions only** or **Include all versions**.
7. Under **Report details**, choose the location of the AWS account that you want to save the reports to: **This account** or **A different account**.
8. Under **Destination**, choose the destination bucket where you want reports to be saved.

The destination bucket must be in the same AWS Region as the bucket for which you are setting up the inventory. The destination bucket can be in a different AWS account. Under the **Destination** bucket field, you see the **Destination bucket permission** that is added to the destination bucket policy to allow Amazon S3 to place data in that bucket. For more information, see [Creating a destination bucket policy \(p. 742\)](#).

9. Under **Frequency**, choose how often the report will be generated: **Daily** or **Weekly**.
10. Choose the **Output format** for the report:
 - **CSV**
 - **Apache ORC**
 - **Apache Parquet**
11. Under **Status**, choose **Enable** or **Disable**.
12. To use server-side encryption, under **Server-side encryption**, follow these steps:
 - a. Choose **Enable**.
 - b. Under **Encryption key type**, choose **Amazon S3 key (SSE-S3)** or **AWS Key Management Service key (SSE-KMS)**.

Amazon S3 server-side encryption uses 256-bit Advanced Encryption Standard (AES-256). For more information, see [Protecting data using server-side encryption with Amazon S3-managed encryption keys \(SSE-S3\) \(p. 355\)](#). For more information about SSE-KMS, see [Protecting data using server-side encryption with AWS Key Management Service \(SSE-KMS\) \(p. 338\)](#).

- c. To use an AWS KMS key, choose one of the following:
 - **Choose from your AWS KMS keys**, and choose your **KMS key**.

- Enter **AWS KMS key ARN**, and enter your AWS KMS key ARN.

Note

To encrypt the inventory list file with SSE-KMS, you must grant Amazon S3 permission to use the AWS KMS key. Therefore, you can only use a [customer managed key](#) and not the [AWS managed key](#) (aws/s3). For instructions, see [Granting Amazon S3 permission to use your AWS KMS key for encryption \(p. 743\)](#).

13. For **Additional fields**, select one or more of the following to add to the inventory report:

- **Size** – The object size in bytes.
- **Last modified date** – The object creation date or the last modified date, whichever is the latest.
- **Storage class** – The storage class used for storing the object.
- **ETag** – The entity tag is a hash of the object. The ETag reflects changes only to the contents of an object, and not its metadata. The ETag may or may not be an MD5 digest of the object data. Whether it is depends on how the object was created and how it is encrypted.
- **Multipart upload** – Specifies that the object was uploaded as a multipart upload. For more information, see [Uploading and copying objects using multipart upload \(p. 167\)](#).
- **Replication status** – The replication status of the object. For more information, see [Using the S3 console \(p. 773\)](#).
- **Encryption status** – The server-side encryption used to encrypt the object. For more information, see [Protecting data using server-side encryption \(p. 338\)](#).
- **S3 Object Lock configurations** – The Object Lock status of the object, including the following settings:
 - **Retention mode** – The level of protection applied to the object, either *Governance* or *Compliance*.
 - **Retain until date** – The date until which the locked object cannot be deleted.
 - **Legal hold status** – The legal hold status of the locked object.

For information about S3 Object Lock, see [How S3 Object Lock works \(p. 681\)](#).

- **Intelligent-Tiering access tier** – Indicates the access tier (frequent or infrequent) of the object if it was stored in Intelligent-Tiering. For more information, see [Storage class for automatically optimizing data with changing or unknown access patterns \(p. 689\)](#).
- **S3 Bucket Key status** – Indicates whether a bucket-level key generated by AWS KMS applies to the object. For more information, see [Reducing the cost of SSE-KMS with Amazon S3 Bucket Keys \(p. 347\)](#).
- **Checksum Algorithm** – Indicates the algorithm used to create the checksum for the object.

For more information about the contents of an inventory report, see [Amazon S3 Inventory list \(p. 740\)](#).

14. Choose **Create**.

Setting up Amazon S3 Event Notifications for inventory completion

You can set up an Amazon S3 event notification to receive notice when the manifest checksum file is created, which indicates that an inventory list has been added to the destination bucket. The manifest is an up-to-date list of all the inventory lists at the destination location.

Amazon S3 can publish events to an Amazon Simple Notification Service (Amazon SNS) topic, an Amazon Simple Queue Service (Amazon SQS) queue, or an AWS Lambda function. For more information, see [Amazon S3 Event Notifications \(p. 1017\)](#).

The following notification configuration defines that all `manifest.checksum` files newly added to the destination bucket are processed by the AWS Lambda `cloud-function-list-write`.

```
<NotificationConfiguration>
  <QueueConfiguration>
    <Id>1</Id>
    <Filter>
      <S3Key>
        <FilterRule>
          <Name>prefix</Name>
          <Value>destination-prefix/source-bucket</Value>
        </FilterRule>
        <FilterRule>
          <Name>suffix</Name>
          <Value>checksum</Value>
        </FilterRule>
      </S3Key>
    </Filter>
    <Cloudcode>arn:aws:lambda:us-west-2:22223334444:cloud-function-list-write</Cloudcode>
    <Event>s3:ObjectCreated:*</Event>
  </QueueConfiguration>
</NotificationConfiguration>
```

For more information, see [Using AWS Lambda with Amazon S3](#) in the *AWS Lambda Developer Guide*.

Locating your inventory list

When an inventory list is published, the manifest files are published to the following location in the destination bucket.

```
destination-prefix/source-bucket/config-ID/YYYY-MM-DDTHH-MMZ/manifest.json
destination-prefix/source-bucket/config-ID/YYYY-MM-DDTHH-MMZ/manifest.checksum
destination-prefix/source-bucket/config-ID/hive/dt=YYYY-MM-DD-HH-MM/symlink.txt
```

- **destination-prefix** is the (object key name) prefix set in the inventory configuration, which can be used to group all the inventory list files in a common location within the destination bucket.
- **source-bucket** is the source bucket that the inventory list is for. It is added to prevent collisions when multiple inventory reports from different source buckets are sent to the same destination bucket.
- **config-ID** is added to prevent collisions with multiple inventory reports from the same source bucket that are sent to the same destination bucket. The **config-ID** comes from the inventory report configuration, and is the name for the report that is defined on setup.
- **YYYY-MM-DDTHH-MMZ** is the timestamp that consists of the start time and the date when the inventory report generation begins scanning the bucket; for example, 2016-11-06T21-32Z.
- **manifest.json** is the manifest file.
- **manifest.checksum** is the MD5 of the content of the `manifest.json` file.
- **symlink.txt** is the Apache Hive-compatible manifest file.

The inventory lists are published daily or weekly to the following location in the destination bucket.

```
destination-prefix/source-bucket/config-ID/data/example-file-name.csv.gz
...
destination-prefix/source-bucket/config-ID/data/example-file-name-1.csv.gz
```

- **destination-prefix** is the (object key name) prefix set in the inventory configuration. It can be used to group all the inventory list files in a common location in the destination bucket.
- **source-bucket** is the source bucket that the inventory list is for. It is added to prevent collisions when multiple inventory reports from different source buckets are sent to the same destination bucket.
- **example-file-name.csv.gz** is one of the CSV inventory files. ORC inventory names end with the file name extension .orc, and Parquet inventory names end with the file name extension .parquet.

Inventory manifest

The manifest files `manifest.json` and `symlink.txt` describe where the inventory files are located. Whenever a new inventory list is delivered, it is accompanied by a new set of manifest files. These files may overwrite each other. In versioning-enabled buckets, Amazon S3 creates new versions of the manifest files.

Each manifest contained in the `manifest.json` file provides metadata and other basic information about an inventory. This information includes the following:

- Source bucket name
- Destination bucket name
- Version of the inventory
- Creation timestamp in the epoch date format that consists of the start time and the date when the inventory report generation begins scanning the bucket
- Format and schema of the inventory files
- Actual list of the inventory files that are in the destination bucket

Whenever a `manifest.json` file is written, it is accompanied by a `manifest.checksum` file that is the MD5 of the content of `manifest.json` file.

Example Inventory manifest in a `manifest.json` file

The following examples show an inventory manifest in a `manifest.json` file for a CSV, ORC, and Parquet-formatted inventories.

CSV

The following is an example of a manifest in a `manifest.json` file for a CSV-formatted inventory.

```
{  
    "sourceBucket": "example-source-bucket",  
    "destinationBucket": "arn:aws:s3:::example-inventory-destination-bucket",  
    "version": "2016-11-30",  
    "creationTimestamp" : "1514944800000",  
    "fileFormat": "CSV",  
    "fileSchema": "Bucket, Key, VersionId, IsLatest, IsDeleteMarker,  
    Size, LastModifiedDate, ETag, StorageClass, IsMultipartUploaded,  
    ReplicationStatus, EncryptionStatus, ObjectLockRetainUntilDate, ObjectLockMode,  
    ObjectLockLegalHoldStatus, IntelligentTieringAccessTier, BucketKeyStatus,  
    ChecksumAlgorithm",
```

```
    "files": [
        {
            "key": "Inventory/example-source-bucket/2016-11-06T21-32Z/
files/939c6d46-85a9-4ba8-87bd-9db705a579ce.csv.gz",
            "size": 2147483647,
            "MD5checksum": "f11166069f1990abeb9c97ace9cdfabc"
        }
    ]
}
```

ORC

The following is an example of a manifest in a `manifest.json` file for an ORC-formatted inventory.

```
{
    "sourceBucket": "example-source-bucket",
    "destinationBucket": "arn:aws:s3:::example-destination-bucket",
    "version": "2016-11-30",
    "creationTimestamp": "1514944800000",
    "fileFormat": "ORC",
    "fileSchema": "struct<bucket:string,key:string,version_id:string,is_latest:boolean,is_delete_marker:boolean,size:int64>",

    "files": [
        {
            "key": "inventory/example-source-bucket/data/
d794c570-95bb-4271-9128-26023c8b4900.orc",
            "size": 56291,
            "MD5checksum": "5925f4e78e1695c2d020b9f6eexample"
        }
    ]
}
```

Parquet

The following is an example of a manifest in a `manifest.json` file for a Parquet-formatted inventory.

```
{
    "sourceBucket": "example-source-bucket",
    "destinationBucket": "arn:aws:s3:::example-destination-bucket",
    "version": "2016-11-30",
    "creationTimestamp": "1514944800000",
    "fileFormat": "Parquet",
    "fileSchema": "message s3.inventory { required binary bucket (UTF8);
required binary key (UTF8); optional binary version_id (UTF8); optional boolean
is_latest; optional boolean is_delete_marker; optional int64 size; optional int64
last_modified_date (TIMESTAMP_MILLIS); optional binary e_tag (UTF8); optional
binary storage_class (UTF8); optional boolean is.multipart_uploaded; optional binary
replication_status (UTF8); optional binary encryption_status (UTF8); optional int64
object_lock_retain_until_date (TIMESTAMP_MILLIS); optional binary object_lock_mode
(UTF8); optional binary object_lock_legal_hold_status (UTF8); optional binary
intelligent_tiering_access_tier (UTF8); optional binary bucket_key_status (UTF8);
optional binary checksum_algorithm (UTF8); }",
    "files": [
        {
            "key": "inventory/example-source-bucket/data/
d754c470-85bb-4255-9218-47023c8b4910.parquet",
            "size": 56291,
            "MD5checksum": "5825f2e18e1695c2d030b9f6eexample"
        }
    ]
}
```

The `symlink.txt` file is an Apache Hive-compatible manifest file that allows Hive to automatically discover inventory files and their associated data files. The Hive-compatible manifest works with the Hive-compatible services Athena and Amazon Redshift Spectrum. It also works with Hive-compatible applications, including [Presto](#), [Apache Hive](#), [Apache Spark](#), and many others.

Important

The `symlink.txt` Apache Hive-compatible manifest file does not currently work with AWS Glue.

Reading `symlink.txt` with [Apache Hive](#) and [Apache Spark](#) is not supported for ORC and Parquet-formatted inventory files.

Querying Amazon S3 Inventory with Amazon Athena

You can query Amazon S3 Inventory using standard SQL by using Amazon Athena in all Regions where Athena is available. To check for AWS Region availability, see the [AWS Region Table](#).

Athena can query Amazon S3 Inventory files in ORC, Parquet, or CSV format. When you use Athena to query inventory, we recommend that you use ORC-formatted or Parquet-formatted inventory files. ORC and Parquet formats provide faster query performance and lower query costs. ORC and Parquet are self-describing type-aware columnar file formats designed for [Apache Hadoop](#). The columnar format lets the reader read, decompress, and process only the columns that are required for the current query. The ORC and Parquet formats for Amazon S3 Inventory are available in all AWS Regions.

To get started using Athena to query Amazon S3 Inventory

1. Create an Athena table. For information about creating a table, see [Creating Tables in Amazon Athena](#) in the *Amazon Athena User Guide*.

The following sample query includes all optional fields in an ORC-formatted inventory report. Drop any optional field that you did not choose for your inventory so that the query corresponds to the fields chosen for your inventory. Also, you must use your bucket name and location to your inventory destination path. Replace the following bucket name and inventory location as appropriate for your configuration: `s3://destination-prefix/DOC-EXAMPLE-BUCKET/config-ID/hive/`. You should also replace the initial date under `projection.dt.range` to the first day with data.

```
CREATE EXTERNAL TABLE your_table_name(  
    bucket string,  
    key string,  
    version_id string,  
    is_latest boolean,  
    is_delete_marker boolean,  
    size bigint,  
    last_modified_date bigint,  
    e_tag string,  
    storage_class string,  
    is.multipart_uploaded boolean,  
    replication_status string,  
    encryption_status string,  
    object_lock_retain_until_date bigint,  
    object_lock_mode string,  
    object_lock_legal_hold_status string,  
    intelligent_tiering_access_tier string,  
    bucket_key_status string,  
    checksum_algorithm string  
) PARTITIONED BY (  
    dt string  
)  
ROW FORMAT SERDE 'org.apache.hadoop.hive.ql.io.orc.OrcSerde'  
STORED AS INPUTFORMAT 'org.apache.hadoop.hive.ql.io.SymlinkTextInputFormat'  
OUTPUTFORMAT 'org.apache.hadoop.hive.ql.io.IgnoreKeyTextOutputFormat'
```

```

LOCATION 's3://destination-prefix/source-bucket/config-ID/hive/'
TBLPROPERTIES (
    "projection.enabled" = "true",
    "projection.dt.type" = "date",
    "projection.dt.format" = "yyyy-MM-dd-HH-mm",
    "projection.dt.range" = "2022-01-01-00-00,NOW",
    "projection.dt.interval" = "1",
    "projection.dt.interval.unit" = "DAYS"
);

```

When using Athena to query a Parquet-formatted inventory report, use the following Parquet SerDe in place of the ORC SerDe in the ROW FORMAT SERDE statement.

```
ROW FORMAT SERDE 'org.apache.hadoop.hive.ql.io.parquet.serde.ParquetHiveSerDe'
```

When using Athena to query a CSV-formatted inventory report, use the following template.

```

CREATE EXTERNAL TABLE your_table_name(
    bucket string,
    key string,
    version_id string,
    is_latest boolean,
    is_delete_marker boolean,
    size string,
    last_modified_date string,
    e_tag string,
    storage_class string,
    is.multipart_uploaded boolean,
    replication_status string,
    encryption_status string,
    object_lock_retain_until_date string,
    object_lock_mode string,
    object_lock_legal_hold_status string,
    intelligent_tiering_access_tier string,
    bucket_key_status string,
    checksum_algorithm string
) PARTITIONED BY (
    dt string
)
ROW FORMAT SERDE 'org.apache.hadoop.hive.serde2.OpenCSVSerde'
    STORED AS INPUTFORMAT 'org.apache.hadoop.hive.ql.io.SymlinkTextInputFormat'
    OUTPUTFORMAT 'org.apache.hadoop.hive.ql.io.IgnoreKeyTextOutputFormat'
    LOCATION 's3://destination-prefix/source-bucket/config-ID/hive/'
TBLPROPERTIES (
    "projection.enabled" = "true",
    "projection.dt.type" = "date",
    "projection.dt.format" = "yyyy-MM-dd-HH-mm",
    "projection.dt.range" = "2022-01-01-00-00,NOW",
    "projection.dt.interval" = "1",
    "projection.dt.interval.unit" = "DAYS"
);

```

2. After performing this step, you can run ad hoc queries on your inventory, as shown in the following examples.

```

# Get list of latest inventory report dates available
SELECT DISTINCT dt FROM your_table_name ORDER BY 1 DESC limit 10;

# Get encryption status for a provided report date.
SELECT encryption_status, count(*) FROM your_table_name WHERE dt = 'YYYY-MM-DD-HH-MM'
GROUP BY encryption_status;

```

```
# Get encryption status for report dates in the provided range.  
SELECT dt, encryption_status, count(*) FROM your_table_name  
WHERE dt > 'YYYY-MM-DD-HH-MM' AND dt < 'YYYY-MM-DD-HH-MM' GROUP BY dt,  
encryption_status;
```

For more information about using Athena, see [Amazon Athena User Guide](#).

The following are the REST operations used for Amazon S3 Inventory.

- [DELETE Bucket Inventory](#)
- [GET Bucket Inventory](#)
- [List Bucket Inventory](#)
- [PUT Bucket Inventory](#)

Converting empty version ID strings in Amazon S3 Inventory reports to null strings

Note

The following procedure applies only to Amazon S3 Inventory reports that include all versions, and only if the "all versions" reports are used as manifests for S3 Batch Operations on buckets that have S3 Versioning enabled. You are not required to convert strings for S3 Inventory reports that specify the current version only.

You can use S3 Inventory reports as manifests for S3 Batch Operations. However, when S3 Versioning is enabled on a bucket, S3 Inventory reports that include all versions mark any null-versioned objects with empty strings in the version ID field. When an Inventory Report includes all object version IDs, Batch Operations recognizes null strings as version IDs, but not empty strings.

When an S3 Batch Operations job uses an "all versions" S3 Inventory report as a manifest, it fails all tasks on objects that have an empty string in the version ID field. To convert empty strings in the version ID field of the S3 Inventory report to null strings for Batch Operations, use the following procedure.

Update an Amazon S3 Inventory report for use with Batch Operations

1. Sign in to the AWS Management Console and open the Amazon S3 console at <https://console.aws.amazon.com/s3/>.
2. Navigate to your S3 Inventory report. The inventory report is located in the destination bucket that you specified while configuring your inventory report. For more information about locating inventory reports, see [Locating your inventory list \(p. 746\)](#).
 - a. Choose the destination bucket.
 - b. Choose the folder. The folder is named after the original source bucket.
 - c. Choose the folder named after the inventory configuration.
 - d. Select the check box next to the folder named **hive**. At the top of the page, choose **Copy S3 URI** to copy the S3 URI for the folder.
3. Open the Amazon Athena console at <https://console.aws.amazon.com/athena/>.
4. In the query editor, choose **Settings**, then choose **Manage**. On the **Manage settings** page, for **Location of query result**, choose an S3 bucket to store your query results in.
5. In the query editor, create an Athena table to hold the data in the inventory report using the following command. Replace **table_name** with a name of your choosing, and in the **LOCATION** clause, insert the S3 URI that you copied earlier. Then choose **Run** to run the query.

```
CREATE EXTERNAL TABLE table_name(bucket string, key string,
version_id string) PARTITIONED BY (dt string)ROW FORMAT SERDE
'org.apache.hadoop.hive.serde2.OpenCSVSerde'STORED AS INPUTFORMAT
'org.apache.hadoop.hive.ql.io.SymlinkTextInputFormat' OUTPUTFORMAT
'org.apache.hadoop.hive.ql.io.IgnoreKeyTextOutputFormat' LOCATION 'Copied S3 URI';
```

6. To clear the query editor, choose **Clear**. Then load the inventory report into the table using the following command. Replace *table_name* with the one that you chose in the prior step. Then choose **Run** to run the query.

```
MSCK REPAIR TABLE table_name;
```

7. To clear the query editor, choose **Clear**. Run the following SELECT query to retrieve all entries in the original inventory report and replace any empty version IDs with null strings. Replace *table_name* with the one that you chose earlier, and replace **YYYY-MM-DD-HH-MM** in the WHERE clause with the date of the inventory report that you want this tool to run on. Then choose **Run** to run the query.

```
SELECT bucket as Bucket, key as Key, CASE WHEN version_id = '' THEN 'null' ELSE
version_id END as VersionId FROM table_name WHERE dt = 'YYYY-MM-DD-HH-MM';
```

8. Return to the Amazon S3 console (<https://console.aws.amazon.com/s3/>), and navigate to the S3 bucket that you chose for **Location of query result** earlier. Inside, there should be a series of folders ending with the date.

For example, you should see something like **s3://DOC-EXAMPLE-BUCKET/query-result-location/Unsaved/2021/10/07/**. You should see .csv files containing the results of the SELECT query that you ran.

Choose the CSV file with the latest modified date. Download this file to your local machine for the next step.

9. The generated CSV file contains a header row. To use this CSV file as input for an S3 Batch Operations job, you must remove the header row, because Batch Operations doesn't support header rows on CSV manifests.

To remove the header row, you can run one of the following commands on the file. Replace *file.csv* with the name of your CSV file.

For macOS and Linux machines, run the tail command in a Terminal window.

```
tail -n +2 file.csv > tmp.csv && mv tmp.csv file.csv
```

For Windows machines, run the following script in a Windows PowerShell window. Replace **File-location** with the path to your file, and *file.csv* with the file name.

```
$ins = New-Object System.IO.StreamReader File-location\file.csv
$out = New-Object System.IO.StreamWriter File-location\temp.csv
try {
    $skip = 0
    while ( !$ins.EndOfStream ) {
        $line = $ins.ReadLine();
        if ( $skip -ne 0 ) {
            $out.WriteLine($line);
        } else {
            $skip = 1
        }
    }
} finally {
    $out.Close();
```

```
$ins.Close();  
}  
Move-Item File-location\temp.csv File-location\file.csv -Force
```

10. After removing the header row from the CSV file, you are ready to use it as a manifest in an S3 Batch Operations job. Upload the CSV file to an S3 bucket or location of your choosing, and then create a Batch Operations job using the CSV file as the manifest.

For more information about creating a Batch Operations job, see [Creating an S3 Batch Operations job \(p. 889\)](#).

Replicating objects

Replication enables automatic, asynchronous copying of objects across Amazon S3 buckets. Buckets that are configured for object replication can be owned by the same AWS account or by different accounts. You can replicate objects to a single destination bucket or to multiple destination buckets. The destination buckets can be in different AWS Regions or within the same Region as the source bucket.

To automatically replicate new objects as they are written to the bucket use live replication, such as Same-Region Replication (SRR) or Cross-Region Replication (CRR). To replicate existing objects to a different bucket on demand, use S3 Batch Replication. For more information about replicating existing objects, see [When to use S3 Batch Replication \(p. 755\)](#).

To enable SRR or CRR, you add a replication configuration to your source bucket. The minimum configuration must provide the following:

- The destination bucket or buckets where you want Amazon S3 to replicate objects
- An AWS Identity and Access Management (IAM) role that Amazon S3 can assume to replicate objects on your behalf

Additional configuration options are available. For more information, see [Additional replication configurations \(p. 805\)](#).

Topics

- [Why use replication \(p. 753\)](#)
- [When to use Cross-Region Replication \(p. 754\)](#)
- [When to use Same-Region Replication \(p. 754\)](#)
- [When to use S3 Batch Replication \(p. 755\)](#)
- [Requirements for replication \(p. 755\)](#)
- [What does Amazon S3 replicate? \(p. 756\)](#)
- [Setting up replication \(p. 758\)](#)
- [Replicating existing objects with S3 Batch Replication \(p. 798\)](#)
- [Additional replication configurations \(p. 805\)](#)
- [Getting replication status information \(p. 820\)](#)
- [Troubleshooting replication \(p. 822\)](#)
- [Additional considerations \(p. 823\)](#)

Why use replication

Replication can help you do the following:

- **Replicate objects while retaining metadata** – You can use replication to make copies of your objects that retain all metadata, such as the original object creation times and version IDs. This capability is important if you must ensure that your replica is identical to the source object.
- **Replicate objects into different storage classes** – You can use replication to directly put objects into S3 Glacier Flexible Retrieval, S3 Glacier Deep Archive, or another storage class in the destination buckets. You can also replicate your data to the same storage class and use lifecycle policies on the destination buckets to move your objects to a colder storage class as they age.
- **Maintain object copies under different ownership** – Regardless of who owns the source object, you can tell Amazon S3 to change replica ownership to the AWS account that owns the destination bucket. This is referred to as the *owner override* option. You can use this option to restrict access to object replicas.
- **Keep objects stored over multiple AWS Regions** – To ensure geographic differences in where your data is kept, you can set multiple destination buckets across different AWS Regions. This feature might help you meet certain compliance requirements.
- **Replicate objects within 15 minutes** – To replicate your data in the same AWS Region or across different Regions within a predictable time frame, you can use S3 Replication Time Control (S3 RTC). S3 RTC replicates 99.99 percent of new objects stored in Amazon S3 within 15 minutes (backed by a service-level agreement). For more information, see [the section called "Using S3 Replication Time Control" \(p. 807\)](#).
- **Sync buckets, replicate existing objects, and replicate previously failed or replicated objects** – To sync buckets and replicate existing objects, use Batch Replication as an on-demand replication action. For more information about when to use Batch Replication, see [When to use S3 Batch Replication \(p. 755\)](#).

Note

S3 RTC does not apply to Batch Replication. Batch Replication is an on-demand replication job, and can be tracked with S3 Batch Operations. For more information, see [Tracking job status and completion reports \(p. 922\)](#).

When to use Cross-Region Replication

S3 Cross-Region Replication (CRR) is used to copy objects across Amazon S3 buckets in different AWS Regions. CRR can help you do the following:

- **Meet compliance requirements** – Although Amazon S3 stores your data across multiple geographically distant Availability Zones by default, compliance requirements might dictate that you store data at even greater distances. To satisfy these requirements, use Cross-Region Replication to replicate data between distant AWS Regions.
- **Minimize latency** – If your customers are in two geographic locations, you can minimize latency in accessing objects by maintaining object copies in AWS Regions that are geographically closer to your users.
- **Increase operational efficiency** – If you have compute clusters in two different AWS Regions that analyze the same set of objects, you might choose to maintain object copies in those Regions.

When to use Same-Region Replication

Same-Region Replication (SRR) is used to copy objects across Amazon S3 buckets in the same AWS Region. SRR can help you do the following:

- **Aggregate logs into a single bucket** – If you store logs in multiple buckets or across multiple accounts, you can easily replicate logs into a single, in-Region bucket. Doing so allows for simpler processing of logs in a single location.

- **Configure live replication between production and test accounts** – If you or your customers have production and test accounts that use the same data, you can replicate objects between those multiple accounts, while maintaining object metadata.
- **Abide by data sovereignty laws** – You might be required to store multiple copies of your data in separate AWS accounts within a certain Region. Same-Region Replication can help you automatically replicate critical data when compliance regulations don't allow the data to leave your country.

When to use S3 Batch Replication

Batch Replication replicates existing objects to different buckets as an on-demand option. Unlike live replication, these jobs can be executed as needed. Batch Replication can help you do the following:

- **Replicate existing objects** – You can use Batch Replication to replicate objects that were added to the bucket before Same-Region Replication or Cross-Region Replication were configured.
- **Replicate objects that previously failed to replicate** – You can filter a Batch Replication job to attempt to replicate objects with a replication status of **FAILED**.
- **Replicate objects that were already replicated** – You might be required to store multiple copies of your data in separate AWS accounts or AWS Regions. Batch Replication can replicate existing objects to newly added destinations.
- **Replicate replicas of objects that were created from a replication rule** – Replication configurations create replicas of objects in destination buckets. Replicas of objects can be replicated only with Batch Replication.

Requirements for replication

Replication requires the following:

- The source bucket owner must have the source and destination AWS Regions enabled for their account. The destination bucket owner must have the destination Region enabled for their account.

For more information about enabling or disabling an AWS Region, see [Managing AWS Regions](#) in the [AWS General Reference](#).

- Both source and destination buckets must have versioning enabled. For more information about versioning, see [Using versioning in S3 buckets \(p. 638\)](#).
- Amazon S3 must have permissions to replicate objects from the source bucket to the destination bucket or buckets on your behalf. For more information about these permissions, see [Setting up permissions \(p. 769\)](#).
- If the owner of the source bucket doesn't own the object in the bucket, the object owner must grant the bucket owner **READ** and **READ_ACP** permissions with the object access control list (ACL). For more information, see [Access control list \(ACL\) overview \(p. 554\)](#).
- If the source bucket has S3 Object Lock enabled, the destination buckets must also have S3 Object Lock enabled.

For more information, see [Using S3 Object Lock \(p. 680\)](#). To enable replication on a bucket that has Object Lock enabled, contact [AWS Support](#).

For more information, see [Setting up replication \(p. 758\)](#).

If you are setting the replication configuration in a *cross-account scenario*, where source and destination buckets are owned by different AWS accounts, the following additional requirement applies:

- The owner of the destination buckets must grant the owner of the source bucket permissions to replicate objects with a bucket policy. For more information, see [Granting permissions when the source and destination buckets are owned by different AWS accounts \(p. 771\)](#).
- The destination buckets cannot be configured as Requester Pays buckets. For more information, see [Using Requester Pays buckets for storage transfers and usage \(p. 144\)](#).

What does Amazon S3 replicate?

Amazon S3 replicates only specific items in buckets that are configured for replication.

Topics

- [What is replicated with replication configurations? \(p. 756\)](#)
- [What isn't replicated with replication configurations? \(p. 757\)](#)

What is replicated with replication configurations?

By default, Amazon S3 replicates the following:

- Objects created after you add a replication configuration.
- Unencrypted objects.
- Objects encrypted at rest under an Amazon S3 managed key (SSE-S3) or a KMS key stored in AWS Key Management Service (SSE-KMS).

To replicate objects encrypted with a KMS key stored in AWS KMS, you must explicitly enable the option. The replicated copy of the object is encrypted using the same type of server-side encryption that was used for the source object. For more information about server-side encryption, see [Protecting data using server-side encryption \(p. 338\)](#).

- Object metadata from the source objects to the replicas. For information about replicating metadata from the replicas to the source objects, see [Replicating metadata changes with Amazon S3 replica modification sync \(p. 811\)](#).
- Only objects in the source bucket for which the bucket owner has permissions to read objects and access control lists (ACLs).

For more information about resource ownership, see [Amazon S3 bucket and object ownership \(p. 396\)](#).

- Object ACL updates, unless you direct Amazon S3 to change the replica ownership when source and destination buckets aren't owned by the same accounts.

For more information, see [Changing the replica owner \(p. 812\)](#).

It can take a while until Amazon S3 can bring the two ACLs in sync. This change in ownership applies only to objects created after you add a replication configuration to the bucket.

- Object tags, if there are any.
- S3 Object Lock retention information, if there is any.

When Amazon S3 replicates objects that have retention information applied, it applies those same retention controls to your replicas, overriding the default retention period configured on your destination buckets. If you don't have retention controls applied to the objects in your source bucket, and you replicate into destination buckets that have a default retention period set, the destination bucket's default retention period is applied to your object replicas. For more information, see [Using S3 Object Lock \(p. 680\)](#).

How delete operations affect replication

If you delete an object from the source bucket, the following actions occur by default:

- If you make a DELETE request without specifying an object version ID, Amazon S3 adds a delete marker. Amazon S3 deals with the delete marker as follows:
 - If you are using the latest version of the replication configuration (that is, you specify the `Filter` element in a replication configuration rule), Amazon S3 does not replicate the delete marker by default. However, you can add *delete marker replication* to non-tag-based rules. For more information, see [Replicating delete markers between buckets \(p. 810\)](#).
 - If you don't specify the `Filter` element, Amazon S3 assumes that the replication configuration is version V1, and it replicates delete markers that resulted from user actions. However, if Amazon S3 deletes an object due to a lifecycle action, the delete marker is not replicated to the destination buckets.
- If you specify an object version ID to delete in a DELETE request, Amazon S3 deletes that object version in the source bucket. But it doesn't replicate the deletion in the destination buckets. In other words, it doesn't delete the same object version from the destination buckets. This protects data from malicious deletions.

What isn't replicated with replication configurations?

By default, Amazon S3 doesn't replicate the following:

- Objects in the source bucket that are replicas that were created by another replication rule. For example, suppose you configure replication where bucket A is the source and bucket B is the destination. Now suppose that you add another replication configuration where bucket B is the source and bucket C is the destination. In this case, objects in bucket B that are replicas of objects in bucket A are not replicated to bucket C.

To replicate objects that are replicas, use Batch Replication. Learn more about configuring Batch Replication at [Replicate existing objects \(p. 798\)](#).

- Objects in the source bucket that have already been replicated to a different destination. For example, if you change the destination bucket in an existing replication configuration, Amazon S3 won't replicate the objects again.

To replicate previously replicated objects, use Batch Replication. Learn more about configuring Batch Replication at [Replicate existing objects \(p. 798\)](#).

- Batch Replication does not support re-replicating objects that were deleted with the version ID of the object from the destination bucket. To re-replicate these objects you can copy the source objects in place with a Batch Copy job. Copying those objects in place will create new versions of the object in the source bucket and initiate replication automatically to the destination. For more information about how to use Batch Copy, see, [Examples that use Batch Operations to copy objects \(p. 896\)](#).
- Objects created with server-side encryption using customer-provided encryption keys (SSE-C).
- By default, when replicating from a different AWS account, delete markers added to the source bucket are not replicated.

For information about how to replicate delete markers, see [Replicating delete markers between buckets \(p. 810\)](#).

- Objects that are stored in the S3 Glacier Flexible Retrieval or S3 Glacier Deep Archive storage class.

To learn more about the Amazon S3 Glacier service, see the [Amazon S3 Glacier Developer Guide](#).

- Objects in the source bucket that the bucket owner doesn't have sufficient permissions to replicate.

For information about how an object owner can grant permissions to a bucket owner, see [Granting cross-account permissions to upload objects while ensuring the bucket owner has full control \(p. 497\)](#).

- Updates to bucket-level subresources.

For example, if you change the lifecycle configuration or add a notification configuration to your source bucket, these changes are not applied to the destination bucket. This feature makes it possible to have different configurations on source and destination buckets.

- Actions performed by lifecycle configuration.

For example, if lifecycle configuration is enabled only on your source bucket, Amazon S3 creates delete markers for expired objects but doesn't replicate those markers. If you want the same lifecycle configuration applied to both the source and destination buckets, enable the same lifecycle configuration on both. For more information about lifecycle configuration, see [Managing your storage lifecycle \(p. 701\)](#).

Setting up replication

Note

Objects that existed before you set up replication aren't replicated automatically. In other words, Amazon S3 doesn't replicate objects retroactively. To replicate objects that were created before your replication configuration, use S3 Batch Replication. Learn more about configuring Batch Replication at [Replicate existing objects \(p. 798\)](#).

To enable Same-Region Replication (SRR) or Cross-Region Replication (CRR), add a replication configuration to your source bucket. The configuration tells Amazon S3 to replicate objects as specified. In the replication configuration, you must provide the following:

- **The destination buckets** – The bucket or buckets where you want Amazon S3 to replicate the objects.
- **The objects that you want to replicate** – You can replicate all of the objects in the source bucket or a subset. You identify a subset by providing a [key name prefix](#), one or more object tags, or both in the configuration.

For example, if you configure a replication rule to replicate only objects with the key name prefix `Tax/`, Amazon S3 replicates objects with keys such as `Tax/doc1` or `Tax/doc2`. But it doesn't replicate objects with the key `Legal/doc3`. If you specify both a prefix and one or more tags, Amazon S3 replicates only objects that have the specific key prefix and tags.

In addition to these minimum requirements, you can choose the following options:

- **Replica storage class** – By default, Amazon S3 stores object replicas using the same storage class as the source object. You can specify a different storage class for the replicas.
- **Replica ownership** – Amazon S3 assumes that an object replica continues to be owned by the owner of the source object. So when it replicates objects, it also replicates the corresponding object access control list (ACL) or S3 Object Ownership setting. If the source and destination buckets are owned by different AWS accounts, you can configure replication to change the owner of a replica to the AWS account that owns the destination bucket.

You can configure replication by using the REST API, AWS SDKs, AWS Command Line Interface (AWS CLI), or the Amazon S3 console.

Amazon S3 also provides API operations to support setting up replication rules. For more information, see the following topics in the *Amazon Simple Storage Service API Reference*:

- [PUT Bucket replication](#)
- [GET Bucket replication](#)
- [DELETE Bucket replication](#)

Topics

- [Replication configuration \(p. 759\)](#)
- [Setting up permissions \(p. 769\)](#)
- [Walkthroughs: Examples for configuring replication \(p. 772\)](#)

Replication configuration

Amazon S3 stores a replication configuration as XML. In the replication configuration XML file, you specify an AWS Identity and Access Management (IAM) role and one or more rules.

```
<ReplicationConfiguration>
  <Role>IAM-role-ARN</Role>
  <Rule>
    ...
  </Rule>
  <Rule>
    ...
  </Rule>
  ...
</ReplicationConfiguration>
```

Amazon S3 can't replicate objects without your permission. You grant permissions with the IAM role that you specify in the replication configuration. Amazon S3 assumes the IAM role to replicate objects on your behalf. You must grant the required permissions to the IAM role first. For more information about managing permissions, see [Setting up permissions \(p. 769\)](#).

You add one rule in a replication configuration in the following scenarios:

- You want to replicate all objects.
- You want to replicate one subset of objects. You identify the object subset by adding a filter in the rule. In the filter, you specify an object key prefix, tags, or a combination of both, to identify the subset of objects that the rule applies to.

You add multiple rules in a replication configuration if you want to replicate different subsets of objects. In each rule, you specify a filter that selects a different subset of objects. For example, you might choose to replicate objects that have either `tax/` or `document/` key prefixes. To do this, you add two rules, one that specifies the `tax/` key prefix filter and another that specifies the `document/` key prefix.

The following sections provide additional information.

Topics

- [Basic rule configuration \(p. 759\)](#)
- [Optional: Specifying a filter \(p. 760\)](#)
- [Additional destination configurations \(p. 761\)](#)
- [Example replication configurations \(p. 764\)](#)
- [Backward compatibility \(p. 768\)](#)

Basic rule configuration

Each rule must include the rule's status and priority. The rule must also indicate whether to replicate delete markers.

- Status indicates whether the rule is enabled or disabled by using the values `Enabled` or `Disabled`. If a rule is disabled, Amazon S3 doesn't perform the actions specified in the rule.

- Priority indicates which rule has precedence whenever two or more replication rules conflict. Amazon S3 attempts to replicate objects according to all replication rules. However, if there are two or more rules with the same destination bucket, then objects are replicated according to the rule with the highest priority. The higher the number, the higher the priority.
- DeleteMarkerReplication indicates whether to replicate delete markers by using the values Enabled or Disabled.

In the destination configuration, you must provide the name of the bucket or buckets where you want Amazon S3 to replicate objects.

The following example shows the minimum requirements for a V2 rule. For backward compatibility, Amazon S3 continues to support the XML V1 format. For more information, see [Backward compatibility \(p. 768\)](#).

```
...
<Rule>
  <ID>Rule-1</ID>
  <Status>Enabled-or-Disabled</Status>
  <Filter>
    <Prefix></Prefix>
  </Filter>
  <Priority>integer</Priority>
  <DeleteMarkerReplication>
    <Status>Enabled-or-Disabled</Status>
  </DeleteMarkerReplication>
  <Destination>
    <Bucket>arn:aws:s3:::DOC-EXAMPLE-BUCKET</Bucket>
  </Destination>
</Rule>
<Rule>
  ...
</Rule>
  ...
...
...
```

You can also specify other configuration options. For example, you might choose to use a storage class for object replicas that differs from the class for the source object.

Optional: Specifying a filter

To choose a subset of objects that the rule applies to, add an optional filter. You can filter by object key prefix, object tags, or a combination of both. If you filter on both a key prefix and object tags, Amazon S3 combines the filters by using a logical AND operator. In other words, the rule applies to a subset of objects with a specific key prefix and specific tags.

Filter based on object key prefix

To specify a rule with a filter based on an object key prefix, use the following code. You can specify only one prefix.

```
<Rule>
  ...
  <Filter>
    <Prefix>key-prefix</Prefix>
  </Filter>
  ...
</Rule>
...
```

Filter based on object tags

To specify a rule with a filter based on object tags, use the following code. You can specify one or more object tags.

```
<Rule>
  ...
  <Filter>
    <And>
      <Tag>
        <Key>key1</Key>
        <Value>value1</Value>
      </Tag>
      <Tag>
        <Key>key2</Key>
        <Value>value2</Value>
      </Tag>
      ...
    </And>
  </Filter>
  ...
</Rule>
...
```

Filter with a key prefix and object tags

To specify a rule filter with a combination of a key prefix and object tags, use the following code. You wrap these filters in an And parent element. Amazon S3 performs a logical AND operation to combine these filters. In other words, the rule applies to a subset of objects with both a specific key prefix and specific tags.

```
<Rule>
  ...
  <Filter>
    <And>
      <Prefix>key-prefix</Prefix>
      <Tag>
        <Key>key1</Key>
        <Value>value1</Value>
      </Tag>
      <Tag>
        <Key>key2</Key>
        <Value>value2</Value>
      </Tag>
      ...
    </And>
  </Filter>
  ...
</Rule>
...
```

Note

If you specify a rule with an empty filter tag, your rule applies to all objects in your bucket.

Additional destination configurations

In the destination configuration, you specify the bucket or buckets where you want Amazon S3 to replicate objects. You can set configurations to replicate objects from one source bucket to one or more destination buckets.

```
...
<Destination>
  <Bucket>arn:aws:s3:::DOC-EXAMPLE-BUCKET</Bucket>
</Destination>
```

...

You can add the following options in the <Destination> element.

Topics

- [Specify storage class \(p. 762\)](#)
- [Add multiple destination buckets \(p. 762\)](#)
- [Specify different parameters for each replication rule with multiple destination buckets \(p. 762\)](#)
- [Change replica ownership \(p. 763\)](#)
- [Enable S3 Replication Time Control \(p. 764\)](#)
- [Replicate objects created with server-side encryption by using AWS KMS \(p. 764\)](#)

Specify storage class

You can specify the storage class for the object replicas. By default, Amazon S3 uses the storage class of the source object to create object replicas, as in the following example.

```
...
<Destination>
    <Bucket>arn:aws:s3:::DOC-EXAMPLE-BUCKET</Bucket>
    <StorageClass>storage-class</StorageClass>
</Destination>
...
```

Add multiple destination buckets

You can add multiple destination buckets in a single replication configuration, as follows.

```
...
<Rule>
    <ID>Rule-1</ID>
    <Status>Enabled-or-Disabled</Status>
    <Priority>integer</Priority>
    <DeleteMarkerReplication>
        <Status>Enabled-or-Disabled</Status>
    </DeleteMarkerReplication>
    <Destination>
        <Bucket>arn:aws:s3:::DOC-EXAMPLE-BUCKET1</Bucket>
    </Destination>
</Rule>
<Rule>
    <ID>Rule-2</ID>
    <Status>Enabled-or-Disabled</Status>
    <Priority>integer</Priority>
    <DeleteMarkerReplication>
        <Status>Enabled-or-Disabled</Status>
    </DeleteMarkerReplication>
    <Destination>
        <Bucket>arn:aws:s3:::DOC-EXAMPLE-BUCKET2</Bucket>
    </Destination>
</Rule>
...
```

Specify different parameters for each replication rule with multiple destination buckets

When adding multiple destination buckets in a single replication configuration, you can specify different parameters for each replication rule, as follows.

```

...
<Rule>
  <ID>Rule-1</ID>
  <Status>Enabled-or-Disabled</Status>
  <Priority>integer</Priority>
  <DeleteMarkerReplication>
    <Status>Disabled</Status>
  </DeleteMarkerReplication>
  <Metrics>
    <Status>Enabled</Status>
    <EventThreshold>
      <Minutes>15</Minutes>
    </EventThreshold>
  </Metrics>
  <Destination>
    <Bucket>arn:aws:s3:::DOC-EXAMPLE-BUCKET1</Bucket>
  </Destination>
</Rule>
<Rule>
  <ID>Rule-2</ID>
  <Status>Enabled-or-Disabled</Status>
  <Priority>integer</Priority>
  <DeleteMarkerReplication>
    <Status>Enabled</Status>
  </DeleteMarkerReplication>
  <Metrics>
    <Status>Enabled</Status>
    <EventThreshold>
      <Minutes>15</Minutes>
    </EventThreshold>
  </Metrics>
  <ReplicationTime>
    <Status>Enabled</Status>
    <Time>
      <Minutes>15</Minutes>
    </Time>
  </ReplicationTime>
  <Destination>
    <Bucket>arn:aws:s3:::DOC-EXAMPLE-BUCKET2</Bucket>
  </Destination>
</Rule>
...

```

Change replica ownership

When the source and destination buckets aren't owned by the same accounts, you can change the ownership of the replica to the AWS account that owns the destination bucket. To do so, add the `AccessControlTranslation` element. This element takes the value `Destination`.

```

...
<Destination>
  <Bucket>arn:aws:s3:::DOC-EXAMPLE-BUCKET</Bucket>
  <Account>destination-bucket-owner-account-id</Account>
  <AccessControlTranslation>
    <Owner>Destination</Owner>
  </AccessControlTranslation>
</Destination>
...

```

If you don't add the `AccessControlTranslation` element to the replication configuration, the replicas are owned by the same AWS account that owns the source object. For more information, see [Changing the replica owner \(p. 812\)](#).

Enable S3 Replication Time Control

You can enable S3 Replication Time Control (S3 RTC) in your replication configuration. S3 RTC replicates most objects in seconds and 99.99 percent of objects within 15 minutes (backed by a service-level agreement).

Note

Only a value of <Minutes>15</Minutes> is accepted for EventThreshold and Time.

```
...
<Destination>
  <Bucket>arn:aws:s3:::DOC-EXAMPLE-BUCKET</Bucket>
  <Metrics>
    <Status>Enabled</Status>
    <EventThreshold>
      <Minutes>15</Minutes>
    </EventThreshold>
  </Metrics>
  <ReplicationTime>
    <Status>Enabled</Status>
    <Time>
      <Minutes>15</Minutes>
    </Time>
  </ReplicationTime>
</Destination>
...
...
```

For more information, see [Meeting compliance requirements using S3 Replication Time Control \(S3 RTC\) \(p. 807\)](#). For API examples, see [PutBucketReplication](#) in the *Amazon Simple Storage Service API Reference*.

Replicate objects created with server-side encryption by using AWS KMS

Your source bucket might contain objects that were created with server-side encryption by using AWS Key Management Service (AWS KMS) keys (SSE-KMS). By default, Amazon S3 doesn't replicate these objects. You can optionally direct Amazon S3 to replicate these objects. To do so, first explicitly opt into this feature by adding the SourceSelectionCriteria element. Then provide the AWS KMS key (for the AWS Region of the destination bucket) to use for encrypting object replicas. The following example shows how to specify these elements.

```
...
<SourceSelectionCriteria>
  <SseKmsEncryptedObjects>
    <Status>Enabled</Status>
  </SseKmsEncryptedObjects>
</SourceSelectionCriteria>
<Destination>
  <Bucket>arn:aws:s3:::DOC-EXAMPLE-BUCKET</Bucket>
  <EncryptionConfiguration>
    <ReplicaKmsKeyId>AWS KMS key ID to use for encrypting object replicas</ReplicaKmsKeyId>
  </EncryptionConfiguration>
</Destination>
...
...
```

For more information, see [Replicating objects created with server-side encryption \(SSE\) using KMS keys \(p. 814\)](#).

Example replication configurations

To get started, you can add the following example replication configurations to your bucket, as appropriate.

Important

To add a replication configuration to a bucket, you must have the `iam:PassRole` permission. This permission allows you to pass the IAM role that grants Amazon S3 replication permissions. You specify the IAM role by providing the Amazon Resource Name (ARN) that is used in the `Role` element in the replication configuration XML. For more information, see [Granting a User Permissions to Pass a Role to an AWS service](#) in the *IAM User Guide*.

Example 1: Replication configuration with one rule

The following basic replication configuration specifies one rule. The rule specifies an IAM role that Amazon S3 can assume and a single destination bucket for object replicas. The `Status` value of `Enabled` indicates that the rule is in effect.

```
<?xml version="1.0" encoding="UTF-8"?>
<ReplicationConfiguration xmlns="http://s3.amazonaws.com/doc/2006-03-01/">
  <Role>arn:aws:iam::account-id:role/role-name</Role>
  <Rule>
    <Status>Enabled</Status>

    <Destination><Bucket>arn:aws:s3:::DOC-EXAMPLE-BUCKET</Bucket></Destination>

  </Rule>
</ReplicationConfiguration>
```

To choose a subset of objects to replicate, you can add a filter. In the following configuration, the filter specifies an object key prefix. This rule applies to objects that have the prefix `Tax/` in their key names.

```
<?xml version="1.0" encoding="UTF-8"?>
<ReplicationConfiguration xmlns="http://s3.amazonaws.com/doc/2006-03-01/">
  <Role>arn:aws:iam::account-id:role/role-name</Role>
  <Rule>
    <Status>Enabled</Status>
    <Priority>1</Priority>
    <DeleteMarkerReplication>
      <Status>string</Status>
    </DeleteMarkerReplication>

    <Filter>
      <Prefix>Tax/</Prefix>
    </Filter>

    <Destination><Bucket>arn:aws:s3:::DOC-EXAMPLE-BUCKET</Bucket></Destination>

  </Rule>
</ReplicationConfiguration>
```

If you specify the `Filter` element, you must also include the `Priority` and `DeleteMarkerReplication` elements. In this example, `Priority` is irrelevant because there is only one rule.

In the following configuration, the filter specifies one prefix and two tags. The rule applies to the subset of objects that have the specified key prefix and tags. Specifically, it applies to objects that have the `Tax/` prefix in their key names and the two specified object tags. `Priority` doesn't apply because there is only one rule.

```
<?xml version="1.0" encoding="UTF-8"?>
<ReplicationConfiguration xmlns="http://s3.amazonaws.com/doc/2006-03-01/">
  <Role>arn:aws:iam::account-id:role/role-name</Role>
  <Rule>
    <Status>Enabled</Status>
    <Priority>1</Priority>
```

```

<DeleteMarkerReplication>
    <Status>string</Status>
</DeleteMarkerReplication>

<Filter>
    <And>
        <Prefix>Tax/</Prefix>
        <Tag>
            <Tag>
                <Key>tagA</Key>
                <Value>valueA</Value>
            </Tag>
        </Tag>
        <Tag>
            <Tag>
                <Key>tagB</Key>
                <Value>valueB</Value>
            </Tag>
        </Tag>
    </And>
</Filter>

<Destination><Bucket>arn:aws:s3:::DOC-EXAMPLE-BUCKET</Bucket></Destination>

</Rule>
</ReplicationConfiguration>

```

You can specify a storage class for the object replicas as follows.

```

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

<ReplicationConfiguration xmlns="http://s3.amazonaws.com/doc/2006-03-01/">
    <Role>arn:aws:iam::account-id:role/role-name</Role>
    <Rule>
        <Status>Enabled</Status>
        <Destination>
            <Bucket>arn:aws:s3:::DOC-EXAMPLE-BUCKET</Bucket>
            <StorageClass>storage-class</StorageClass>
        </Destination>
    </Rule>
</ReplicationConfiguration>

```

You can specify any storage class that Amazon S3 supports.

Example 2: Replication configuration with two rules

Example

In the following replication configuration:

- Each rule filters on a different key prefix so that each rule applies to a distinct subset of objects. In this example, Amazon S3 replicates objects with the key names **Tax/doc1.pdf** and **Project/project1.txt**, but it doesn't replicate objects with the key name **PersonalDoc/documentA**.
- Rule priority is irrelevant because the rules apply to two distinct sets of objects. The next example shows what happens when rule priority is applied.
- The second rule specifies the S3 Standard-IA storage class for object replicas. Amazon S3 uses the specified storage class for those object replicas.

```

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

```

<ReplicationConfiguration xmlns="http://s3.amazonaws.com/doc/2006-03-01/">
  <Role>arn:aws:iam::account-id:role/role-name</Role>
  <Rule>
    <Status>Enabled</Status>
    <Priority>1</Priority>
    <DeleteMarkerReplication>
      <Status>string</Status>
    </DeleteMarkerReplication>
    <Filter>
      <Prefix>Tax</Prefix>
    </Filter>
    <Status>Enabled</Status>
    <Destination>
      <Bucket>arn:aws:s3:::DOC-EXAMPLE-BUCKET1</Bucket>
    </Destination>
    ...
  </Rule>
  <Rule>
    <Status>Enabled</Status>
    <Priority>2</Priority>
    <DeleteMarkerReplication>
      <Status>string</Status>
    </DeleteMarkerReplication>
    <Filter>
      <Prefix>Project</Prefix>
    </Filter>
    <Status>Enabled</Status>
    <Destination>
      <Bucket>arn:aws:s3:::DOC-EXAMPLE-BUCKET1</Bucket>
      <StorageClass>STANDARD_IA</StorageClass>
    </Destination>
    ...
  </Rule>

</ReplicationConfiguration>

```

Example 3: Replication configuration with two rules with overlapping prefixes

In this configuration, the two rules specify filters with overlapping key prefixes, *star*/ and *starship*. Both rules apply to objects with the key name *starship-x*. In this case, Amazon S3 uses the rule priority to determine which rule to apply. The higher the number, the higher the priority.

```

<ReplicationConfiguration>

  <Role>arn:aws:iam::account-id:role/role-name</Role>

  <Rule>
    <Status>Enabled</Status>
    <Priority>1</Priority>
    <DeleteMarkerReplication>
      <Status>string</Status>
    </DeleteMarkerReplication>
    <Filter>
      <Prefix>star</Prefix>
    </Filter>
    <Destination>
      <Bucket>arn:aws:s3:::DOC-EXAMPLE-BUCKET1</Bucket>
    </Destination>
  </Rule>
  <Rule>
    <Status>Enabled</Status>
    <Priority>2</Priority>

```

```
<DeleteMarkerReplication>
    <Status>string</Status>
</DeleteMarkerReplication>
<Filter>
    <Prefix>starship</Prefix>
</Filter>
<Destination>
    <Bucket>arn:aws:s3:::DOC-EXAMPLE-BUCKET1</Bucket>
</Destination>
</Rule>
</ReplicationConfiguration>
```

Example 4: Example walkthroughs

For example walkthroughs, see [Walkthroughs: Examples for configuring replication \(p. 772\)](#).

For more information about the XML structure of replication configuration, see [PutBucketReplication](#) in the *Amazon Simple Storage Service API Reference*.

Backward compatibility

The latest version of the replication configuration XML is V2. XML V2 replication configurations are those that contain the `Filter` element for rules, and rules that specify S3 Replication Time Control (S3 RTC).

To see your replication configuration version, you can use the `GetBucketReplication` API operation. For more information, see [GetBucketReplication](#) in the *Amazon Simple Storage Service API Reference*.

For backward compatibility, Amazon S3 continues to support the XML V1 replication configuration. If you've used XML V1 replication configuration, consider the following issues that affect backward compatibility:

- Replication configuration XML V2 includes the `Filter` element for rules. With the `Filter` element, you can specify object filters based on the object key prefix, tags, or both to scope the objects that the rule applies to. Replication configuration XML V1 supports filtering based only on the key prefix. In that case, you add the `Prefix` directly as a child element of the `Rule` element, as in the following example.

```
<?xml version="1.0" encoding="UTF-8"?>
<ReplicationConfiguration xmlns="http://s3.amazonaws.com/doc/2006-03-01/">
    <Role>arn:aws:iam::account-id:role/role-name</Role>
    <Rule>
        <Status>Enabled</Status>
        <Prefix>key-prefix</Prefix>
        <Destination><Bucket>arn:aws:s3:::DOC-EXAMPLE-BUCKET</Bucket></Destination>
    </Rule>
</ReplicationConfiguration>
```

For backward compatibility, Amazon S3 continues to support the V1 configuration.

- When you delete an object from your source bucket without specifying an object version ID, Amazon S3 adds a delete marker. If you use V1 of the replication configuration XML, Amazon S3 replicates delete markers that result from user actions. In other words, Amazon S3 replicates the delete marker only if a user deletes an object. If an expired object is removed by Amazon S3 (as part of a lifecycle action), Amazon S3 does not replicate the delete marker.

In V2 replication configurations, you can enable delete marker replication for tag-based rules. For more information, see [Replicating delete markers between buckets \(p. 810\)](#).

Setting up permissions

When setting up replication, you must acquire the necessary permissions as follows:

- Amazon S3 needs permissions to replicate objects on your behalf. You grant these permissions by creating an IAM role and then specifying that role in your replication configuration.
- When the source and destination buckets aren't owned by the same accounts, the owner of the destination bucket must grant the source bucket owner permissions to store the replicas.

Topics

- [Creating an IAM role \(p. 769\)](#)
- [Granting permissions when the source and destination buckets are owned by different AWS accounts \(p. 771\)](#)
- [Changing replica ownership \(p. 772\)](#)
- [Enable receiving replicated objects from a source bucket \(p. 772\)](#)

Creating an IAM role

By default, all Amazon S3 resources—buckets, objects, and related subresources—are private, and only the resource owner can access the resource. Amazon S3 needs permissions to read and replicate objects from the source bucket. You grant these permissions by creating an IAM role and specifying the role in your replication configuration.

This section explains the trust policy and minimum required permissions policy. The example walkthroughs provide step-by-step instructions to create an IAM role. For more information, see [Walkthroughs: Examples for configuring replication \(p. 772\)](#).

- The following example shows a *trust policy*, where you identify Amazon S3 as the service principal who can assume the role.

```
{  
    "Version": "2012-10-17",  
    "Statement": [  
        {  
            "Effect": "Allow",  
            "Principal": {  
                "Service": "s3.amazonaws.com"  
            },  
            "Action": "sts:AssumeRole"  
        }  
    ]  
}
```

For more information about IAM roles, see [IAM Roles](#) in the *IAM User Guide*.

- The following example shows an *access policy*, where you grant the role permissions to perform replication tasks on your behalf. When Amazon S3 assumes the role, it has the permissions that you specify in this policy. In this policy, **DOC-EXAMPLE-BUCKET1** is the source bucket, and **DOC-EXAMPLE-BUCKET2** is the destination bucket.

```
{  
    "Version": "2012-10-17",  
    "Statement": [  
        {  
            "Effect": "Allow",  
            "Action": "s3:ReplicateObject",  
            "Resource": "arn:aws:s3:::DOC-EXAMPLE-BUCKET1/*",  
            "Condition": {  
                "StringEquals": {  
                    "aws:SourceRegion": "us-west-2",  
                    "aws:SourceBucket": "DOC-EXAMPLE-BUCKET1"  
                }  
            }  
        }  
    ]  
}
```

```
"Action": [
    "s3:GetReplicationConfiguration",
    "s3>ListBucket"
],
"Resource": [
    "arn:aws:s3::::DOC-EXAMPLE-BUCKET1"
]
},
{
    "Effect": "Allow",
    "Action": [
        "s3.GetObjectVersionForReplication",
        "s3GetObjectVersionAcl",
        "s3GetObjectVersionTagging"
    ],
    "Resource": [
        "arn:aws:s3::::DOC-EXAMPLE-BUCKET1/*"
    ]
},
{
    "Effect": "Allow",
    "Action": [
        "s3:ReplicateObject",
        "s3:ReplicateDelete",
        "s3:ReplicateTags"
    ],
    "Resource": "arn:aws:s3::::DOC-EXAMPLE-BUCKET2/*"
}
]
```

The access policy grants permissions for the following actions:

- `s3:GetReplicationConfiguration` and `s3>ListBucket` – Permissions for these actions on the `DOC-EXAMPLE-BUCKET1` bucket (the source bucket) allow Amazon S3 to retrieve the replication configuration and list the bucket content. (The current permissions model requires the `s3>ListBucket` permission for accessing delete markers.)
- `s3.GetObjectVersionForReplication` and `s3GetObjectVersionAcl` – Permissions for these actions are granted on all objects to allow Amazon S3 to get a specific object version and access control list (ACL) associated with the objects.
- `s3:ReplicateObject` and `s3:ReplicateDelete` – Permissions for these actions on all objects in the `DOC-EXAMPLE-BUCKET2` bucket (the destination bucket) allow Amazon S3 to replicate objects or delete markers to the destination bucket. For information about delete markers, see [How delete operations affect replication \(p. 757\)](#).

Note

Permissions for the `s3:ReplicateObject` action on the `DOC-EXAMPLE-BUCKET2` bucket (the destination bucket) also allow replication of object tags, so you don't need to explicitly grant permission for the `s3:ReplicateTags` action.

- `s3GetObjectVersionTagging` – Permissions for this action on objects in the `DOC-EXAMPLE-BUCKET1` bucket (the source bucket) allow Amazon S3 to read object tags for replication. For more information, see [Categorizing your storage using tags \(p. 825\)](#). If Amazon S3 doesn't have these permissions, it replicates the objects, but not the object tags.

For a list of Amazon S3 actions, see [Amazon S3 actions \(p. 415\)](#).

Important

The AWS account that owns the IAM role must have permissions for the actions that it grants to the IAM role.

For example, suppose that the source bucket contains objects owned by another AWS account. The owner of the objects must explicitly grant the AWS account that owns the IAM role the

required permissions through the object ACL. Otherwise, Amazon S3 can't access the objects, and replication of the objects fails. For information about ACL permissions, see [Access control list \(ACL\) overview \(p. 554\)](#).

The permissions described here are related to the minimum replication configuration. If you choose to add optional replication configurations, you must grant additional permissions to Amazon S3. For more information, see [Additional replication configurations \(p. 805\)](#).

Granting permissions when the source and destination buckets are owned by different AWS accounts

When the source and destination buckets aren't owned by the same accounts, the owner of the destination bucket must also add a bucket policy to grant the owner of the source bucket permissions to perform replication actions, as follows. In this policy, `DOC-EXAMPLE-BUCKET2` is the destination bucket.

```
{
    "Version": "2012-10-17",
    "Id": "PolicyForDestinationBucket",
    "Statement": [
        {
            "Sid": "Permissions on objects",
            "Effect": "Allow",
            "Principal": {
                "AWS": "arn:aws:iam::SourceBucket-account-ID:role/service-role/source-account-IAM-role"
            },
            "Action": [
                "s3:ReplicateDelete",
                "s3:ReplicateObject"
            ],
            "Resource": "arn:aws:s3:::DOC-EXAMPLE-BUCKET2/*"
        },
        {
            "Sid": "Permissions on bucket",
            "Effect": "Allow",
            "Principal": {
                "AWS": "arn:aws:iam::SourceBucket-account-ID:role/service-role/source-account-IAM-role"
            },
            "Action": [
                "s3>List*",
                "s3:GetBucketVersioning",
                "s3:PutBucketVersioning"
            ],
            "Resource": "arn:aws:s3:::DOC-EXAMPLE-BUCKET2"
        }
    ]
}
```

For an example, see [Configuring replication when source and destination buckets are owned by different accounts \(p. 785\)](#).

If objects in the source bucket are tagged, note the following:

- If the source bucket owner grants Amazon S3 permission for the `s3:GetObjectVersionTagging` and `s3:ReplicateTags` actions to replicate object tags (through the IAM role), Amazon S3 replicates the tags along with the objects. For information about the IAM role, see [Creating an IAM role \(p. 769\)](#).
- If the owner of the destination bucket doesn't want to replicate the tags, they can add the following statement to the destination bucket policy to explicitly deny permission for the `s3:ReplicateTags` action. In this policy, `DOC-EXAMPLE-BUCKET2` is the destination bucket.

```
...
    "Statement": [
        {
            "Effect": "Deny",
            "Principal": {
                "AWS": "arn:aws:iam::SourceBucket-account-id:role/service-role/source-account-
IAM-role"
            },
            "Action": "s3:ReplicateTags",
            "Resource": "arn:aws:s3:::DOC-EXAMPLE-BUCKET2/*"
        }
    ]
...

```

Changing replica ownership

When different AWS accounts own the source and destination buckets, you can tell Amazon S3 to change the ownership of the replica to the AWS account that owns the destination bucket. For more information about owner override, see [Changing the replica owner \(p. 812\)](#).

Enable receiving replicated objects from a source bucket

You can quickly generate the policies needed to enable receiving replicated objects from a source bucket through the AWS Management Console.

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. In the **Buckets** list, choose the bucket that you want to use as a destination bucket.
4. Choose the **Management** tab, and scroll down to **Replication rules**.
5. For **Actions**, choose **Receive replicated objects**.

- Follow the prompts and enter the AWS account ID of the source bucket account and choose **Generate policies**. This will generate an Amazon S3 bucket policy and a KMS key policy.
6. To add this policy to your existing bucket policy, either choose **Apply settings** or choose **Copy** to manually copy the changes.
 7. (Optional) Copy the AWS KMS policy to your desired KMS key policy on the AWS Key Management Service console.

Walkthroughs: Examples for configuring replication

The following examples show how to configure live replication for common use cases. The examples demonstrate replication configuration using the Amazon S3 console, AWS Command Line Interface (AWS CLI), and AWS SDKs (Java and .NET SDK examples are shown). For information about installing and configuring the AWS CLI, see the following topics in the *AWS Command Line Interface User Guide*.

Note

Live replication refers to Same-Region Replication (SRR) and Cross-Region Replication (CRR). For an on-demand replication action to sync buckets and replicate existing objects, see [Replicate existing objects \(p. 798\)](#).

- [Installing the AWS Command Line Interface](#)
- [Configuring the AWS CLI](#) – You must set up at least one profile. If you are exploring cross-account scenarios, set up two profiles.

For information about AWS SDKs, see [AWS SDK for Java](#) and [AWS SDK for .NET](#).

Topics

- [Configuring replication for source and destination buckets owned by the same account \(p. 773\)](#)
- [Configuring replication when source and destination buckets are owned by different accounts \(p. 785\)](#)
- [Changing the replica owner when source and destination buckets are owned by different accounts \(p. 786\)](#)
- [Replicating encrypted objects \(p. 790\)](#)
- [Replicating objects with S3 Replication Time Control \(S3 RTC\) \(p. 795\)](#)
- [Managing replication rules using the Amazon S3 console \(p. 797\)](#)

Configuring replication for source and destination buckets owned by the same account

Replication is the automatic, asynchronous copying of objects across buckets in the same or different AWS Regions. Replication copies newly created objects and object updates from a source bucket to a destination bucket or buckets. For more information, see [Replicating objects \(p. 753\)](#).

When you configure replication, you add replication rules to the source bucket. Replication rules define which source bucket objects to replicate and the destination bucket or buckets where the replicated objects are stored. You can create a rule to replicate all the objects in a bucket or a subset of objects with a specific key name prefix, one or more object tags, or both. A destination bucket can be in the same AWS account as the source bucket, or it can be in a different account.

If you specify an object version ID to delete, Amazon S3 deletes that object version in the source bucket. But it doesn't replicate the deletion in the destination bucket. In other words, it doesn't delete the same object version from the destination bucket. This protects data from malicious deletions.

When you add a replication rule to a bucket, the rule is enabled by default, so it starts working as soon as you save it.

In this example, you set up replication for source and destination buckets that are owned by the same AWS account. Examples are provided for using the Amazon S3 console, the AWS Command Line Interface (AWS CLI), and the AWS SDK for Java and AWS SDK for .NET.

Using the S3 console

Follow these steps to configure a replication rule when the destination bucket is in the same AWS account as the source bucket.

If the destination bucket is in a different account from the source bucket, you must add a bucket policy to the destination bucket to grant the owner of the source bucket account permission to replicate objects in the destination bucket. For more information, see [Granting permissions when the source and destination buckets are owned by different AWS accounts \(p. 771\)](#).

1. Sign in to the AWS Management Console and open the Amazon S3 console at <https://console.aws.amazon.com/s3/>.
2. In the **Buckets** list, choose the name of the bucket that you want.
3. Choose **Management**, scroll down to **Replication rules**, and then choose **Create replication rule**.
4. Under **Rule name**, enter a name for your rule to help identify the rule later. The name is required and must be unique within the bucket.
5. Set up an AWS Identity and Access Management (IAM) role that Amazon S3 can assume to replicate objects on your behalf.

To set up an IAM role, on the **Replication rule configuration** section, under **IAM role**, do one of the following:

- We highly recommend that you choose **Create new role** to have Amazon S3 create a new IAM role for you. When you save the rule, a new policy is generated for the IAM role that matches the source and destination buckets that you choose.
- You can choose to use an existing IAM role. If you do, you must choose a role that grants Amazon S3 the necessary permissions for replication. Replication fails if this role does not grant Amazon S3 sufficient permissions to follow your replication rule.

Important

When you add a replication rule to a bucket, you must have the `iam:PassRole` permission to be able to pass the IAM role that grants Amazon S3 replication permissions. For more information, see [Granting a user permissions to pass a role to an AWS service](#) in the *IAM User Guide*.

6. Under **Status**, see that **Enabled** is selected by default. An enabled rule starts to work as soon as you save it. If you want to enable the rule later, select **Disabled**.
7. If the bucket has existing replication rules, you are instructed to set a priority for the rule. You must set a priority for the rule to avoid conflicts caused by objects that are included in the scope of more than one rule. In the case of overlapping rules, Amazon S3 uses the rule priority to determine which rule to apply. The higher the number, the higher the priority. For more information about rule priority, see [Replication configuration \(p. 759\)](#).
8. In the **Replication rule configuration**, under **Source bucket**, you have the following options for setting the replication source:
 - To replicate the whole bucket, choose **This rule applies to all objects in the bucket**.
 - To replicate all objects that have the same prefix, choose **Limit the scope of this rule using one or more filters**. This limits replication to all objects that have names that begin with the string (for example `pictures`). Enter a prefix in the box.

Note

If you enter a prefix that is the name of a folder, you must use `/` (forward slash) as the last character (for example, `pictures/`).

- To replicate all objects with one or more object tags, select **Add tag** and enter the key-value pair in the boxes. Repeat the procedure to add another tag. You can combine a prefix and tags. For more information about object tags, see [Categorizing your storage using tags \(p. 825\)](#).

The new schema supports prefix and tag filtering and the prioritization of rules. For more information about the new schema, see [Backward compatibility \(p. 768\)](#). For more information about the XML used with the Amazon S3 API that works behind the user interface, see [Replication configuration \(p. 759\)](#). The new schema is described as *replication configuration XML V2*.

9. Under **Destination**, select the bucket where you want Amazon S3 to replicate objects.

Note

The number of destination buckets is limited to the number of AWS Regions in a given partition. A partition is a grouping of Regions. AWS currently has three partitions: `aws` (Standard Regions), `aws-cn` (China Regions), and `aws-us-gov` (AWS GovCloud [US] Regions). You can use [service quotas](#) to request an increase in your destination bucket limit.

- To replicate to a bucket or buckets in your account, select **Choose a bucket in this account**, and enter or browse for the destination bucket names.
- To replicate to a bucket or buckets in a different AWS account, select **Choose a bucket in another account**, and enter the destination bucket account ID and name.

If the destination is in a different account from the source bucket, you must add a bucket policy to the destination buckets to grant the owner of the source bucket account permission to replicate objects. For more information, see [Granting permissions when the source and destination buckets are owned by different AWS accounts \(p. 771\)](#).

Note

If versioning is not enabled on the destination bucket, you get a warning that contains an **Enable versioning** button. Choose this button to enable versioning on the bucket.

10. You have the following additional options while setting the **Destination**:

- If you want to enable **Object Ownership** to help standardize ownership of new objects in the destination bucket, choose **Change object ownership to the destination bucket owner**. For more information about this option, see [Controlling ownership of objects and disabling ACLs for your bucket \(p. 601\)](#).
- If you want to replicate your data into a specific storage class in the destination, choose **Change the storage class for the replicated objects**. Then choose the storage class that you want to use for the replicated objects in the destination. If you don't choose this option, the storage class for replicated objects is the same class as the original objects.
- If you want to enable delete marker replication in your replication configuration, select **Delete marker replication**. For more information see, [Replicating delete markers between buckets \(p. 810\)](#).
- If you want to enable Amazon S3 replica modification sync in your replication configuration, select **Replica modification sync**. For more information see, [Replicating metadata changes with Amazon S3 replica modification sync \(p. 811\)](#).
- If you want to enable S3 replication metrics in your replication configuration, select **Replication metrics and events**. For more information see, [Monitoring progress with replication metrics and Amazon S3 event notifications \(p. 805\)](#).
- If you want to enable S3 Replication Time Control (S3 RTC) in your replication configuration, select **S3 Replication Time Control**. For more information about this option, see [Meeting compliance requirements using S3 Replication Time Control \(S3 RTC\) \(p. 807\)](#).

Note

When you use S3 RTC or S3 replication metrics, additional fees apply.

11. To replicate objects in the source bucket that are encrypted with AWS Key Management Service (AWS KMS), under **Replication criteria**, select **Replicate objects encrypted with AWS KMS**. Under **AWS KMS key for encrypting destination objects** are the source keys that you allow replication to use. All source KMS keys are included by default. You can choose to narrow the KMS key selection.

Objects encrypted by AWS KMS keys that you do not select are not replicated. A KMS key or a group of KMS keys is chosen for you, but you can choose the KMS keys if you want. For information about using AWS KMS with replication, see [Replicating objects created with server-side encryption \(SSE\) using KMS keys \(p. 814\)](#).

Important

When you replicate objects that are encrypted with AWS KMS, the AWS KMS request rate doubles in the source Region and increases in the destination Region by the same amount. These increased call rates to AWS KMS are due to the way that data is re-encrypted using the KMS key that you define for the replication destination Region. AWS KMS has a request rate limit that is per calling account per Region. For information about the limit defaults, see [AWS KMS Limits - Requests per Second: Varies](#) in the *AWS Key Management Service Developer Guide*.

If your current Amazon S3 PUT object request rate during replication is more than half the default AWS KMS rate limit for your account, we recommend that you request an increase to your AWS KMS request rate limit. To request an increase, create a case in the

AWS Support Center at [Contact Us](#). For example, suppose that your current PUT object request rate is 1,000 requests per second and you use AWS KMS to encrypt your objects. In this case, we recommend that you ask AWS Support to increase your AWS KMS rate limit to 2,500 requests per second, in both your source and destination Regions (if different), to ensure that there is no throttling by AWS KMS.

To see your PUT object request rate in the source bucket, view `PutRequests` in the Amazon CloudWatch request metrics for Amazon S3. For information about viewing CloudWatch metrics, see [Using the S3 console \(p. 1013\)](#)

If you chose to replicate objects encrypted with AWS KMS, enter the Amazon Resource Name (ARN) of the AWS KMS key to use to encrypt the replicas in the destination bucket. You can find the ARN for your KMS key in the IAM console, under **Encryption keys**. Or, you can choose a KMS key name from the drop-down list.

For more information about creating an AWS KMS key, see [Creating Keys](#) in the *AWS Key Management Service Developer Guide*.

Important

The Amazon S3 console lists only 100 KMS keys per AWS Region. If you have more than 100 KMS keys in the same Region, you can see only the first 100 KMS keys in the S3 console.

To use a KMS key that is not listed in the console, choose **Custom KMS ARN**, and enter the KMS key ARN.

12. To finish, choose **Save**.
13. After you save your rule, you can edit, enable, disable, or delete your rule by selecting your rule and choosing **Edit rule**.

Using the AWS CLI

To use the AWS CLI to set up replication when the source and destination buckets are owned by the same AWS account, you do the following:

- Create source and destination buckets
- Enable versioning on the buckets
- Create an IAM role that gives Amazon S3 permission to replicate objects
- Add the replication configuration to the source bucket

To verify your setup, you test it.

To set up replication when source and destination buckets are owned by the same AWS account

1. Set a credentials profile for the AWS CLI. In this example, we use the profile name `acctA`. For information about setting credential profiles, see [Named Profiles](#) in the *AWS Command Line Interface User Guide*.

Important

The profile you use for this exercise must have the necessary permissions. For example, in the replication configuration, you specify the IAM role that Amazon S3 can assume. You can do this only if the profile you use has the `iam:PassRole` permission. For more information, see [Granting a User Permissions to Pass a Role to an AWS Service](#) in the *IAM User Guide*. If you use administrator credentials to create a named profile, you can perform all the tasks.

2. Create a `source` bucket and enable versioning on it. The following code creates a `source` bucket in the US East (N. Virginia) (`us-east-1`) Region.

```
aws s3api create-bucket \
```

```
--bucket source \  
--region us-east-1 \  
--profile acctA
```

```
aws s3api put-bucket-versioning \  
--bucket source \  
--versioning-configuration Status=Enabled \  
--profile acctA
```

3. Create a *destination* bucket and enable versioning on it. The following code creates a *destination* bucket in the US West (Oregon) (us-west-2) Region.

Note

To set up replication configuration when both source and destination buckets are in the same AWS account, you use the same profile. This example uses acctA. To test replication configuration when the buckets are owned by different AWS accounts, you specify different profiles for each. This example uses the acctB profile for the destination bucket.

```
aws s3api create-bucket \  
--bucket destination \  
--region us-west-2 \  
--create-bucket-configuration LocationConstraint=us-west-2 \  
--profile acctA
```

```
aws s3api put-bucket-versioning \  
--bucket destination \  
--versioning-configuration Status=Enabled \  
--profile acctA
```

4. Create an IAM role. You specify this role in the replication configuration that you add to the *source* bucket later. Amazon S3 assumes this role to replicate objects on your behalf. You create an IAM role in two steps:

- Create a role.
- Attach a permissions policy to the role.

- a. Create the IAM role.

- i. Copy the following trust policy and save it to a file named `s3-role-trust-policy.json` in the current directory on your local computer. This policy grants Amazon S3 service principal permissions to assume the role.

```
{  
    "Version": "2012-10-17",  
    "Statement": [  
        {  
            "Effect": "Allow",  
            "Principal": {  
                "Service": "s3.amazonaws.com"  
            },  
            "Action": "sts:AssumeRole"  
        }  
    ]  
}
```

- ii. Run the following command to create a role.

```
$ aws iam create-role \
--role-name replicationRole \
--assume-role-policy-document file://s3-role-trust-policy.json \
--profile acctA
```

- b. Attach a permissions policy to the role.

- i. Copy the following permissions policy and save it to a file named `s3-role-permissions-policy.json` in the current directory on your local computer. This policy grants permissions for various Amazon S3 bucket and object actions.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "s3:GetObjectVersionForReplication",
                "s3:GetObjectVersionAcl",
                "s3:GetObjectVersionTagging"
            ],
            "Resource": [
                "arn:aws:s3:::source-bucket/*"
            ]
        },
        {
            "Effect": "Allow",
            "Action": [
                "s3>ListBucket",
                "s3:GetReplicationConfiguration"
            ],
            "Resource": [
                "arn:aws:s3:::source-bucket"
            ]
        },
        {
            "Effect": "Allow",
            "Action": [
                "s3:ReplicateObject",
                "s3:ReplicateDelete",
                "s3:ReplicateTags"
            ],
            "Resource": "arn:aws:s3:::destination-bucket/*"
        }
    ]
}
```

- ii. Run the following command to create a policy and attach it to the role.

```
$ aws iam put-role-policy \
--role-name replicationRole \
--policy-document file://s3-role-permissions-policy.json \
--policy-name replicationRolePolicy \
--profile acctA
```

5. Add replication configuration to the `source` bucket.

- a. Although the Amazon S3 API requires replication configuration as XML, the AWS CLI requires that you specify the replication configuration as JSON. Save the following JSON in a file called `replication.json` to the local directory on your computer.

```
{  
    "Role": "IAM-role-ARN",  
    "Rules": [  
        {  
            "Status": "Enabled",  
            "Priority": 1,  
            "DeleteMarkerReplication": { "Status": "Disabled" },  
            "Filter" : { "Prefix": "Tax"},  
            "Destination": {  
                "Bucket": "arn:aws:s3:::destination-bucket"  
            }  
        }  
    ]  
}
```

- b. Update the JSON by providing values for the *destination-bucket* and *IAM-role-ARN*. Save the changes.
- c. Run the following command to add the replication configuration to your source bucket. Be sure to provide the *source* bucket name.

```
$ aws s3api put-bucket-replication \  
--replication-configuration file://replication.json \  
--bucket source \  
--profile acctA
```

To retrieve the replication configuration, use the `get-bucket-replication` command.

```
$ aws s3api get-bucket-replication \  
--bucket source \  
--profile acctA
```

6. Test the setup in the Amazon S3 console:

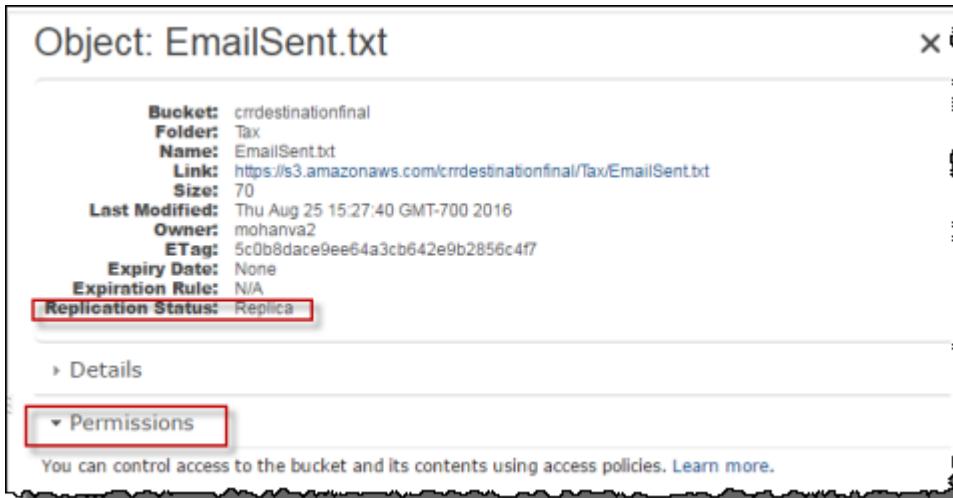
- a. Sign in to the AWS Management Console and open the Amazon S3 console at <https://console.aws.amazon.com/s3/>.
- b. In the *source* bucket, create a folder named Tax.
- c. Add sample objects to the Tax folder in the *source* bucket.

Note

The amount of time it takes for Amazon S3 to replicate an object depends on the size of the object. For information about how to see the status of replication, see [Getting replication status information \(p. 820\)](#).

In the *destination* bucket, verify the following:

- That Amazon S3 replicated the objects.
- In object **properties**, that the **Replication Status** is set to **Replica** (identifying this as a replica object).
- In object **properties**, that the permission section shows no permissions. This means that the replica is still owned by the *source* bucket owner, and the *destination* bucket owner has no permission on the object replica. You can add optional configuration to tell Amazon S3 to change the replica ownership. For an example, see [Changing the replica owner when source and destination buckets are owned by different accounts \(p. 786\)](#).



Using the AWS SDKs

Use the following code examples to add a replication configuration to a bucket with the AWS SDK for Java and AWS SDK for .NET, respectively.

Java

The following example adds a replication configuration to a bucket and then retrieves and verifies the configuration. For instructions on creating and testing a working sample, see [Testing the Amazon S3 Java Code Examples \(p. 1192\)](#).

```
import com.amazonaws.AmazonServiceException;
import com.amazonaws.SdkClientException;
import com.amazonaws.auth.profile.ProfileCredentialsProvider;
import com.amazonaws.regions.Regions;
import com.amazonaws.services.identitymanagement.AmazonIdentityManagement;
import com.amazonaws.services.identitymanagement.AmazonIdentityManagementClientBuilder;
import com.amazonaws.services.identitymanagement.model.CreateRoleRequest;
import com.amazonaws.services.identitymanagement.model.PutRolePolicyRequest;
import com.amazonaws.services.s3.AmazonS3;
import com.amazonaws.services.s3.AmazonS3Client;
import com.amazonaws.services.s3.model.BucketReplicationConfiguration;
import com.amazonaws.services.s3.model.BucketVersioningConfiguration;
import com.amazonaws.services.s3.model.CreateBucketRequest;
import com.amazonaws.services.s3.model.DeleteMarkerReplication;
import com.amazonaws.services.s3.model.DeleteMarkerReplicationStatus;
import com.amazonaws.services.s3.model.ReplicationDestinationConfig;
import com.amazonaws.services.s3.model.ReplicationRule;
import com.amazonaws.services.s3.model.ReplicationRuleStatus;
import com.amazonaws.services.s3.model.SetBucketVersioningConfigurationRequest;
import com.amazonaws.services.s3.model.StorageClass;
import com.amazonaws.services.s3.model.replication.ReplicationFilter;
import com.amazonaws.services.s3.model.replication.ReplicationFilterPredicate;
import com.amazonaws.services.s3.model.replication.ReplicationPrefixPredicate;

import java.io.IOException;
import java.util.ArrayList;
import java.util.HashMap;
import java.util.List;
```

```
import java.util.Map;

public class CrossRegionReplication {

    public static void main(String[] args) throws IOException {
        Regions clientRegion = Regions.DEFAULT_REGION;
        String accountId = "*** Account ID ***";
        String roleName = "*** Role name ***";
        String sourceBucketName = "*** Source bucket name ***";
        String destBucketName = "*** Destination bucket name ***";
        String prefix = "Tax/";

        String roleARN = String.format("arn:aws:iam::%s:%s", accountId, roleName);
        String destinationBucketARN = "arn:aws:s3:::" + destBucketName;

        AmazonS3 s3Client = AmazonS3Client.builder()
            .withCredentials(new ProfileCredentialsProvider())
            .withRegion(clientRegion)
            .build();

        createBucket(s3Client, clientRegion, sourceBucketName);
        createBucket(s3Client, clientRegion, destBucketName);
        assignRole(roleName, clientRegion, sourceBucketName, destBucketName);

        try {

            // Create the replication rule.
            List<ReplicationFilterPredicate> andOperands = new
            ArrayList<ReplicationFilterPredicate>();
            andOperands.add(new ReplicationPrefixPredicate(prefix));

            Map<String, ReplicationRule> replicationRules = new HashMap<String,
            ReplicationRule>();
            replicationRules.put("ReplicationRule1",
                new ReplicationRule()
                    .withPriority(0)
                    .withStatus(ReplicationRuleStatus.Enabled)
                    .withDeleteMarkerReplication(new
                    DeleteMarkerReplication().withStatus(DeleteMarkerReplicationStatus.DISABLED))
                    .withFilter(new ReplicationFilter().withPredicate(new
                    ReplicationPrefixPredicate(prefix)))
                    .withDestinationConfig(new ReplicationDestinationConfig()
                        .withBucketARN(destinationBucketARN)
                        .withStorageClass(StorageClass.Standard)));

            // Save the replication rule to the source bucket.
            s3Client.setBucketReplicationConfiguration(sourceBucketName,
                new BucketReplicationConfiguration()
                    .withRoleARN(roleARN)
                    .withRules(replicationRules));

            // Retrieve the replication configuration and verify that the configuration
            // matches the rule we just set.
            BucketReplicationConfiguration replicationConfig =
            s3Client.getBucketReplicationConfiguration(sourceBucketName);
            ReplicationRule rule = replicationConfig.getRule("ReplicationRule1");
            System.out.println("Retrieved destination bucket ARN: " +
rule.getDestinationConfig().getBucketARN());
            System.out.println("Retrieved priority: " + rule.getPriority());
            System.out.println("Retrieved source-bucket replication rule status: " +
rule.getStatus());
        } catch (AmazonServiceException e) {
            // The call was transmitted successfully, but Amazon S3 couldn't process
        }
    }
}
```

```

        // it, so it returned an error response.
        e.printStackTrace();
    } catch (SdkClientException e) {
        // Amazon S3 couldn't be contacted for a response, or the client
        // couldn't parse the response from Amazon S3.
        e.printStackTrace();
    }
}

private static void createBucket(AmazonS3 s3Client, Regions region, String
bucketName) {
    CreateBucketRequest request = new CreateBucketRequest(bucketName,
region.getName());
    s3Client.createBucket(request);
    BucketVersioningConfiguration configuration = new
BucketVersioningConfiguration().withStatus(BucketVersioningConfiguration.ENABLED);

    SetBucketVersioningConfigurationRequest enableVersioningRequest = new
SetBucketVersioningConfigurationRequest(bucketName, configuration);
    s3Client.setBucketVersioningConfiguration(enableVersioningRequest);

}

private static void assignRole(String roleName, Regions region, String
sourceBucket, String destinationBucket) {
    AmazonIdentityManagement iamClient =
AmazonIdentityManagementClientBuilder.standard()
        .withRegion(region)
        .withCredentials(new ProfileCredentialsProvider())
        .build();
    StringBuilder trustPolicy = new StringBuilder();
    trustPolicy.append("{\r\n  ");
    trustPolicy.append("\\\"Version\\\":\\\"2012-10-17\\\",\\r\\n  ");
    trustPolicy.append("\\\"Statement\\\":[\r\n    {\r\n      \"Effect\\\":\\\"Allow\\\",\\r\\n      \"Principal\\\":
{\r\n        \"AWS\\\":\\\""+destinationBucket+"\\\"},\\r\\n      \"Action\\\":\\\"sts:AssumeRole\\\",\\r\\n    }\\r\\n  ];
\\r\\n}");\\r\\n

    CreateRoleRequest createRoleRequest = new CreateRoleRequest()
        .withRoleName(roleName)
        .withAssumeRolePolicyDocument(trustPolicy.toString());

    iamClient.createRole(createRoleRequest);

    StringBuilder permissionPolicy = new StringBuilder();
    permissionPolicy.append("{\r\n  \"Version\\\":\\\"2012-10-17\\\",\\r\\n
\\\"Statement\\\":[\r\n    {\r\n      \"Effect\\\":\\\"Allow\\\",\\r\\n      \"Action\\\":\\\"s3:GetObjectVersionForReplication\\\",\\r\\n
    };
    permissionPolicy.append(" \\\"s3:GetObjectVersionAcl\\\"\\r\\n      ,\\r\\n      \"Resource\\\":[\r\n        {\r\n          \"arn:aws:s3:::");
    permissionPolicy.append(sourceBucket);
    permissionPolicy.append("/*\\\"\\r\\n      ]\\r\\n      ,\\r\\n      {\r\n        \"");
    permissionPolicy.append("Effect\\\":\\\"Allow\\\",\\r\\n      }\\r\\n    );
    permissionPolicy.append(" \\\"Effect\\\":\\\"Allow\\\",\\r\\n      \"Action\\\":\\\"s3>ListBucket\\\",\\r\\n      \"s3:GetReplicationConfiguration\\\"\\r\\n      ");
\\r\\n
}

```

```

        permissionPolicy.append("],\\r\\n          \\"Resource\\\":[\\r\\n          \
\\\"arn:aws:s3:::");
        permissionPolicy.append(sourceBucket);
        permissionPolicy.append("\\r\\n          ");
        permissionPolicy.append("]\\r\\n          },\\r\\n          {\\r\\n          \\\"Effect\\
\\\":\\\"Allow\\\",\\r\\n          ");
        permissionPolicy.append("\\\"Action\\\":[\\r\\n          \\
\\\"s3:ReplicateObject\\\",\\r\\n          ");
        permissionPolicy.append("\\\"s3:ReplicateDelete\\\",\\r\\n          \\
\\\"s3:ReplicateTags\\\",\\r\\n          ");
        permissionPolicy.append("\\\"s3:GetObjectVersionTagging\\\"\\r\\n          \\
],\\r\\n          ");
        permissionPolicy.append("\\\"Resource\\\":\\\"arn:aws:s3:::");
        permissionPolicy.append(destinationBucket);
        permissionPolicy.append("/*\\\"\\r\\n          }\\r\\n      ]\\r\\n}"));

PutRolePolicyRequest putRolePolicyRequest = new PutRolePolicyRequest()
    .withRoleName(roleName)
    .withPolicyDocument(permissionPolicy.toString())
    .withPolicyName("crrRolePolicy");

iamClient.putRolePolicy(putRolePolicyRequest);

}
}

```

C#

The following AWS SDK for .NET code example adds a replication configuration to a bucket and then retrieves it. To use this code, provide the names for your buckets and the Amazon Resource Name (ARN) for your IAM role. For instructions on how to create and test a working sample, see [Running the Amazon S3 .NET Code Examples \(p. 1193\)](#).

```

using Amazon;
using Amazon.S3;
using Amazon.S3.Model;
using System;
using System.Threading.Tasks;

namespace Amazon.DocSamples.S3
{
    class CrossRegionReplicationTest
    {
        private const string sourceBucket = "*** source bucket ***";
        // Bucket ARN example - arn:aws:s3:::destinationbucket
        private const string destinationBucketArn = "*** destination bucket ARN ***";
        private const string roleArn = "*** IAM Role ARN ***";
        // Specify your bucket region (an example region is shown).
        private static readonly RegionEndpoint sourceBucketRegion =
RegionEndpoint.USWest2;
        private static IAmazonS3 s3Client;
        public static void Main()
        {
            s3Client = new AmazonS3Client(sourceBucketRegion);
            EnableReplicationAsync().Wait();
        }
        static async Task EnableReplicationAsync()
        {
            try
            {
                ReplicationConfiguration replConfig = new ReplicationConfiguration
                {

```

```
Role = roleArn,
Rules =
{
    new ReplicationRule
    {
        Prefix = "Tax",
        Status = ReplicationRuleStatus.Enabled,
        Destination = new ReplicationDestination
        {
            BucketArn = destinationBucketArn
        }
    }
};

PutBucketReplicationRequest putRequest = new
PutBucketReplicationRequest
{
    BucketName = sourceBucket,
    Configuration = replConfig
};

PutBucketReplicationResponse putResponse = await
s3Client.PutBucketReplicationAsync(putRequest);

// Verify configuration by retrieving it.
await RetrieveReplicationConfigurationAsync(s3Client);
}
catch (AmazonS3Exception e)
{
    Console.WriteLine("Error encountered on server. Message:{0} when
writing an object", e.Message);
}
catch (Exception e)
{
    Console.WriteLine("Unknown encountered on server. Message:{0} when
writing an object", e.Message);
}
private static async Task RetrieveReplicationConfigurationAsync(IAmazonS3
client)
{
    // Retrieve the configuration.
    GetBucketReplicationRequest getRequest = new GetBucketReplicationRequest
    {
        BucketName = sourceBucket
    };
    GetBucketReplicationResponse getResponse = await
client.GetBucketReplicationAsync(getRequest);
    // Print.
    Console.WriteLine("Printing replication configuration information...");
    Console.WriteLine("Role ARN: {0}", getResponse.Configuration.Role);
    foreach (var rule in getResponse.Configuration.Rules)
    {
        Console.WriteLine("ID: {0}", rule.Id);
        Console.WriteLine("Prefix: {0}", rule.Prefix);
        Console.WriteLine("Status: {0}", rule.Status);
    }
}
```

Configuring replication when source and destination buckets are owned by different accounts

Setting up replication when *source* and *destination* buckets are owned by different AWS accounts is similar to setting replication when both buckets are owned by the same account. The only difference is that the *destination* bucket owner must grant the *source* bucket owner permission to replicate objects by adding a bucket policy.

For more information about configuring replication using server-side encryption with AWS Key Management Service in cross-account scenarios, see [Granting additional permissions for cross-account scenarios \(p. 819\)](#).

To configure replication when the source and destination buckets are owned by different AWS accounts

1. In this example, you create *source* and *destination* buckets in two different AWS accounts. You need to have two credential profiles set for the AWS CLI (in this example, we use acctA and acctB for profile names). For more information about setting credential profiles, see [Named Profiles](#) in the *AWS Command Line Interface User Guide*.
2. Follow the step-by-step instructions in [Configuring for buckets in the same account \(p. 773\)](#) with the following changes:
 - For all AWS CLI commands related to *source* bucket activities (for creating the *source* bucket, enabling versioning, and creating the IAM role), use the acctA profile. Use the acctB profile to create the *destination* bucket.
 - Make sure that the permissions policy specifies the *source* and *destination* buckets that you created for this example.
3. In the console, add the following bucket policy on the *destination* bucket to allow the owner of the *source* bucket to replicate objects. Be sure to edit the policy by providing the AWS account ID of the *source* bucket owner and the *destination* bucket name.

```
{  
    "Version": "2012-10-17",  
    "Id": "",  
    "Statement": [  
        {  
            "Sid": "Set permissions for objects",  
            "Effect": "Allow",  
            "Principal": {  
                "AWS": "arn:aws:iam::source-bucket-acct-ID:role/service-role/source-acct-IAM-role"  
            },  
            "Action": ["s3:ReplicateObject", "s3:ReplicateDelete"],  
            "Resource": "arn:aws:s3:::DOC-EXAMPLE-BUCKET/*"  
        },  
        {  
            "Sid": "Set permissions on bucket",  
            "Effect": "Allow",  
            "Principal": {  
                "AWS": "arn:aws:iam::source-bucket-acct-ID:role/service-role/source-acct-IAM-role"  
            },  
            "Action": ["s3>List*", "s3:GetBucketVersioning", "s3:PutBucketVersioning"],  
            "Resource": "arn:aws:s3:::DOC-EXAMPLE-BUCKET"  
        }  
    ]  
}
```

Choose the bucket and add the bucket policy. For instructions, see [Adding a bucket policy using the Amazon S3 console \(p. 488\)](#).

In replication, the owner of the source object owns the replica by default. When source and destination buckets are owned by different AWS accounts, you can add optional configuration settings to change replica ownership to the AWS account that owns the destination buckets. This includes granting the `ObjectOwnerOverrideToBucketOwner` permission. For more information, see [Changing the replica owner \(p. 812\)](#).

Changing the replica owner when source and destination buckets are owned by different accounts

When the `source` and `destination` buckets in a replication configuration are owned by different AWS accounts, you can tell Amazon S3 to change replica ownership to the AWS account that owns the `destination` bucket. This example explains how to use the Amazon S3 console and the AWS CLI to change replica ownership. For more information, see [Changing the replica owner \(p. 812\)](#).

Note

When you use S3 replication and the source and destination buckets are owned by different AWS accounts, the bucket owner of the destination bucket can disable ACLs (with the bucket owner enforced setting for Object Ownership) to change replica ownership to the AWS account that owns the destination bucket. This setting mimics the existing owner override behavior without the need of `s3:ObjectOwnerOverrideToBucketOwner` permission. This means that all objects that are replicated to the destination bucket with the bucket owner enforced setting are owned by the destination bucket owner. For more information about Object Ownership, see [Controlling ownership of objects and disabling ACLs for your bucket \(p. 601\)](#).

For more information about configuring replication using sever-side encryption with AWS Key Management Service in cross-account scenarios, see [Granting additional permissions for cross-account scenarios \(p. 819\)](#).

Using the S3 console

For step-by-step instructions, see [Configuring replication for source and destination buckets owned by the same account \(p. 773\)](#). This topic provides instructions for setting replication configuration when buckets are owned by same and different AWS accounts.

Using the AWS CLI

To change replica ownership using the AWS CLI, you create buckets, enable versioning on the buckets, create an IAM role that gives Amazon S3 permission to replicate objects, and add the replication configuration to the source bucket. In the replication configuration you direct Amazon S3 to change replica owner. You also test the setup.

To change replica ownership when source and destination buckets are owned by different AWS accounts (AWS CLI)

1. In this example, you create the `source` and `destination` buckets in two different AWS accounts. Configure the AWS CLI with two named profiles. This example uses profiles named `acctA` and `acctB`, respectively. For more information about setting credential profiles, see [Named Profiles in the AWS Command Line Interface User Guide](#).

Important

The profiles you use for this exercise must have the necessary permissions. For example, in the replication configuration, you specify the IAM role that Amazon S3 can assume. You can do this only if the profile you use has the `iam:PassRole` permission. If you use administrator user credentials to create a named profile then you can perform all the tasks. For more information, see [Granting a User Permissions to Pass a Role to an AWS Service](#) in the [IAM User Guide](#).

You will need to make sure these profiles have necessary permissions. For example, the replication configuration includes an IAM role that Amazon S3 can assume. The named profile you use to attach such configuration to a bucket can do so only if it has the `iam:PassRole` permission. If you specify administrator user credentials when creating these named profiles, they have all the permissions. For more information, see [Granting a User Permissions to Pass a Role to an AWS Service in the IAM User Guide](#).

2. Create the `source` bucket and enable versioning. This example creates the `source` bucket in the US East (N. Virginia) (us-east-1) Region.

```
aws s3api create-bucket \  
--bucket source \  
--region us-east-1 \  
--profile acctA
```

```
aws s3api put-bucket-versioning \  
--bucket source \  
--versioning-configuration Status=Enabled \  
--profile acctA
```

3. Create a `destination` bucket and enable versioning. This example creates the `destination` bucket in the US West (Oregon) (us-west-2) Region. Use an AWS account profile different from the one you used for the `source` bucket.

```
aws s3api create-bucket \  
--bucket destination \  
--region us-west-2 \  
--create-bucket-configuration LocationConstraint=us-west-2 \  
--profile acctB
```

```
aws s3api put-bucket-versioning \  
--bucket destination \  
--versioning-configuration Status=Enabled \  
--profile acctB
```

4. You must add permissions to your `destination` bucket policy to allow changing the replica ownership.

- a. Save the following policy to `destination-bucket-policy.json`.

```
{  
    "Version": "2012-10-17",  
    "Statement": [  
        {  
            "Sid": "destination_bucket_policy_sid",  
            "Principal": {  
                "AWS": "source-bucket-owner-account-id"  
            },  
            "Action": [  
                "s3:ReplicateObject",  
                "s3:ReplicateDelete",  
                "s3:ObjectOwnerOverrideToBucketOwner",  
                "s3:ReplicateTags",  
                "s3:GetObjectVersionTagging"  
            ],  
            "Effect": "Allow",  
            "Resource": [  
                "arn:aws:s3:::destination/*"  
            ]  
        }  
    ]  
}
```

```
        ]  
    }  
}
```

- b. Put the above policy to *destination* bucket:

```
aws s3api put-bucket-policy --region ${destination_region} --  
bucket ${destination} --policy file:///${destination_bucket_policy}.json
```

5. Create an IAM role. You specify this role in the replication configuration that you add to the *source* bucket later. Amazon S3 assumes this role to replicate objects on your behalf. You create an IAM role in two steps:

- Create a role.
- Attach a permissions policy to the role.

- a. Create an IAM role.

- i. Copy the following trust policy and save it to a file named `s3-role-trust-policy.json` in the current directory on your local computer. This policy grants Amazon S3 permissions to assume the role.

```
{  
    "Version": "2012-10-17",  
    "Statement": [  
        {  
            "Effect": "Allow",  
            "Principal": {  
                "Service": "s3.amazonaws.com"  
            },  
            "Action": "sts:AssumeRole"  
        }  
    ]  
}
```

- ii. Run the following AWS CLI command to create a role.

```
$ aws iam create-role \  
--role-name replicationRole \  
--assume-role-policy-document file://s3-role-trust-policy.json \  
--profile accta
```

- b. Attach a permissions policy to the role.

- i. Copy the following permissions policy and save it to a file named `s3-role-perm-pol-changeowner.json` in the current directory on your local computer. This policy grants permissions for various Amazon S3 bucket and object actions. In the following steps, you create an IAM role and attach this policy to the role.

```
{  
    "Version": "2012-10-17",  
    "Statement": [  
        {  
            "Effect": "Allow",  
            "Action": [  
                "s3:GetObjectVersionForReplication",  
                "s3:GetObjectVersionAcl"  
            ],  
            "Resource": [  
                "arn:aws:s3:::sourceBucketName/*"  
            ]  
        }  
    ]  
}
```

```

        "arn:aws:s3:::source/*"
    ],
},
{
    "Effect":"Allow",
    "Action":[
        "s3>ListBucket",
        "s3:GetReplicationConfiguration"
    ],
    "Resource":[
        "arn:aws:s3:::source"
    ],
{
    "Effect":"Allow",
    "Action":[
        "s3:ReplicateObject",
        "s3:ReplicateDelete",
        "s3:ObjectOwnerOverrideToBucketOwner",
        "s3:ReplicateTags",
        "s3:GetObjectVersionTagging"
    ],
    "Resource": "arn:aws:s3:::destination/*"
}
]
}
}

```

- ii. To create a policy and attach it to the role, run the following command.

```

$ aws iam put-role-policy \
--role-name replicationRole \
--policy-document file://s3-role-perm-pol-changeowner.json \
--policy-name replicationRolechangeownerPolicy \
--profile accta

```

6. Add a replication configuration to your source bucket.

- a. The AWS CLI requires specifying the replication configuration as JSON. Save the following JSON in a file named `replication.json` in the current directory on your local computer. In the configuration, the addition of `AccessControlTranslation` to indicate change in replica ownership.

```

{
    "Role": "IAM-role-ARN",
    "Rules": [
        {
            "Status": "Enabled",
            "Priority": 1,
            "DeleteMarkerReplication": {
                "Status": "Disabled"
            },
            "Filter": {
            },
            "Status": "Enabled",
            "Destination": {
                "Bucket": "arn:aws:s3:::destination",
                "Account": "destination-bucket-owner-account-id",
                "AccessControlTranslation": {
                    "Owner": "Destination"
                }
            }
        }
    ]
}

```

}

- b. Edit the JSON by providing values for the *destination* bucket owner account ID and *IAM-role-ARN*. Save the changes.
- c. To add the replication configuration to the source bucket, run the following command. Provide the *source* bucket name.

```
$ aws s3api put-bucket-replication \
--replication-configuration file://replication.json \
--bucket source \
--profile acctA
```

7. Check replica ownership in the Amazon S3 console.

- a. Sign in to the AWS Management Console and open the Amazon S3 console at <https://console.aws.amazon.com/s3/>.
- b. Add objects to the *source* bucket. Verify that the *destination* bucket contains the object replicas and that the ownership of the replicas has changed to the AWS account that owns the *destination* bucket.

Using the AWS SDKs

For a code example to add replication configuration, see [Using the AWS SDKs \(p. 780\)](#). You need to modify the replication configuration appropriately. For conceptual information, see [Changing the replica owner \(p. 812\)](#).

Replicating encrypted objects

By default, Amazon S3 doesn't replicate objects that are stored at rest using server-side encryption with KMS keys. To replicate encrypted objects, you modify the bucket replication configuration to tell Amazon S3 to replicate these objects. This example explains how to use the Amazon S3 console and the AWS Command Line Interface (AWS CLI) to change the bucket replication configuration to enable replicating encrypted objects. For more information, see [Replicating objects created with server-side encryption \(SSE\) using KMS keys \(p. 814\)](#).

Note

You can use multi-Region AWS KMS keys in Amazon S3. However, Amazon S3 currently treats multi-Region keys as though they were single-Region keys, and does not use the multi-Region features of the key. For more information, see [Using multi-Region keys in AWS Key Management Service Developer Guide](#).

Using the S3 console

For step-by-step instructions, see [Configuring replication for source and destination buckets owned by the same account \(p. 773\)](#). This topic provides instructions for setting replication configuration when buckets are owned by same and different AWS accounts.

Using the AWS CLI

To replicate encrypted objects with the AWS CLI, you create buckets, enable versioning on the buckets, create an IAM role that gives Amazon S3 permission to replicate objects, and add the replication configuration to the source bucket. The replication configuration provides information related to replicating objects encrypted using KMS keys. The IAM role permissions include necessary permissions to replicate the encrypted objects. You also test the setup.

To replicate server-side encrypted objects (AWS CLI)

1. In this example, we create both the *source* and *destination* buckets in the same AWS account. Set a credentials profile for the AWS CLI. In this example, we use the profile name acctA. For

more information about setting credential profiles, see [Named Profiles](#) in the AWS Command Line Interface User Guide.

2. Create the `source` bucket and enable versioning on it. In this example, we create the `source` bucket in the US East (N. Virginia) (us-east-1) Region.

```
aws s3api create-bucket \
--bucket source \
--region us-east-1 \
--profile acctA
```

```
aws s3api put-bucket-versioning \
--bucket source \
--versioning-configuration Status=Enabled \
--profile acctA
```

3. Create the `destination` bucket and enable versioning on it. In this example, we create the `destination` bucket in the US West (Oregon) (us-west-2) Region.

Note

To set up replication configuration when both `source` and `destination` buckets are in the same AWS account, you use the same profile. In this example, we use `acctA`. To test replication configuration when the buckets are owned by different AWS accounts, you specify different profiles for each.

```
aws s3api create-bucket \
--bucket destination \
--region us-west-2 \
--create-bucket-configuration LocationConstraint=us-west-2 \
--profile acctA
```

```
aws s3api put-bucket-versioning \
--bucket destination \
--versioning-configuration Status=Enabled \
--profile acctA
```

4. Create an IAM role. You specify this role in the replication configuration that you add to the `source` bucket later. Amazon S3 assumes this role to replicate objects on your behalf. You create an IAM role in two steps:

- Create a role
- Attach a permissions policy to the role

- a. Create an IAM role.

- i. Copy the following trust policy and save it to a file called `s3-role-trust-policy-kmsobj.json` in the current directory on your local computer. This policy grants Amazon S3 service principal permissions to assume the role so Amazon S3 can perform tasks on your behalf.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Principal": {
                "Service": "s3.amazonaws.com"
            }
        }
    ]
}
```

```
        "Action":"sts:AssumeRole"
    }
}
```

- ii. Create a role.

```
$ aws iam create-role \
--role-name replicationRolekmsobj \
--assume-role-policy-document file://s3-role-trust-policy-kmsobj.json \
--profile acctA
```

- b. Attach a permissions policy to the role. This policy grants permissions for various Amazon S3 bucket and object actions.
- i. Copy the following permissions policy and save it to a file named s3-role-permissions-policykmsobj.json in the current directory on your local computer. You create an IAM role and attach the policy to it later.

Important

In the permissions policy, you specify the AWS KMS key IDs that will be used for encryption of *source* and destination buckets. You must create two separate KMS keys for the source and destination buckets. AWS KMS keys are never shared outside the AWS Region in which they were created.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Action": [
                "s3>ListBucket",
                "s3:GetReplicationConfiguration",
                "s3GetObjectVersionForReplication",
                "s3GetObjectVersionAcl",
                "s3GetObjectVersionTagging"
            ],
            "Effect": "Allow",
            "Resource": [
                "arn:aws:s3:::source",
                "arn:aws:s3:::source/*"
            ]
        },
        {
            "Action": [
                "s3:ReplicateObject",
                "s3:ReplicateDelete",
                "s3:ReplicateTags"
            ],
            "Effect": "Allow",
            "Condition": {
                "StringLikeIfExists": {
                    "s3:x-amz-server-side-encryption": [
                        "aws:kms",
                        "AES256"
                    ],
                    "s3:x-amz-server-side-encryption-aws-kms-key-id": [
                        "AWS KMS key IDs(in ARN format) to use for encrypting object replicas"
                    ]
                }
            },
            "Resource": "arn:aws:s3:::destination/*"
        },
    ],
}
```

```
{
    "Action": [
        "kms:Decrypt"
    ],
    "Effect": "Allow",
    "Condition": {
        "StringLike": {
            "kms:ViaService": "s3.us-east-1.amazonaws.com",
            "kms:EncryptionContext:aws:s3:arn": [
                "arn:aws:s3:::/*"
            ]
        }
    },
    "Resource": [
        "AWS KMS key IDs(in ARN format) used to encrypt source objects."
    ]
},
{
    "Action": [
        "kms:Encrypt"
    ],
    "Effect": "Allow",
    "Condition": {
        "StringLike": {
            "kms:ViaService": "s3.us-west-2.amazonaws.com",
            "kms:EncryptionContext:aws:s3:arn": [
                "arn:aws:s3:::/*"
            ]
        }
    },
    "Resource": [
        "AWS KMS key IDs(in ARN format) to use for encrypting object replicas"
    ]
}
}
```

- ii. Create a policy and attach it to the role.

```
$ aws iam put-role-policy \
--role-name replicationRolekmsobj \
--policy-document file://s3-role-permissions-policykmsobj.json \
--policy-name replicationRolechangeownerPolicy \
--profile accta
```

5. Add the following replication configuration to the **source** bucket. It tells Amazon S3 to replicate objects with the **Tax/** prefix to the **destination** bucket.

Important

In the replication configuration you specify the IAM role that Amazon S3 can assume. You can do this only if you have the `iam:PassRole` permission. The profile you specify in the CLI command must have the permission. For more information, see [Granting a User Permissions to Pass a Role to an AWS Service](#) in the *IAM User Guide*.

```
<ReplicationConfiguration>
<Role>IAM-Role-ARN</Role>
<Rule>
    <Priority>1</Priority>
    <DeleteMarkerReplication>
        <Status>Disabled</Status>
    </DeleteMarkerReplication>
    <Filter>
        <Prefix>Tax</Prefix>
```

```

</Filter>
<Status>Enabled</Status>
<SourceSelectionCriteria>
  <SseKmsEncryptedObjects>
    <Status>Enabled</Status>
  </SseKmsEncryptedObjects>
</SourceSelectionCriteria>
<Destination>
  <Bucket>arn:aws:s3:::dest-bucket-name</Bucket>
  <EncryptionConfiguration>
    <ReplicaKmsKeyId>AWS KMS key IDs to use for encrypting object replicas</ReplicaKmsKeyId>
  </EncryptionConfiguration>
</Destination>
</Rule>
</ReplicationConfiguration>

```

To add replication configuration to the *source* bucket, do the following:

- The AWS CLI requires you to specify the replication configuration as JSON. Save the following JSON in a file (*replication.json*) in the current directory on your local computer.

```

{
  "Role": "IAM-Role-ARN",
  "Rules": [
    {
      "Status": "Enabled",
      "Priority": 1,
      "DeleteMarkerReplication": {
        "Status": "Disabled"
      },
      "Filter": {
        "Prefix": "Tax"
      },
      "Destination": {
        "Bucket": "arn:aws:s3:::destination",
        "EncryptionConfiguration": {
          "ReplicaKmsKeyId": "AWS KMS key IDs(in ARN format) to use for encrypting object replicas"
        }
      },
      "SourceSelectionCriteria": {
        "SseKmsEncryptedObjects": {
          "Status": "Enabled"
        }
      }
    }
  ]
}

```

- Edit the JSON to provide values for the *destination* bucket, *KMS ID ARN* and *IAM-role-ARN*. Save the changes.
- Add the replication configuration to your *source* bucket. Be sure to provide the *source* bucket name.

```

$ aws s3api put-bucket-replication \
--replication-configuration file://replication.json \
--bucket source \
--profile acctA

```

- Test the setup to verify that encrypted objects are replicated. In the Amazon S3 console:

- a. Sign in to the AWS Management Console and open the Amazon S3 console at <https://console.aws.amazon.com/s3/>.
- b. In the *source* bucket, create a folder named `Tax`.
- c. Add sample objects to the folder. Be sure to choose the encryption option and specify your KMS key to encrypt the objects.
- d. Verify that the *destination* bucket contains the object replicas and that they are encrypted using the KMS key that you specified in the configuration.

Using the AWS SDKs

For a code example to add replication configuration, see [Using the AWS SDKs \(p. 780\)](#). You need to modify the replication configuration appropriately.

For conceptual information, see [Replicating objects created with server-side encryption \(SSE\) using KMS keys \(p. 814\)](#).

Replicating objects with S3 Replication Time Control (S3 RTC)

S3 Replication Time Control (S3 RTC) helps you meet compliance or business requirements for data replication and provides visibility into Amazon S3 replication times. S3 RTC replicates most objects that you upload to Amazon S3 in seconds, and 99.99 percent of those objects within 15 minutes.

With S3 RTC, you can monitor the total number and size of objects that are pending replication, and the maximum replication time to the destination Region. Replication metrics are available through the [AWS Management Console](#) and [Amazon CloudWatch User Guide](#). For more information, see [the section called "Amazon S3 replication metrics in CloudWatch" \(p. 1007\)](#).

Using the S3 console

For step-by-step instructions, see [Configuring replication for source and destination buckets owned by the same account \(p. 773\)](#). This topic provides instructions for enabling S3 RTC in your replication configuration when buckets are owned by same and different AWS accounts.

Using the AWS CLI

To use the AWS CLI to replicate objects with S3 RTC enabled, you create buckets, enable versioning on the buckets, create an IAM role that gives Amazon S3 permission to replicate objects, and add the replication configuration to the source bucket. The replication configuration needs to have S3 Replication Time Control (S3 RTC) enabled.

To replicate with S3 RTC enabled (AWS CLI)

- The following example sets `ReplicationTime` and `Metric`, and adds replication configuration to the source bucket.

```
{  
    "Rules": [  
        {  
            "Status": "Enabled",  
            "Filter": {  
                "Prefix": "Tax"  
            },  
            "DeleteMarkerReplication": {  
                "Status": "Disabled"  
            },  
            "Destination": {  
                "Bucket": "arn:aws:s3:::destination",  
                "Metrics": {  
                    "Status": "Enabled",  
                    "MetricName": "Standard",  
                    "MetricsToPut": ["ObjectReplicated", "ObjectCopied", "ObjectSize"],  
                    "MetricsUnit": "Bytes",  
                    "MetricsFormat": "JSON",  
                    "MetricsEncryptionType": "AES256",  
                    "MetricsRetentionDays": 30,  
                    "MetricsNonCompliantObjectReplication": "Disabled"  
                }  
            }  
        }  
    ]  
}
```

```
        "EventThreshold": {
            "Minutes": 15
        }
    },
    "ReplicationTime": {
        "Status": "Enabled",
        "Time": {
            "Minutes": 15
        }
    }
},
"Priority": 1
}
],
"Role": "IAM-Role-ARN"
}
```

Important

Metrics:EventThreshold:Minutes and ReplicationTime:Time:Minutes can only have 15 as a valid value.

Using the AWS SDK for Java

The following Java example adds replication configuration with S3 Replication Time Control (S3 RTC).

```
import software.amazon.awssdk.auth.credentials.AwsBasicCredentials;
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.s3.model.DeleteMarkerReplication;
import software.amazon.awssdk.services.s3.model.Destination;
import software.amazon.awssdk.services.s3.model.Metrics;
import software.amazon.awssdk.services.s3.model.MetricsStatus;
import software.amazon.awssdk.services.s3.model.PutBucketReplicationRequest;
import software.amazon.awssdk.services.s3.model.ReplicationConfiguration;
import software.amazon.awssdk.services.s3.model.ReplicationRule;
import software.amazon.awssdk.services.s3.model.ReplicationRuleFilter;
import software.amazon.awssdk.services.s3.model.ReplicationTime;
import software.amazon.awssdk.services.s3.model.ReplicationTimeStatus;
import software.amazon.awssdk.services.s3.model.ReplicationTimeValue;

public class Main {

    public static void main(String[] args) {
        S3Client s3 = S3Client.builder()
            .region(Region.US_EAST_1)
            .credentialsProvider(() -> AwsBasicCredentials.create(
                "AWS_ACCESS_KEY_ID",
                "AWS_SECRET_ACCESS_KEY")
        )
        .build();

        ReplicationConfiguration replicationConfig = ReplicationConfiguration
            .builder()
            .rules(
                ReplicationRule
                    .builder()
                    .status("Enabled")
                    .priority(1)
                    .deleteMarkerReplication(
                        DeleteMarkerReplication
                            .builder()
                            .status("Disabled")
                            .build()
                    )
            )
    }
}
```

```
.destination(
    Destination
        .builder()
        .bucket("destination_bucket_arn")
        .replicationTime(
            ReplicationTime.builder().time(
                ReplicationTimeValue.builder().minutes(15).build()
            ).status(
                ReplicationTimeStatus.ENABLED
            ).build()
        )
        .metrics(
            Metrics.builder().eventThreshold(
                ReplicationTimeValue.builder().minutes(15).build()
            ).status(
                MetricsStatus.ENABLED
            ).build()
        )
        .build()
)
.filter(
    ReplicationRuleFilter
        .builder()
        .prefix("testtest")
        .build()
)
.build())
.role("role_arn")
.build();

// Put replication configuration
PutBucketReplicationRequest putBucketReplicationRequest = PutBucketReplicationRequest
    .builder()
    .bucket("source_bucket")
    .replicationConfiguration(replicationConfig)
    .build();

s3.putBucketReplication(putBucketReplicationRequest);
}
```

For more information, see [Meeting compliance requirements using S3 Replication Time Control \(S3 RTC\) \(p. 807\)](#).

Managing replication rules using the Amazon S3 console

Replication is the automatic, asynchronous copying of objects across buckets in the same or different AWS Regions. It replicates newly created objects and object updates from a source bucket to a specified destination bucket.

You use the Amazon S3 console to add replication rules to the source bucket. Replication rules define the source bucket objects to replicate and the destination bucket or buckets where the replicated objects are stored. For more information about replication, see [Replicating objects \(p. 753\)](#).

You can manage replication rules on the **Replication** page. You can add, view, enable, disable, delete, and change the priority of the replication rules. For information about adding replication rules to a bucket, see [Using the S3 console \(p. 773\)](#).

To manage the replication rules for an S3 bucket

1. Sign in to the AWS Management Console and open the Amazon S3 console at <https://console.aws.amazon.com/s3/>.

2. In the **Buckets** list, choose the name of the bucket that you want.
3. Choose **Management**, and then scroll down to **Replication rules**.
4. You change the replication rules in the following ways.
 - To enable or disable a replication rule, select the rule, choose **Actions**, and in the drop-down list, choose **Enable rule** or **Disable rule**. You can also disable, enable, or delete all the rules in the bucket from the **Actions** drop-down list.
 - To change the rule priorities, select the rule and choose **Edit**, which starts the Replication wizard to help you make the change. For information about using the wizard, see [Using the S3 console \(p. 773\)](#).

You set rule priorities to avoid conflicts caused by objects that are included in the scope of more than one rule. In the case of overlapping rules, Amazon S3 uses the rule priority to determine which rule to apply. The higher the number, the higher the priority. For more information about rule priority, see [Replication configuration \(p. 759\)](#).

Replicating existing objects with S3 Batch Replication

S3 Batch Replication provides you a way to replicate objects that existed before a replication configuration was in place, objects that have previously been replicated, and objects that have failed replication. This is done through the use of a Batch Operations job. This differs from live replication which continuously and automatically replicates new objects across Amazon S3 buckets. To get started with Batch Replication you may:

- **Initiate Batch Replication for a new replication rule or destination** – You may create a one-time Batch Replication job when creating the first rule in a new replication configuration or when adding a new destination to an existing configuration through the AWS Management Console.
- **Initiate Batch Replication for an existing replication configuration** – You can create a new Batch Replication job using **S3 Batch Operations** through the AWS SDKs, AWS Command Line Interface (AWS CLI), or the Amazon S3 console.

When the Batch Replication job finishes, you receive a completion report. For more information about how to use the report to examine the job, see [Tracking job status and completion reports \(p. 922\)](#).

S3 Batch Replication considerations

- Your source bucket must have an existing replication configuration. To enable replication, see [Setting up replication \(p. 758\)](#) and [Walkthroughs: Examples for configuring replication \(p. 772\)](#).
- If you have S3 Lifecycle configured for your bucket, we recommend disabling your Lifecycle rules while the Batch Replication job is active. This will ensure parity between the source and destination buckets. Otherwise these buckets could diverge and the destination bucket will not be an exact replica of the source bucket. Consider the following:
 - Your source bucket has multiple versions on an object and a delete marker.
 - Your source and destination buckets have a Lifecycle policy to remove expired delete markers.

Batch Replication may replicate the delete marker to the destination bucket before replicating the object versions. This could result in the delete marker being marked as expired and being removed from the destination bucket before the objects are copied.

- The AWS Identity and Access Management (IAM) role that you specify to run the Batch Operations job must have permissions to perform the underlying Batch Replication operation. For more information about creating IAM roles, see [Configuring IAM policies for Batch Replication \(p. 800\)](#).
- Batch Replication requires a manifest which can be generated by Amazon S3. The generated manifest must be stored in the same AWS Region as the source bucket. If you choose to not generate the

manifest you may supply a Amazon S3 Inventory report or CSV file that contains the objects you wish to replicate.

- Batch Replication does not support re-replicating objects that were deleted with the version ID of the object from the destination bucket. To re-replicate these objects you can copy the source objects in place with a Batch Copy job. Copying those objects in place will create new versions of the object in the source bucket and initiate replication automatically to the destination.

For more information on Batch Copy, see, [Examples that use Batch Operations to copy objects \(p. 896\)](#).

Specifying a manifest for a Batch Replication job

A manifest is an Amazon S3 object that contains object keys that you want Amazon S3 to act upon. If you wish to create a Batch Replication job you must supply either a user-generated manifest or have Amazon S3 generate a manifest based on your replication configuration.

If you supply a user-generated manifest it must be in the form of a Amazon S3 Inventory report or CSV file. If the objects in your manifest are in a versioned bucket, you must specify the version IDs for the objects. Only the object with the version ID specified in the manifest will be replicated. To learn more about specifying a manifest, see [Specifying a manifest \(p. 890\)](#).

If you choose to have Amazon S3 generate a manifest file on your behalf the objects listed will use the same source bucket, prefix, and tags as your replication configuration. With a generated manifest Amazon S3 will replicate all eligible versions of your objects.

Note

If you choose to have the manifest generated it must be stored in the same AWS Region as the source bucket.

Filters for a Batch Replication job

When creating your Batch Replication job you can optionally specify additional filters, such as object creation date and replication status to reduce the scope of the job.

You can filter objects to replicate based on the `ObjectReplicationStatuses` value, by providing one or more of the following values:

- "NONE" – Indicates that Amazon S3 has never attempted to replicate the object before.
- "FAILED" – Indicates that Amazon S3 has attempted, but failed to replicate the object before.
- "COMPLETED" – Indicates that Amazon S3 has successfully replicated the object before.
- "REPLICA" – Indicates that this is a replica object that Amazon S3 replicated from another source.

For more information about replication statuses, see [Getting replication status information \(p. 820\)](#).

If you do not filter based on replication status Batch Operations will attempt to replicate everything eligible. Depending on your goal, you might set `ObjectReplicationStatuses` to one of the following values:

- If you want to replicate only existing objects that have never been replicated, only include "NONE".
- If you want to retry replicating only objects that previously failed to replicate, only include "FAILED".
- If you want to both replicate existing objects and retry replicating objects that previously failed to replicate, include both "NONE" and "FAILED".
- If you want to back-fill a destination bucket with objects that have been replicated to another destination, include "COMPLETED".

- If you want replicate objects previously replicated, include "REPLICA".

Configuring IAM policies for Batch Replication

Because S3 Batch Replication is a type of Batch Operations job, you must create a Batch Operations AWS Identity and Access Management (IAM) role to grant Amazon S3 permissions to perform actions on your behalf. You also must attach a Batch Replication IAM policy to the Batch Operations IAM role. The following example creates an IAM role that gives Batch Operations permission to initiate a Batch Replication job.

Create IAM role and policy

1. Sign in to the AWS Management Console and open the IAM console at <https://console.aws.amazon.com/iam/>.
2. Under **Access management**, choose **Roles**.
3. Choose **Create Role**.
4. Choose **AWS service** as the type of trusted entity, **Amazon S3** as the service, and **S3 Batch Operations** as the use case.
5. Choose **Next: Permissions**.
6. Choose **Create Policy**.
7. Choose **JSON** and insert one of the following policies based on your manifest.

Note

Different permission are needed if you are generating a manifest or supplying one. For more information see, [Specifying a manifest for a Batch Replication job \(p. 799\)](#).

Policy if using and storing a S3 generated manifest

```
{  
    "Version": "2012-10-17",  
    "Statement": [  
        {  
            "Action": [  
                "s3:InitiateReplication"  
            ],  
            "Effect": "Allow",  
            "Resource": [  
                "arn:aws:s3:::*** replication source bucket ***/*"  
            ]  
        },  
        {  
            "Action": [  
                "s3:GetReplicationConfiguration",  
                "s3:PutInventoryConfiguration"  
            ],  
            "Effect": "Allow",  
            "Resource": [  
                "arn:aws:s3:::*** replication source bucket ***"  
            ]  
        },  
        {  
            "Action": [  
                "s3:GetObject",  
                "s3:GetObjectVersion"  
            ],  
            "Effect": "Allow",  
            "Resource": [  
                "arn:aws:s3:::*** replication source bucket ***/*"  
            ]  
        }  
    ]  
}
```

```
        "arn:aws:s3:::*** manifest bucket ***/*"
    ],
},
{
    "Effect": "Allow",
    "Action": [
        "s3:PutObject"
    ],
    "Resource": [
        "arn:aws:s3:::*** completion report bucket ****/*",
        "arn:aws:s3:::*** manifest bucket ****/*"
    ]
}
]
```

Policy if using a user supplied manifest

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Action": [
                "s3:InitiateReplication"
            ],
            "Effect": "Allow",
            "Resource": [
                "arn:aws:s3:::*** replication source bucket ***/*"
            ]
        },
        {
            "Action": [
                "s3:GetObject",
                "s3:GetObjectVersion"
            ],
            "Effect": "Allow",
            "Resource": [
                "arn:aws:s3:::*** manifest bucket ***/*"
            ]
        },
        {
            "Effect": "Allow",
            "Action": [
                "s3:PutObject"
            ],
            "Resource": [
                "arn:aws:s3:::*** completion report bucket ****/*"
            ]
        }
    ]
}
```

8. Choose **Next: Tags**.
9. Choose **Next: Review**.
10. Choose a name for the policy and choose **Create policy**.
11. Attach this policy to your role and choose **Next: Tags**.
12. Choose **Next: Review**.
13. Choose a name for the role and choose **Create role**.

Verify trust policy

1. Sign in to the AWS Management Console and open the IAM console at <https://console.aws.amazon.com/iam/>.
2. Under **Access management**, choose **Roles**, and select your newly created role.
3. Under **Trust relationships** tab, choose **Edit trust relationship**.
4. Verify this role is using the following trust policy:

```
{  
    "Version": "2012-10-17",  
    "Statement": [  
        {  
            "Effect": "Allow",  
            "Principal": {  
                "Service": "batchoperations.s3.amazonaws.com"  
            },  
            "Action": "sts:AssumeRole"  
        }  
    ]  
}
```

Create a Batch Replication job for a first replication rule or new destination

When you create the first rule in a new replication configuration or add a new destination to an existing configuration through the AWS Management Console, you can optionally create a Batch Replication job.

To use Batch Replication for an existing configuration without adding a new destination see, [Create a Batch Replication job for existing replication rules \(p. 803\)](#).

Using Batch Replication for a new replication rule or destination through the AWS Management Console

1. Sign in to the AWS Management Console and open the Amazon S3 console at <https://console.aws.amazon.com/s3/>.
2. In the **Buckets** list, choose the name of the bucket that contains the objects that you want to replicate.
3. To create a new replication rule or edit an existing rule, choose **Management**, and scroll down to **Replication rules**:
 - To create a new replication rule, choose **Create replication rule**.

Note

For examples on how to set up a basic replication rule see, [Walkthroughs: Examples for configuring replication \(p. 772\)](#).

4. Create your new replication rule or edit the destination for your existing replication rule, and choose **Save**.

After you create the first rule in a new replication configuration or edit an existing configuration to add a new destination, a **Replicate existing objects?** dialog appears, giving you the option to create a Batch Replication job.

5. To create a Batch Replication job, choose **Yes, replicate existing objects**.

To run your Batch Replication job at a later time, choose **No, do not replicate existing objects**.

6. Create your S3 Batch Replication job. The S3 Batch Replication job has several settings:

Job run option

If you want the S3 Batch Replication job to run immediately, you can choose **Job runs automatically when ready**. If you want to run the job at a later time, choose **Job waits to be run when ready**.

If you choose **Job runs automatically when ready**, you will not be able to create and save a Batch Operations manifest. To save the Batch Operations manifest, choose **Job waits to be run when ready**.

Batch Operations manifest

The manifest is a list of all of the objects that you want to run the specified action on. You can choose to save the Batch Operations manifest. Similar to S3 Inventory files, the manifest will be saved as a CSV file and stored in a bucket. To learn more about Batch Operations manifests, see [Specifying a manifest \(p. 890\)](#).

Completion report

S3 Batch Operations execute one task for each object specified in the manifest. Completion reports provide an easy way to view the results of your tasks in a consolidated format with no additional setup required. You can request a completion report for all tasks or only for failed tasks. To learn more about completion reports, see [Completion reports \(p. 927\)](#).

Permissions

One of the most common causes of replication failures is insufficient permissions in the provided AWS Identity and Access Management (IAM) role. For information about creating this role see, [Configuring IAM policies for Batch Replication \(p. 800\)](#).

7. Choose **Create Batch Operations job**.

Create a Batch Replication job for existing replication rules

You can configure S3 Batch Replication for an existing replication configuration by using the AWS SDKs, AWS Command Line Interface (AWS CLI), or the Amazon S3 console. For an overview of Batch Replication see, [Replicating existing objects with S3 Batch Replication \(p. 798\)](#).

As a prerequisite, you must create a Batch Operations AWS Identity and Access Management (IAM) role to grant Amazon S3 permissions to perform actions on your behalf, see [Configuring IAM policies for Batch Replication \(p. 800\)](#).

When the Batch Replication job finishes, you receive a completion report. For more information about how to use the report to examine the job, see [Tracking job status and completion reports \(p. 922\)](#).

Using the S3 console

1. Sign in to the AWS Management Console and open the Amazon S3 console at <https://console.aws.amazon.com/s3/>.
2. Choose **Batch Operations** on the navigation pane of the Amazon S3 console.
3. Choose **Create job**.
4. Choose the **Region** where you want to create your job.
5. Select the **Manifest format**. This example will show how to create a manifest based on an existing S3 replication configuration.

Note

The manifest is a list of all of the objects that you want to run the specified action on. To learn more about Batch Operations manifests, see [Specifying a manifest \(p. 890\)](#).

If you have a manifest prepared, choose **S3 inventory report (manifest.json) or CSV**. If the objects in your manifest are in a versioned bucket, you should specify the version IDs for the objects. For more information about creating a manifest see, [Specifying a manifest \(p. 890\)](#).

6. To create a manifest based on your replication configuration, choose **Create manifest using S3 Replication configuration**. Then chose the source bucket of your replication configuration.
7. (Optional) You may include additional filters such as object creation date and replication status. For examples on how to filter by replication status see, [Specifying a manifest for a Batch Replication job \(p. 799\)](#).
8. To save a manifest, select **Save Batch Operations manifest**.
 - a. If you choose to generate and save a manifest, you must choose either **Bucket in this account** or **Bucket in another AWS account**. Specify the bucket name in the text box.

Note

The generated manifest must be stored in the same AWS Region as the source bucket.

- b. Choose the **Encryption key type**.

9. (Optional) Provide a **Description**.
10. Adjust the **Priority** of the job if needed. Higher numbers indicate higher priority. Amazon S3 attempts to run higher priority jobs before lower priority jobs. For more information about job priority, see [Assigning job priority \(p. 921\)](#).
11. (Optional) Generate a completion report. To generate select **Generate completion report**.

If you choose to generate a completion report, you must choose either to report **Failed tasks only** or **All tasks**, and provide a destination bucket for the report.

12. Select a valid IAM role.

Note

For more information about creating a IAM role, see [Configuring IAM policies for Batch Replication \(p. 800\)](#).

13. (Optional) Add job tags to the Batch Replication job.
14. Choose **Next**.
15. Review your job configuration and select **Create job**.

Using the AWS CLI with an S3 manifest

The following example creates an S3 Batch Replication job using a S3 generated manifest for the AWS account **111122223333**. This example will try to replicate existing objects and objects that previously failed to replicate. For information about filtering by replication status see, [Specifying a manifest for a Batch Replication job \(p. 799\)](#).

```
aws s3control create-job --account-id 111122223333 --operation
  '{"S3ReplicateObject":{} }' --report '{ "Bucket": "arn:aws:s3:::***"
  completion_report bucket ***", "Prefix": "batch-replication-report",
  "Format": "Report_CSV_20180820", "Enabled": true, "ReportScope": "AllTasks"}' --manifest-
  generator '{ "S3JobManifestGenerator": { "ExpectedBucketOwner": "111122223333",
  "SourceBucket": "arn:aws:s3:::*** replication source bucket ***", "EnableManifestOutput": 
  false, "Filter": { "EligibleForReplication": true, "ObjectReplicationStatuses": [
  "NONE", "FAILED"]}}}' --priority 1 --role-arn arn:aws:iam::111122223333:role/batch-
  Replication-IAM-policy --no-confirmation-required --region source-bucket-region
```

Note

The job must be initiated from the same AWS Region replication source bucket. The IAM role **role/batch-Replication-IAM-policy** was previously created. See [Configuring IAM policies for Batch Replication \(p. 800\)](#).

After you have successfully initiated a Batch Replication job, you receive the job ID as the response. You can monitor this job using the following command.

```
aws s3control describe-job --account-id 111122223333 --job-id job-id --region source-bucket-region
```

Using the AWS CLI with a user-provided manifest

The following example creates an S3 Batch Replication job using a user-defined manifest for AWS account **111122223333**. If the objects in your manifest are in a versioned bucket, you must specify the version IDs for the objects. Only the object with the version ID specified in the manifest will be replicated. For more information about creating a manifest see, [Specifying a manifest \(p. 890\)](#).

```
aws s3control create-job --account-id 111122223333 --operation
'{"S3ReplicateObject":{}}' --report '{"Bucket":"arn:aws:s3:::***  

completion report bucket ***", "Prefix": "batch-replication-report",  

"Format":"Report_CSV_20180820","Enabled":true,"ReportScope":"AllTasks"}'  

--manifest '{"Spec":{"Format":"S3BatchOperations_CSV_20180820","Fields":  

["Bucket","Key","VersionId"]},"Location":{"ObjectArn":"arn:aws:s3:::*** completion  

report bucket ***/manifest.csv", "ETag": "Manifest Etag"}}}' --priority 1 --role-arn  

arn:aws:iam::111122223333:role/batch-Replication-IAM-policy --no-confirmation-required --  

region source-bucket-region
```

Note

The job must be initiated from the same AWS Region replication source bucket. The IAM role **role/batch-Replication-IAM-policy** was previously created. See [Configuring IAM policies for Batch Replication \(p. 800\)](#).

After you have successfully initiated a Batch Replication job, you receive the job ID as the response. You can monitor this job using the following command.

```
aws s3control describe-job --account-id 111122223333 --job-id job-id --region source-bucket-region
```

Additional replication configurations

This section describes additional replication configuration options that are available in Amazon S3. For information about core replication configuration, see [Setting up replication \(p. 758\)](#).

Topics

- [Monitoring progress with replication metrics and Amazon S3 event notifications \(p. 805\)](#)
- [Meeting compliance requirements using S3 Replication Time Control \(S3 RTC\) \(p. 807\)](#)
- [Replicating delete markers between buckets \(p. 810\)](#)
- [Replicating metadata changes with Amazon S3 replica modification sync \(p. 811\)](#)
- [Changing the replica owner \(p. 812\)](#)
- [Replicating objects created with server-side encryption \(SSE\) using KMS keys \(p. 814\)](#)

Monitoring progress with replication metrics and Amazon S3 event notifications

S3 replication metrics provide detailed metrics for the replication rules in your replication configuration. With replication metrics, you can monitor minute-by-minute progress of replication by tracking bytes pending, operations pending, and replication latency. Additionally, you can set up Amazon S3 Event Notifications to receive replication failure events to assist in troubleshooting any configuration issues.

When enabled, S3 replication metrics publish the following metrics to Amazon CloudWatch:

Bytes Pending Replication—The total number of bytes of objects pending replication for a given replication rule.

Replication Latency—The maximum number of seconds by which the replication destination buckets are behind the source bucket for a given replication rule.

Operations Pending Replication—The number of operations pending replication for a given replication rule. Operations include objects, delete markers, tags, ACLs, and Object Lock operations.

Note

S3 replication metrics are billed at the same rate as Amazon CloudWatch custom metrics. For information, see [Amazon CloudWatch pricing](#).

S3 replication metrics are turned on automatically when you enable S3 Replication Time Control (S3 RTC). S3 RTC includes other features such as a service level agreement (SLA) and notifications for missed thresholds. For more information, see [Meeting compliance requirements using S3 Replication Time Control \(S3 RTC\) \(p. 807\)](#).

Topics

- [Enabling S3 replication metrics \(p. 806\)](#)
- [Receiving replication failure events with Amazon S3 event notifications \(p. 807\)](#)
- [Viewing replication metrics using the Amazon S3 console \(p. 807\)](#)

Enabling S3 replication metrics

You can start using S3 replication metrics with a new or existing replication rule. You can choose to apply your replication rule to an entire S3 bucket, or to Amazon S3 objects with a specific prefix or tag.

This topic provides instructions for enabling S3 replication metrics in your replication configuration when buckets are owned by the same or different AWS accounts.

To enable replication metrics using the AWS Command Line Interface (AWS CLI), you must add a replication configuration to the source bucket with Metrics enabled. In this example configuration, objects under the prefix `Tax` are replicated to the destination bucket `DOC-EXAMPLE-BUCKET`, and metrics are generated for those objects.

```
{  
    "Rules": [  
        {  
            "Status": "Enabled",  
            "Filter": {  
                "Prefix": "Tax"  
            },  
            "Destination": {  
                "Bucket": "arn:aws:s3:::DOC-EXAMPLE-BUCKET",  
                "Metrics": {  
                    "Status": "Enabled"  
                }  
            },  
            "Priority": 1  
        }  
    ],  
    "Role": "IAM-Role-ARN"  
}
```

For full instructions on creating replication rules, see [Configuring replication for source and destination buckets owned by the same account \(p. 773\)](#).

For more information about viewing replication metrics in the S3 console, see [Viewing replication metrics using the Amazon S3 console \(p. 807\)](#).

Receiving replication failure events with Amazon S3 event notifications

Amazon S3 event notifications can notify you in the rare instance when objects do not replicate to their destination Region. Amazon S3 events are available through Amazon Simple Queue Service (Amazon SQS), Amazon Simple Notification Service (Amazon SNS), or AWS Lambda. For more information, see [Configuring Amazon S3 event notifications](#).

Viewing replication metrics using the Amazon S3 console

There are three types of Amazon CloudWatch metrics for Amazon S3: storage metrics, request metrics, and replication metrics. *Replication* metrics are turned on automatically when you enable replication with S3 Replication Time Control (S3 RTC) using the AWS Management Console or the Amazon S3 API. Replication metrics are available 15 minutes after you enable a replication rule with S3 Replication Time Control (S3 RTC) (S3 RTC).

Replication metrics track the rule IDs of the replication configuration. A replication rule ID can be specific to a prefix, a tag, or a combination of both. For more information about S3 Replication Time Control (S3 RTC), see [Meeting compliance requirements using S3 Replication Time Control \(S3 RTC\) \(p. 807\)](#).

For more information about CloudWatch metrics for Amazon S3, see [Monitoring metrics with Amazon CloudWatch \(p. 1002\)](#).

Prerequisites

Enable a replication rule that has S3 RTC.

To view replication metrics

1. Sign in to the AWS Management Console and open the Amazon S3 console at <https://console.aws.amazon.com/s3/>.
2. In the **Buckets** list, choose the name of the bucket that contains the objects you want replication metrics for.
3. Choose the **Metrics** tab.
4. Under **Replication metrics**, choose **Replication rules**.
5. Choose **Display charts**.

Amazon S3 displays **Replication Latency (in seconds)**, **Operations pending replication** in charts.

6. To view all replication metrics, including **Bytes pending replication**, **Replication Latency (in seconds)**, and **Operations pending replication** together on a separate page, choose **View 1 more chart**.

You can then view the replication metrics **Replication Latency (in seconds)**, **Operations pending replication**, and **Bytes pending replication** for the rules that you selected. Amazon CloudWatch begins reporting replication metrics 15 minutes after you enable S3 RTC on the respective replication rule. You can view replication metrics on the Amazon S3 or CloudWatch console. For information, see [Replication metrics with S3 RTC \(p. 808\)](#).

Meeting compliance requirements using S3 Replication Time Control (S3 RTC)

S3 Replication Time Control (S3 RTC) helps you meet compliance or business requirements for data replication and provides visibility into Amazon S3 replication times. S3 RTC replicates most objects that you upload to Amazon S3 in seconds, and 99.99 percent of those objects within 15 minutes.

S3 RTC by default includes S3 replication metrics and S3 event notifications, with which you can monitor the total number of S3 API operations that are pending replication, the total size of objects pending replication, and the maximum replication time. Replication metrics can be enabled independently of S3 RTC, see [Monitoring progress with replication metrics](#). Additionally, S3 RTC provides `OperationMissedThreshold` and `OperationReplicatedAfterThreshold` events that notify the bucket owner if object replication exceeds or replicates after the 15-minute threshold.

With S3 RTC, Amazon S3 events can notify you in the rare instance when objects do not replicate within 15 minutes and when those objects replicate successfully to their destination Region. Amazon S3 events are available through Amazon SQS, Amazon SNS, or AWS Lambda. For more information, see the section called ["Amazon S3 Event Notifications" \(p. 1017\)](#).

Topics

- [Enabling S3 Replication Time Control \(p. 808\)](#)
- [Replication metrics with S3 RTC \(p. 808\)](#)
- [Using Amazon S3 event notifications to track replication objects \(p. 808\)](#)
- [Best practices and guidelines for S3 RTC \(p. 809\)](#)

Enabling S3 Replication Time Control

You can start using S3 Replication Time Control (S3 RTC) with a new or existing replication rule. You can choose to apply your replication rule to an entire S3 bucket, or to Amazon S3 objects with a specific prefix or tag. When you enable S3 RTC, replication metrics are also enabled on your replication rule.

If you are using the latest version of the replication configuration (that is, you specify the `Filter` element in a replication configuration rule), Amazon S3 does not replicate the delete marker by default. However you can add delete marker replication to non-tag-based rules.

Note

Replication metrics are billed at the same rate as Amazon CloudWatch custom metrics. For information, see [Amazon CloudWatch pricing](#).

For more information about creating a rule with S3 RTC, see [Replicating objects with S3 Replication Time Control \(S3 RTC\) \(p. 795\)](#).

Replication metrics with S3 RTC

Replication rules with S3 Replication Time Control (S3 RTC) enabled publishes replication metrics. With replication metrics, you can monitor the total number of S3 API operations that are pending replication, the total size of objects pending replication, and the maximum replication time to the destination Region. You can then monitor each dataset that you replicate separately.

Replication metrics are available within 15 minutes of enabling S3 RTC. Replication metrics are available through the [Amazon S3 console](#), the [Amazon S3 API](#), the AWS SDKs, the [AWS Command Line Interface \(AWS CLI\)](#), and [Amazon CloudWatch](#). For more information, see [Monitoring metrics with Amazon CloudWatch \(p. 1002\)](#).

For more information about finding replication metrics via the Amazon S3 console, see [Viewing replication metrics using the Amazon S3 console \(p. 807\)](#).

Using Amazon S3 event notifications to track replication objects

You can track replication time for objects that did not replicate within 15 minutes by monitoring specific event notifications that S3 Replication Time Control (S3 RTC) publishes. These events are published when an object that was eligible for replication using S3 RTC didn't replicate within 15 minutes, and when that object replicates to the destination Region.

Replication events are available within 15 minutes of enabling S3 RTC. Amazon S3 events are available through Amazon SQS, Amazon SNS, or AWS Lambda. For more information, see [Amazon S3 Event Notifications \(p. 1017\)](#).

Best practices and guidelines for S3 RTC

When replicating data in Amazon S3 using S3 Replication Time Control (S3 RTC), follow these best practice guidelines to optimize replication performance for your workloads.

Topics

- [Amazon S3 Replication and request rate performance guidelines \(p. 809\)](#)
- [Estimating your replication request rates \(p. 809\)](#)
- [Exceeding S3 RTC data transfer rate limits \(p. 809\)](#)
- [AWS KMS encrypted object replication request rates \(p. 810\)](#)

Amazon S3 Replication and request rate performance guidelines

When uploading and retrieving storage from Amazon S3, your applications can achieve thousands of transactions per second in request performance. For example, an application can achieve at least 3,500 PUT/COPY/POST/DELETE or 5,500 GET/HEAD requests per second per prefix in an S3 bucket, including the requests that S3 replication makes on your behalf. There are no limits to the number of prefixes in a bucket. You can increase your read or write performance by parallelizing reads. For example, if you create 10 prefixes in an S3 bucket to parallelize reads, you could scale your read performance to 55,000 read requests per second.

Amazon S3 automatically scales in response to sustained request rates above these guidelines, or sustained request rates concurrent with LIST requests. While Amazon S3 is internally optimizing for the new request rate, you might receive HTTP 503 request responses temporarily until the optimization is complete. This might occur with increases in request per second rates, or when you first enable S3 RTC. During these periods, your replication latency might increase. The S3 RTC service level agreement (SLA) doesn't apply to time periods when Amazon S3 performance guidelines on requests per second are exceeded.

The S3 RTC SLA also doesn't apply during time periods where your replication data transfer rate exceeds the default 1 Gbps limit. If you expect your replication transfer rate to exceed 1 Gbps, you can contact [AWS Support Center](#) or use [Service Quotas](#) to request an increase in your limit.

Estimating your replication request rates

Your total request rate including the requests that Amazon S3 replication makes on your behalf should be within the Amazon S3 request rate guidelines for both the replication source and destination buckets. For each object replicated, Amazon S3 replication makes up to five GET/HEAD requests and one PUT request to the source bucket, and one PUT request to each destination bucket.

For example, if you expect to replicate 100 objects per second, Amazon S3 replication might perform an additional 100 PUT requests on your behalf for a total of 200 PUTs per second to the source S3 bucket. Amazon S3 replication also might perform up to 500 GET/HEAD (5 GET/HEAD requests for each object replicated.)

Note

You incur costs for only one PUT request per object replicated. For more information, see the pricing information in the [Amazon S3 FAQ on replication](#).

Exceeding S3 RTC data transfer rate limits

If you expect your S3 Replication Time Control data transfer rate to exceed the default 1 Gbps limit, contact [AWS Support Center](#) or use [Service Quotas](#) to request an increase in your limit.

AWS KMS encrypted object replication request rates

When you replicate objects encrypted with server-side encryption (SSE-KMS) using Amazon S3 replication, AWS Key Management Service (AWS KMS) requests per second limits apply. AWS KMS might reject an otherwise valid request because your request rate exceeds the limit for the number of requests per second. When a request is throttled, AWS KMS returns a `ThrottlingException` error. The AWS KMS request rate limit applies to requests you make directly and to requests made by Amazon S3 replication on your behalf.

For example, if you expect to replicate 1,000 objects per second, you can subtract 2,000 requests from your AWS KMS request rate limit. The resulting request rate per second is available for your AWS KMS workloads excluding replication. You can use [AWS KMS request metrics in Amazon CloudWatch](#) to monitor the total AWS KMS request rate on your AWS account.

Replicating delete markers between buckets

By default, when Amazon S3 Replication is enabled and an object is deleted in the source bucket, Amazon S3 adds a delete marker in the source bucket only. This action protects data from malicious deletions.

If you have *delete marker replication* enabled, these markers are copied to the destination buckets, and Amazon S3 behaves as if the object was deleted in both source and destination buckets. For more information about how delete markers work, see [Working with delete markers \(p. 662\)](#).

Note

Delete marker replication is not supported for tag-based replication rules. Delete marker replication also does not adhere to the 15-minute SLA granted when using S3 Replication Time Control.

If you are not using the latest replication configuration version, delete operations will affect replication differently. For more information, see [How delete operations affect replication \(p. 757\)](#).

Enabling delete marker replication

You can start using delete marker replication with a new or existing replication rule. You can apply it to an entire S3 bucket or to Amazon S3 objects that have a specific prefix.

To enable delete marker replication using the Amazon S3 console, see [Using the S3 console \(p. 773\)](#). This topic provides instructions for enabling delete marker replication in your replication configuration when buckets are owned by the same or different AWS accounts.

To enable delete marker replication using the AWS Command Line Interface (AWS CLI), you must add a replication configuration to the source bucket with `DeleteMarkerReplication` enabled.

In the following example configuration, delete markers are replicated to the destination bucket `DOC-EXAMPLE-BUCKET` for objects under the prefix `Tax`.

```
{  
    "Rules": [  
        {  
            "Status": "Enabled",  
            "Filter": {  
                "Prefix": "Tax"  
            },  
            "DeleteMarkerReplication": {  
                "Status": "Enabled"  
            },  
            "Destination": {  
                "Bucket": "DOC-EXAMPLE-BUCKET",  
                "Format": "JSON",  
                "Encryption": "AES256",  
                "Role": "arn:aws:iam::123456789012:role/S3-Replication-Role",  
                "Status": "Enabled"  
            }  
        }  
    ]  
}
```

```

        "Bucket": "arn:aws:s3:::DOC-EXAMPLE-BUCKET"
    },
    "Priority": 1
}
],
"Role": "IAM-Role-ARN"
}

```

For full instructions on creating replication rules through the AWS CLI, see [Configuring replication for source and destination buckets owned by the same account \(p. 773\)](#) in the Replication walkthroughs section.

Replicating metadata changes with Amazon S3 replica modification sync

Amazon S3 replica modification sync can help you keep object metadata such as tags, ACLs, and Object Lock settings replicated between replicas and source objects. By default, Amazon S3 replicates metadata from the source objects to the replicas only. When replica modification sync is enabled, Amazon S3 replicates metadata changes made to the replica copies back to the source object, making the replication bidirectional.

Enabling replica modification sync

You can use Amazon S3 replica modification sync with new or existing replication rules. You can apply it to an entire S3 bucket or to Amazon S3 objects that have a specific prefix.

To enable replica modification sync using the Amazon S3 console, see [Walkthroughs: Examples for configuring replication \(p. 772\)](#). This topic provides instructions for enabling replica modification sync in your replication configuration when buckets are owned by the same or different AWS accounts.

To enable replica modification sync using the AWS Command Line Interface (AWS CLI), you must add a replication configuration to the bucket containing the replicas with `ReplicaModifications` enabled. To set up two-way replication, create a replication rule from the source bucket (*DOC-EXAMPLE-BUCKET1*) to the bucket containing the replicas (*DOC-EXAMPLE-BUCKET2*). Then, create a second replication rule from the bucket containing the replicas (*DOC-EXAMPLE-BUCKET2*) to the source bucket (*DOC-EXAMPLE-BUCKET1*). Buckets can be in the same, or in different, AWS Regions.

Note

You must enable replica modification sync on both buckets to replicate replica metadata changes like object access control lists (ACLs), object tags, or Object Lock settings on the replicated objects. Like all replication rules, these rules can either be applied to the entire Amazon S3 bucket or a subset of Amazon S3 objects filtered by prefix or object tags.

In the following example configuration, Amazon S3 replicates metadata changes under the prefix `Tax` to the bucket *DOC-EXAMPLE-BUCKET*, which would contain the source objects.

```
{
    "Rules": [
        {
            "Status": "Enabled",
            "Filter": {
                "Prefix": "Tax"
            },
            "SourceSelectionCriteria": {
                "ReplicaModifications": {
                    "Status": "Enabled"
                }
            },
        }
    ]
}
```

```
        "Destination": {
            "Bucket": "arn:aws:s3:::DOC-EXAMPLE-BUCKET"
        },
        "Priority": 1
    ],
    "Role": "IAM-Role-ARN"
}
```

For full instructions on creating replication rules using the AWS CLI, see [Configuring replication for source and destination buckets owned by the same account \(p. 773\)](#).

Changing the replica owner

In replication, the owner of the source object also owns the replica by default. When source and destination buckets are owned by different AWS accounts and you want to change replica ownership to the AWS account that owns the destination buckets, you can add optional configuration settings to change replica ownership to the AWS account that owns the destination buckets. You might do this, for example, to restrict access to object replicas. This is referred to as the *owner override* option of the replication configuration. For more information about the owner override option, see [Adding the owner override option to the replication configuration \(p. 812\)](#). For information about setting the replication configuration, see [Replicating objects \(p. 753\)](#).

To configure the owner override, you do the following:

- Add the owner override option to the replication configuration to tell Amazon S3 to change replica ownership.
- Grant Amazon S3 permissions to change replica ownership.
- Add permission in the destination buckets policy to allow changing replica ownership. This allows the owner of the destination buckets to accept the ownership of object replicas.

For more information, see [Adding the owner override option to the replication configuration \(p. 812\)](#). For a working example with step-by-step instructions, see [Changing the replica owner when source and destination buckets are owned by different accounts \(p. 786\)](#).

Bucket owner enforced setting for Object Ownership

When you use Amazon S3 replication and the source and destination buckets are owned by different AWS accounts, the bucket owner of the destination bucket can disable ACLs (with the bucket owner enforced setting for Object Ownership) to change replica ownership to the AWS account that owns the destination bucket. This setting mimics the existing owner override behavior without the need of `s3:ObjectOwnerOverrideToBucketOwner` permission. This means that all objects that are replicated to the destination bucket with the bucket owner enforced setting are owned by the destination bucket owner. For more information about Object Ownership, see [Controlling ownership of objects and disabling ACLs for your bucket \(p. 601\)](#).

Adding the owner override option to the replication configuration

Warning

Add the owner override option only when the source and destination buckets are owned by different AWS accounts. Amazon S3 doesn't check if the buckets are owned by same or different accounts. If you add the owner override when both buckets are owned by same AWS account, Amazon S3 applies the owner override. It grants full permissions to the owner of the destination bucket and doesn't replicate subsequent updates to the source object access control list (ACL). The replica owner can directly change the ACL associated with a replica with a `PUT ACL` request, but not through replication.

To specify the owner override option, add the following to each `Destination` element:

- The `AccessControlTranslation` element, which tells Amazon S3 to change replica ownership
- The `Account` element, which specifies the AWS account of the destination bucket owner

```
<ReplicationConfiguration xmlns="http://s3.amazonaws.com/doc/2006-03-01/">
  ...
  <Destination>
    ...
    <AccessControlTranslation>
      <Owner>Destination</Owner>
    </AccessControlTranslation>
    <Account>destination-bucket-owner-account-id</Account>
  </Destination>
</Rule>
</ReplicationConfiguration>
```

The following example replication configuration tells Amazon S3 to replicate objects that have the `Tax` key prefix to the destination bucket and change ownership of the replicas.

```
<?xml version="1.0" encoding="UTF-8"?>
<ReplicationConfiguration xmlns="http://s3.amazonaws.com/doc/2006-03-01/">
  <Role>arn:aws:iam::account-id:role/role-name</Role>
  <Rule>
    <ID>Rule-1</ID>
    <Priority>1</Priority>
    <Status>Enabled</Status>
    <DeleteMarkerReplication>
      <Status>Disabled</Status>
    </DeleteMarkerReplication>
    <Filter>
      <Prefix>Tax</Prefix>
    </Filter>
    <Destination>
      <Bucket>arn:aws:s3:::destination-bucket</Bucket>
      <Account>destination-bucket-owner-account-id</Account>
      <AccessControlTranslation>
        <Owner>Destination</Owner>
      </AccessControlTranslation>
    </Destination>
  </Rule>
</ReplicationConfiguration>
```

Granting Amazon S3 permission to change replica ownership

Grant Amazon S3 permissions to change replica ownership by adding permission for the `s3:ObjectOwnerOverrideToBucketOwner` action in the permissions policy associated with the IAM role. This is the IAM role that you specified in the replication configuration that allows Amazon S3 to assume and replicate objects on your behalf.

```
...
{
  "Effect": "Allow",
  "Action": [
    "s3:ObjectOwnerOverrideToBucketOwner"
  ],
  "Resource": "arn:aws:s3:::destination-bucket/*"
}
```

Adding permission in the destination bucket policy to allow changing replica ownership

The owner of the destination bucket must grant the owner of the source bucket permission to change replica ownership. The owner of the destination bucket grants the owner of the source bucket permission for the `s3:ObjectOwnerOverrideToBucketOwner` action. This allows the destination bucket owner to accept ownership of the object replicas. The following example bucket policy statement shows how to do this.

```
...
{
    "Sid": "1",
    "Effect": "Allow",
    "Principal": {"AWS": "source-bucket-account-id"},
    "Action": [ "s3:ObjectOwnerOverrideToBucketOwner" ],
    "Resource": "arn:aws:s3:::destination-bucket/*"
}
...
```

Additional considerations

When you configure the ownership override option, the following considerations apply:

- By default, the owner of the source object also owns the replica. Amazon S3 replicates the object version and the ACL associated with it.

If you add the owner override, Amazon S3 replicates only the object version, not the ACL. In addition, Amazon S3 doesn't replicate subsequent changes to the source object ACL. Amazon S3 sets the ACL on the replica that grants full control to the destination bucket owner.

- When you update a replication configuration to enable, or disable, the owner override, the following occurs.

- If you add the owner override option to the replication configuration:

When Amazon S3 replicates an object version, it discards the ACL that is associated with the source object. Instead, it sets the ACL on the replica, giving full control to the owner of the destination bucket. It doesn't replicate subsequent changes to the source object ACL. However, this ACL change doesn't apply to object versions that were replicated before you set the owner override option. ACL updates on source objects that were replicated before the owner override was set continue to be replicated (because the object and its replicas continue to have the same owner).

- If you remove the owner override option from the replication configuration:

Amazon S3 replicates new objects that appear in the source bucket and the associated ACLs to the destination buckets. For objects that were replicated before you removed the owner override, Amazon S3 doesn't replicate the ACLs because the object ownership change that Amazon S3 made remains in effect. That is, ACLs put on the object version that were replicated when the owner override was set continue to be not replicated.

Replicating objects created with server-side encryption (SSE) using KMS keys

By default, Amazon S3 doesn't replicate objects that are stored at rest using server-side encryption with customer managed keys stored in AWS KMS. This section explains additional configuration that you add to direct Amazon S3 to replicate these objects.

Note

You can use multi-Region AWS KMS keys in Amazon S3. However, Amazon S3 currently treats multi-Region keys as though they were single-Region keys, and does not use the multi-Region features of the key. For more information, see [Using multi-Region keys in AWS Key Management Service Developer Guide](#).

For an example with step-by-step instructions, see [Replicating encrypted objects \(p. 790\)](#). For information about creating a replication configuration, see [Replicating objects \(p. 753\)](#).

Important

Replication of encrypted data is a server-side process that occurs entirely within Amazon S3. Objects created with server-side encryption using customer-provided (SSE-C) encryption keys are not replicated.

Topics

- [Specifying additional information in the replication configuration \(p. 815\)](#)
- [Granting additional permissions for the IAM role \(p. 816\)](#)
- [Granting additional permissions for cross-account scenarios \(p. 819\)](#)
- [AWS KMS transaction limit considerations \(p. 820\)](#)

Specifying additional information in the replication configuration

In the replication configuration, you do the following:

- In the Destination configuration, add the symmetric encryption AWS KMS customer managed key that you want Amazon S3 to use to encrypt object replicas.
- Explicitly opt in by enabling replication of objects encrypted using KMS keys by adding the SourceSelectionCriteria element.

```
<ReplicationConfiguration>
  <Rule>
    ...
    <SourceSelectionCriteria>
      <SseKmsEncryptedObjects>
        <Status>Enabled</Status>
      </SseKmsEncryptedObjects>
    </SourceSelectionCriteria>

    <Destination>
      ...
      <EncryptionConfiguration>
        <ReplicaKmsKeyId>AWS KMS key ID for the AWS Region of the destination bucket.</ReplicaKmsKeyId>
      </EncryptionConfiguration>
    </Destination>
    ...
  </Rule>
</ReplicationConfiguration>
```

Important

The KMS key must have been created in the same AWS Region as the destination buckets. The KMS key *must* be valid. The PUT Bucket replication API doesn't check the validity of KMS keys. If you use an invalid KMS key, you will receive the 200 OK status code in response, but replication fails.

The following example shows a replication configuration, which includes optional configuration elements.

```
<?xml version="1.0" encoding="UTF-8"?>
<ReplicationConfiguration>
    <Role>arn:aws:iam::account-id:role/role-name</Role>
    <Rule>
        <ID>Rule-1</ID>
        <Priority>1</Priority>
        <Status>Enabled</Status>
        <DeleteMarkerReplication>
            <Status>Disabled</Status>
        </DeleteMarkerReplication>
        <Filter>
            <Prefix>Tax</Prefix>
        </Filter>
        <Destination>
            <Bucket>arn:aws:s3:::destination-bucket</Bucket>
            <EncryptionConfiguration>
                <ReplicaKmsKeyID>The AWS KMS key ID for the AWS Region of the destination buckets (S3 uses it to encrypt object replicas).</ReplicaKmsKeyID>
            </EncryptionConfiguration>
        </Destination>
        <SourceSelectionCriteria>
            <SseKmsEncryptedObjects>
                <Status>Enabled</Status>
            </SseKmsEncryptedObjects>
        </SourceSelectionCriteria>
    </Rule>
</ReplicationConfiguration>
```

This replication configuration has one rule. The rule applies to objects with the Tax key prefix. Amazon S3 uses the AWS KMS key ID to encrypt these object replicas.

Granting additional permissions for the IAM role

To replicate objects that are encrypted at rest under AWS Key Management Service (AWS KMS), grant the following additional permissions to the IAM role you specify in the replication configuration. You grant these permissions by updating the permission policy associated with the IAM role. Objects created with server-side encryption using customer-provided (SSE-C) encryption keys are not replicated.

- **s3:GetObjectVersionForReplication action for source objects** – Allows Amazon S3 to replicate both unencrypted objects and objects created with server-side encryption using Amazon S3 managed encryption (SSE-S3) keys or KMS keys (SSE-KMS).

Note

We recommend that you use the s3:GetObjectVersionForReplication action instead of the s3:GetObjectVersion action because it provides Amazon S3 with only the minimum permissions necessary for replication. In addition, permission for the s3:GetObjectVersion action allows replication of unencrypted and SSE-S3-encrypted objects, but not of objects created using a KMS key.

- **kms:Decrypt and kms:Encrypt AWS KMS actions:**

- kms:Decrypt permissions for the KMS key used to decrypt the source object
- kms:Encrypt permissions for the KMS key used to encrypt the object replica

Note

The key policy must allow the use of IAM policies to control access to a KMS key. If the key policy doesn't allow it, IAM policies that attempt to control access to a KMS key are ineffective. For information, see [Default key policy](#) in the AWS Key Management Service Developer Guide.

We recommend that you restrict these permissions only to the destination buckets and objects using AWS KMS condition keys. The AWS account that owns the IAM role must have permissions for these AWS

KMS actions (`kms:Encrypt` and `kms:Decrypt`) for KMS keys listed in the policy. If the KMS keys are owned by another AWS account, the KMS key owner must grant these permissions to the AWS account that owns the IAM role. For more information about managing access to these KMS keys, see [Using IAM Policies with AWS KMS](#) in the *AWS Key Management Service Developer Guide*.

Amazon S3 Bucket Keys and replication

When an S3 Bucket Key is enabled for the source and destination bucket, the encryption context will be the bucket Amazon Resource Name (ARN) and not the object ARN, for example, `arn:aws:s3:::bucket_ARN`. You need to update your IAM policies to use the bucket ARN for the encryption context. However, if an S3 Bucket Key is only enabled on the destination bucket and not the source bucket, you don't need to update your IAM policies to use the bucket ARN for the encryption context.

The example below shows the encryption context with the bucket ARN.

```
"kms:EncryptionContext:aws:s3:arn": [  
    "arn:aws:s3:::bucket_ARN"  
]
```

For more information, see [Encryption context \(p. 341\)](#) and [Changes to note before enabling an S3 Bucket Key \(p. 349\)](#).

Example policies - Using AWS KMS server-side-encryption (SSE-KMS) with replication

The following example IAM policies show statements for using AWS KMS server-side encryption with replication.

In this example, the encryption context is the object ARN. If you use SSE-KMS with an S3 Bucket Key *enabled*, you must use the bucket ARN as the encryption context. For more information, see [Encryption context \(p. 341\)](#).

Example Using AWS KMS server-side-encryption (SSE-KMS) – separate destination buckets

The following example policy shows statements for using AWS KMS with separate destination buckets.

```
{  
    "Version": "2012-10-17",  
    "Statement": [  
        {  
            "Action": ["kms:Decrypt"],  
            "Effect": "Allow",  
            "Resource": "List of AWS KMS key ARNs used to encrypt source objects.",  
            "Condition": {  
                "StringLike": {  
                    "kms:ViaService": "s3.source-bucket-region.amazonaws.com",  
                    "kms:EncryptionContext:aws:s3:arn": "arn:aws:s3:::source-bucket-name/key-prefix1/*"  
                }  
            }  
        },  
        {  
            "Action": ["kms:Encrypt"],  
            "Effect": "Allow",  
            "Resource": "AWS KMS key ARNs (for the AWS Region of the destination bucket 1). Used to encrypt object replicas created in destination bucket 1.",  
            "Condition": {  
                "StringLike": {  
                    "kms:ViaService": "s3.destination-bucket-1-region.amazonaws.com",  
                    "kms:EncryptionContext:aws:s3:arn": "arn:aws:s3:::destination-bucket-name-1/key-prefix1/*"  
                }  
            }  
        }  
    ]  
}
```

```

    }
},
{
    "Action": ["kms:Encrypt"],
    "Effect": "Allow",
    "Resource": "AWS KMS key ARNs (for the AWS Region of destination bucket 2). Used to
    encrypt object replicas created in destination bucket 2.",
    "Condition": {
        "StringLike": {
            "kms:ViaService": "s3.destination-bucket-2-region.amazonaws.com",
            "kms:EncryptionContext:aws:s3:arn": "arn:aws:s3:::destination-bucket-2-name/key-
prefix1*"
        }
    }
}
]
}

```

Example Replicating objects created with server-side encryption using Amazon S3 managed encryption keys and KMS keys

The following is a complete IAM policy that grants the necessary permissions to replicate unencrypted objects, objects created with server-side encryption using Amazon S3 managed encryption keys and KMS keys.

Note

Objects created with server-side encryption using customer-provided (SSE-C) encryption keys are not replicated.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "s3:GetReplicationConfiguration",
                "s3>ListBucket"
            ],
            "Resource": [
                "arn:aws:s3:::source-bucket"
            ]
        },
        {
            "Effect": "Allow",
            "Action": [
                "s3:GetObjectVersionForReplication",
                "s3:GetObjectVersionAcl"
            ],
            "Resource": [
                "arn:aws:s3:::source-bucket/key-prefix1*"
            ]
        },
        {
            "Effect": "Allow",
            "Action": [
                "s3:ReplicateObject",
                "s3:ReplicateDelete"
            ],
            "Resource": "arn:aws:s3:::destination-bucket/key-prefix1*"
        },
        {
            "Action": [
                "kms:Decrypt"
            ],

```

```

        "Effect": "Allow",
        "Condition": {
            "StringLike": {
                "kms:ViaService": "s3.source-bucket-region.amazonaws.com",
                "kms:EncryptionContext:aws:s3:arn": [
                    "arn:aws:s3:::source-bucket-name/key-prefix1*"
                ]
            }
        },
        "Resource": [
            "List of AWS KMS key ARNs used to encrypt source objects."
        ]
    },
    {
        "Action": [
            "kms:Encrypt"
        ],
        "Effect": "Allow",
        "Condition": {
            "StringLike": {
                "kms:ViaService": "s3.destination-bucket-region.amazonaws.com",
                "kms:EncryptionContext:aws:s3:arn": [
                    "arn:aws:s3:::destination-bucket-name/prefix1*"
                ]
            }
        },
        "Resource": [
            "AWS KMS key ARNs (for the AWS Region of the destination buckets) to use for encrypting object replicas"
        ]
    }
]
}

```

Granting additional permissions for cross-account scenarios

In a cross-account scenario, where **source** and **destination** buckets are owned by different AWS accounts, you can use a customer managed key to encrypt object replicas. However, the KMS key owner must grant the source bucket owner permission to use the KMS key.

To grant the source bucket owner permission to use the KMS key (IAM console)

1. Sign in to the AWS Management Console and open the AWS Key Management Service (AWS KMS) console at <https://console.aws.amazon.com/kms>.
2. To change the AWS Region, use the Region selector in the upper-right corner of the page.
3. To view the keys in your account that you create and manage, in the navigation pane choose **Customer managed keys**.
4. Choose the KMS key.
5. Under **General configuration**, choose the **Key policy** tab.
6. Choose **Other AWS accounts**.
7. Choose **Add another AWS account**.
8. In **arn:aws:iam::**, enter the source bucket account ID.
9. Choose **Save Changes**.

To grant the source bucket owner permission to use the KMS key (AWS CLI)

- For information, see [put-key-policy](#) in the *AWS CLI Command Reference*. For information about the underlying API, see [PutKeyPolicy](#) in the *AWS Key Management Service API Reference*.

AWS KMS transaction limit considerations

When you add many new objects with AWS KMS encryption after enabling cross-region replication (CRR), you might experience throttling (HTTP 503 Slow Down errors). Throttling occurs when the number of AWS KMS transactions per second exceeds the current limit. For more information, see [Limits in the AWS Key Management Service Developer Guide](#).

To request a limit increase, use Service Quotas. For more information, see [Amazon Web Services Limits](#). If Service Quotas isn't supported in your Region, [open an AWS Support case](#).

Getting replication status information

Replication status can help you determine the current state of an object being replicated. The replication status of a source object will return either PENDING, COMPLETED, or FAILED. The replication status of a replica will return REPLICA.

Topics

- [Replication status overview \(p. 820\)](#)
- [Replication status if replicating to multiple destination buckets \(p. 821\)](#)
- [Replication status if Amazon S3 replica modification sync is enabled \(p. 821\)](#)
- [Finding replication status \(p. 821\)](#)

Replication status overview

In replication, you have a source bucket on which you configure replication and destination where Amazon S3 replicates objects. When you request an object (using `GET object`) or object metadata (using `HEAD object`) from these buckets, Amazon S3 returns the `x-amz-replication-status` header in the response:

- When you request an object from the source bucket, Amazon S3 returns the `x-amz-replication-status` header if the object in your request is eligible for replication.

For example, suppose that you specify the object prefix `TaxDocs` in your replication configuration to tell Amazon S3 to replicate only objects with the key name prefix `TaxDocs`. Any objects that you upload that have this key name prefix—for example, `TaxDocs/document1.pdf`—will be replicated. For object requests with this key name prefix, Amazon S3 returns the `x-amz-replication-status` header with one of the following values for the object's replication status: PENDING, COMPLETED, or FAILED.

Note

If object replication fails after you upload an object, you can't retry replication. You must upload the object again. Objects transition to a FAILED state for issues such as missing replication role permissions, AWS KMS permissions, or bucket permissions. For temporary failures, such as if a bucket or Region is unavailable, replication status will not transition to FAILED, but will remain PENDING. After the resource is back online, S3 will resume replicating those objects.

- When you request an object from a destination bucket, if the object in your request is a replica that Amazon S3 created, Amazon S3 returns the `x-amz-replication-status` header with the value REPLICA.

Note

Before deleting an object from a source bucket that has replication enabled, check the object's replication status to ensure that the object has been replicated.

If lifecycle configuration is enabled on the source bucket, Amazon S3 suspends lifecycle actions until it marks the objects status as either COMPLETED or FAILED.

Replication status if replicating to multiple destination buckets

When you replicate objects to multiple destination buckets, the `x-amz-replication-status` header acts differently. The header of the source object only returns a value of `COMPLETED` when replication is successful to all destinations. The header remains at the `PENDING` value until replication has completed for all destinations. If one or more destinations fail replication, the header returns `FAILED`.

Replication status if Amazon S3 replica modification sync is enabled

When your replication rules enable Amazon S3 replica modification sync, replicas can report statuses other than `REPLICA`. If metadata changes are in the process of replicating, the `x-amz-replication-status` header returns `PENDING`. If replica modification sync fails to replicate metadata, the header returns `FAILED`. If metadata is replicated correctly, the replicas return the header `REPLICA`.

Finding replication status

To get the replication status of the objects in a bucket, you can use the Amazon S3 Inventory tool. Amazon S3 sends a CSV file to the destination bucket that you specify in the inventory configuration. You can also use Amazon Athena to query the replication status in the inventory report. For more information about Amazon S3 Inventory, see [Amazon S3 Inventory \(p. 739\)](#).

You can also find the object replication status using the console, the AWS Command Line Interface (AWS CLI), or the AWS SDK.

Using the S3 console

In the S3 console, you can view the replication status for an object on the object **Details** page under **Object management overview**.

1. Sign in to the AWS Management Console and open the Amazon S3 console at <https://console.aws.amazon.com/s3/>.
2. In the **Buckets** list, choose the bucket name.
3. In the **Objects** list, choose the object name.
The object **Details** page opens.
4. Under **Object management overview**, you can see the **Replication status**.

Using the AWS CLI

Use the `head-object` command to retrieve object metadata, as follows.

```
aws s3api head-object --bucket source-bucket --key object-key --version-id object-version-id
```

The command returns object metadata, including the `ReplicationStatus` as shown in the following example response.

```
{  
    "AcceptRanges": "bytes",  
    "ContentType": "image/jpeg",  
    "LastModified": "Mon, 23 Mar 2015 21:02:29 GMT",  
    "ContentLength": 3191,  
    "ReplicationStatus": "COMPLETED",  
}
```

```
"VersionId":"jfnW.HIMOFYid_9rGbSkmroXsFj3fqZ.",  
"ETag":"\"6805f2cfc46c0f04559748bb039d69ae\"",  
"Metadata":{  
}  
}
```

Using the AWS SDKs

The following code fragments get replication status with the AWS SDK for Java and AWS SDK for .NET, respectively.

Java

```
GetObjectMetadataRequest metadataRequest = new GetObjectMetadataRequest(bucketName,  
    key);  
ObjectMetadata metadata = s3Client.getObjectMetadata(metadataRequest);  
  
System.out.println("Replication Status : " +  
    metadata.getRawMetadataValue(Headers.OBJECT_REPLICATION_STATUS));
```

.NET

```
GetObjectMetadataRequest getmetadataRequest = new GetObjectMetadataRequest  
{  
    BucketName = sourceBucket,  
    Key        = objectKey  
};  
  
GetObjectMetadataResponse getmetadataResponse =  
    client.GetObjectMetadata(getmetadataRequest);  
Console.WriteLine("Object replication status: {0}",  
    getmetadataResponse.ReplicationStatus);
```

Troubleshooting replication

If object replicas don't appear in the destination bucket after you configure replication, use these troubleshooting tips to identify and fix issues.

- The majority of objects replicate within 15 minutes, but they can sometimes take a couple of hours. In rare cases, the replication can take longer. The time it takes Amazon S3 to replicate an object depends on several factors, including source and destination Region pair, and the size of the object. For large objects, replication can take up to several hours.

If the object that is being replicated is large, wait a while before checking to see whether it appears in the destination. You can also check the source object replication status. If the object replication status is `PENDING`, Amazon S3 has not completed the replication. If the object replication status is `FAILED`, check the replication configuration set on the source bucket.

- In the replication configuration on the source bucket, verify the following:
 - The Amazon Resource Name (ARN) of the destination buckets are correct.
 - The key name prefix is correct. For example, if you set the configuration to replicate objects with the prefix `Tax`, then only objects with key names such as `Tax/document1` or `Tax/document2` are replicated. An object with the key name `document3` is not replicated.
 - The status is `Enabled`.
- Verify that versioning has not been suspended on any bucket. Both source and destination buckets must have versioning enabled.

- If you're granting ownership of the object to the bucket owner, you must add the `s3:ObjectOwnerOverrideToBucketOwner` action in the permissions policy associated with the IAM role. This is the IAM role that you specified in the replication configuration that allows Amazon S3 to assume and replicate objects on your behalf.
- If the destination bucket is owned by another AWS account, verify that the bucket owner has a bucket policy on the destination bucket that allows the source bucket owner to replicate objects. For an example, see [Configuring replication when source and destination buckets are owned by different accounts \(p. 785\)](#).
- If an object replica doesn't appear in the destination bucket, the following might have prevented replication:
 - Amazon S3 doesn't replicate an object in a source bucket that is a replica created by another replication configuration. For example, if you set replication configuration from bucket A to bucket B to bucket C, Amazon S3 doesn't replicate object replicas in bucket B to bucket C.
 - A source bucket owner can grant other AWS accounts permission to upload objects. By default, the source bucket owner doesn't have permissions for the objects created by other accounts. The replication configuration replicates only the objects for which the source bucket owner has access permissions. The source bucket owner can grant other AWS accounts permissions to create objects conditionally, requiring explicit access permissions on those objects. For an example policy, see [Granting cross-account permissions to upload objects while ensuring the bucket owner has full control \(p. 497\)](#).
- Suppose that in the replication configuration, you add a rule to replicate a subset of objects having a specific tag. In this case, you must assign the specific tag key and value at the time of creating the object for Amazon S3 to replicate the object. If you first create an object and then add the tag to the existing object, Amazon S3 does not replicate the object.
- Replication fails if the bucket policy denies access to the replication role for any of the following actions:

Source bucket:

```
"s3:GetReplicationConfiguration",
"s3>ListBucket",
"s3:GetObjectVersionForReplication",
"s3:GetObjectVersionAcl",
"s3:GetObjectVersionTagging"
```

Destination buckets:

```
"s3:ReplicateObject",
"s3:ReplicateDelete",
"s3:ReplicateTags"
```

Related topics

[Replicating objects \(p. 753\)](#)

Additional considerations

Amazon S3 also supports bucket configurations for the following:

- Versioning – For more information, see [Using versioning in S3 buckets \(p. 638\)](#).

- Website hosting – For more information, see [Hosting a static website using Amazon S3 \(p. 1116\)](#).
- Bucket access through a bucket policy or access control list (ACL) — For more information, see [Bucket policies and user policies \(p. 411\)](#) and see [Access control list \(ACL\) overview \(p. 554\)](#).
- Log storage – For more information, [Logging requests using server access logging \(p. 978\)](#).
- Lifecycle management for objects in a bucket – For more information, see [Managing your storage lifecycle \(p. 701\)](#).

This topic explains how bucket replication configuration affects the behavior of these bucket configurations.

Topics

- [Lifecycle configuration and object replicas \(p. 824\)](#)
- [Versioning configuration and replication configuration \(p. 824\)](#)
- [Logging configuration and replication configuration \(p. 824\)](#)
- [CRR and the destination Region \(p. 825\)](#)
- [Pausing replication \(p. 825\)](#)

Lifecycle configuration and object replicas

The time it takes for Amazon S3 to replicate an object depends on the size of the object. For large objects, it can take several hours. Although it might take a while before a replica is available in the destination, it takes the same amount of time to create the replica as it took to create the corresponding object in the source bucket. If a lifecycle policy is enabled on a destination bucket, the lifecycle rules honor the original creation time of the object, not when the replica became available in the destination bucket.

Replication configuration requires the bucket to be versioning-enabled. When you enable versioning on a bucket, keep the following in mind:

- If you have an object Expiration lifecycle policy, after you enable versioning, add a `NonCurrentVersionExpiration` policy to maintain the same permanent delete behavior as before you enabled versioning.
- If you have a Transition lifecycle policy, after you enable versioning, consider adding a `NonCurrentVersionTransition` policy.

Versioning configuration and replication configuration

Both the source and destination buckets must be versioning-enabled when you configure replication on a bucket. After you enable versioning on both the source and destination buckets and configure replication on the source bucket, you will encounter the following issues:

- If you attempt to disable versioning on the source bucket, Amazon S3 returns an error. You must remove the replication configuration before you can disable versioning on the source bucket.
- If you disable versioning on the destination bucket, replication fails. The source object has the replication status `FAILED`.

Logging configuration and replication configuration

If Amazon S3 delivers logs to a bucket that has replication enabled, it replicates the log objects.

If server access logs ([Logging requests using server access logging \(p. 978\)](#)) or AWS CloudTrail logs ([Logging Amazon S3 API calls using AWS CloudTrail \(p. 962\)](#)) are enabled on your source or destination

bucket, Amazon S3 includes replication-related requests in the logs. For example, Amazon S3 logs each object that it replicates.

CRR and the destination Region

Amazon S3 Cross-Region Replication (CRR) is used to copy objects across S3 buckets in different AWS Regions. You might choose the Region for your destination bucket based on either your business needs or cost considerations. For example, inter-Region data transfer charges vary depending on the Regions that you choose.

Suppose that you chose US East (N. Virginia) (us-east-1) as the Region for your source bucket. If you choose US West (Oregon) (us-west-2) as the Region for your destination buckets, you pay more than if you choose the US East (Ohio) (us-east-2) Region. For pricing information, see "Data Transfer Pricing" in [Amazon S3 pricing](#).

There are no data transfer charges associated with Same-Region Replication (SRR).

Pausing replication

To temporarily pause replication, disable the relevant rule in the replication configuration.

If replication is enabled and you remove the IAM role that grants Amazon S3 the required permissions, replication fails. Amazon S3 reports the replication status for affected objects as `FAILED`.

Categorizing your storage using tags

Use object tagging to categorize storage. Each tag is a key-value pair.

You can add tags to new objects when you upload them, or you can add them to existing objects.

- You can associate up to 10 tags with an object. Tags that are associated with an object must have unique tag keys.
- A tag key can be up to 128 Unicode characters in length, and tag values can be up to 256 Unicode characters in length.
- The key and values are case sensitive.
- For more information about tag restrictions, see [User-Defined Tag Restrictions](#).

Examples

Consider the following tagging examples:

Example PHI information

Suppose that an object contains protected health information (PHI) data. You might tag the object using the following key-value pair.

`PHI=True`

or

`Classification=PHI`

Example Project files

Suppose that you store project files in your S3 bucket. You might tag these objects with a key named `Project` and a value, as shown following.

```
Project=Blue
```

Example Multiple tags

You can add multiple tags to an object, as shown following.

```
Project=x
Classification=confidential
```

Key name prefixes and tags

Object key name prefixes also enable you to categorize storage. However, prefix-based categorization is one-dimensional. Consider the following object key names:

```
photos/photo1.jpg
project/projectx/document.pdf
project/projecty/document2.pdf
```

These key names have the prefixes `photos/`, `project/projectx/`, and `project/projecty/`. These prefixes enable one-dimensional categorization. That is, everything under a prefix is one category. For example, the prefix `project/projectx` identifies all documents related to project x.

With tagging, you now have another dimension. If you want `photo1` in project x category, you can tag the object accordingly.

Additional benefits

In addition to data classification, tagging offers benefits such as the following:

- Object tags enable fine-grained access control of permissions. For example, you could grant an IAM user permissions to read-only objects with specific tags.
- Object tags enable fine-grained object lifecycle management in which you can specify a tag-based filter, in addition to a key name prefix, in a lifecycle rule.
- When using Amazon S3 analytics, you can configure filters to group objects together for analysis by object tags, by key name prefix, or by both prefix and tags.
- You can also customize Amazon CloudWatch metrics to display information by specific tag filters. The following sections provide details.

Important

It is acceptable to use tags to label objects containing confidential data, such as personally identifiable information (PII) or protected health information (PHI). However, the tags themselves shouldn't contain any confidential information.

Adding object tag sets to multiple Amazon S3 object with a single request

To add object tag sets to more than one Amazon S3 object with a single request, you can use S3 Batch Operations. You provide S3 Batch Operations with a list of objects to operate on. S3 Batch Operations calls the respective API to perform the specified operation. A single Batch Operations job can perform the specified operation on billions of objects containing exabytes of data.

The S3 Batch Operations feature tracks progress, sends notifications, and stores a detailed completion report of all actions, providing a fully managed, auditable, serverless experience. You can use S3 Batch Operations through the AWS Management Console, AWS CLI, AWS SDKs, or REST API. For more information, see [the section called “Batch Operations basics” \(p. 881\)](#).

For more information about object tags, see [Managing object tags \(p. 831\)](#).

API operations related to object tagging

Amazon S3 supports the following API operations that are specifically for object tagging:

Object API operations

- [PUT Object tagging](#) – Replaces tags on an object. You specify tags in the request body. There are two distinct scenarios of object tag management using this API.
 - Object has no tags – Using this API you can add a set of tags to an object (the object has no prior tags).
 - Object has a set of existing tags – To modify the existing tag set, you must first retrieve the existing tag set, modify it on the client side, and then use this API to replace the tag set.

Note

If you send this request with an empty tag set, Amazon S3 deletes the existing tag set on the object. If you use this method, you will be charged for a Tier 1 Request (PUT). For more information, see [Amazon S3 Pricing](#).

The [DELETE Object tagging](#) request is preferred because it achieves the same result without incurring charges.

- [GET Object tagging](#) – Returns the tag set associated with an object. Amazon S3 returns object tags in the response body.
- [DELETE Object tagging](#) – Deletes the tag set associated with an object.

Other API operations that support tagging

- [PUT Object](#) and [Initiate Multipart Upload](#) – You can specify tags when you create objects. You specify tags using the `x-amz-tagging` request header.
- [GET Object](#) – Instead of returning the tag set, Amazon S3 returns the object tag count in the `x-amz-tag-count` header (only if the requester has permissions to read tags) because the header response size is limited to 8 K bytes. If you want to view the tags, you make another request for the [GET Object tagging](#) API operation.
- [POST Object](#) – You can specify tags in your POST request.

As long as the tags in your request don't exceed the 8 K byte HTTP request header size limit, you can use the [PUT Object](#) API to create objects with tags. If the tags you specify exceed the header size limit, you can use this POST method in which you include the tags in the body.

[PUT Object - Copy](#) – You can specify the `x-amz-tagging-directive` in your request to direct Amazon S3 to either copy (default behavior) the tags or replace tags by a new set of tags provided in the request.

Note the following:

- S3 Object Tagging is strongly consistent. For more information, see [Amazon S3 data consistency model \(p. 6\)](#).

Additional configurations

This section explains how object tagging relates to other configurations.

Object tagging and lifecycle management

In bucket lifecycle configuration, you can specify a filter to select a subset of objects to which the rule applies. You can specify a filter based on the key name prefixes, object tags, or both.

Suppose that you store photos (raw and the finished format) in your Amazon S3 bucket. You might tag these objects as shown following.

```
phototype=raw
or
phototype=finished
```

You might consider archiving the raw photos to S3 Glacier sometime after they are created. You can configure a lifecycle rule with a filter that identifies the subset of objects with the key name prefix (`photos/`) that have a specific tag (`phototype=raw`).

For more information, see [Managing your storage lifecycle \(p. 701\)](#).

Object tagging and replication

If you configured Replication on your bucket, Amazon S3 replicates tags, provided you grant Amazon S3 permission to read the tags. For more information, see [Setting up replication \(p. 758\)](#).

Object tagging event notifications

You can set up an Amazon S3 event notification to receive notice when an object tag is added or deleted from an object. The `s3:ObjectTagging:Put` event type notifies you when a tag is PUT on an object or when an existing tag is updated. The `s3:ObjectTagging:Delete` event type notifies you when a tag is removed from an object. For more information, see [Enabling event notifications](#).

For more information about object tagging, see the following topics:

Topics

- [Tagging and access control policies \(p. 828\)](#)
- [Managing object tags \(p. 831\)](#)

Tagging and access control policies

You can also use permissions policies (bucket and user policies) to manage permissions related to object tagging. For policy actions see the following topics:

- [Example — Object operations \(p. 415\)](#)
- [Example — Bucket operations \(p. 416\)](#)

Object tags enable fine-grained access control for managing permissions. You can grant conditional permissions based on object tags. Amazon S3 supports the following condition keys that you can use to grant conditional permissions based on object tags:

- `s3:ExistingObjectTag/<tag-key>` – Use this condition key to verify that an existing object tag has the specific tag key and value.

Note

When granting permissions for the `PUT Object` and `DELETE Object` operations, this condition key is not supported. That is, you cannot create a policy to grant or deny a user permissions to delete or overwrite an object based on its existing tags.

- `s3:RequestObjectTagKeys` – Use this condition key to restrict the tag keys that you want to allow on objects. This is useful when adding tags to objects using the `PutObjectTagging` and `PutObject`, and `POST` object requests.

- **s3:RequestObjectTag/<tag-key>** – Use this condition key to restrict the tag keys and values that you want to allow on objects. This is useful when adding tags to objects using the PutObjectTagging and PutObject, and POST Bucket requests.

For a complete list of Amazon S3 service-specific condition keys, see [Amazon S3 condition key examples \(p. 420\)](#). The following permissions policies illustrate how object tagging enables fine grained access permissions management.

Example 1: Allow a user to read only the objects that have a specific tag

The following permissions policy grants a user permission to read objects, but the condition limits the read permission to only objects that have the following specific tag key and value.

```
security : public
```

Note that the policy uses the Amazon S3 condition key, **s3:ExistingObjectTag/<tag-key>** to specify the key and value.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": "s3:GetObject",
            "Resource": "arn:aws:s3:::awsexamplebucket1/*",
            "Principal": "*",
            "Condition": { "StringEquals": {"s3:ExistingObjectTag/security": "public" } }
        }
    ]
}
```

Example 2: Allow a user to add object tags with restrictions on the allowed tag keys

The following permissions policy grants a user permissions to perform the **s3:PutObjectTagging** action, which allows user to add tags to an existing object. The condition limits the tag keys that the user is allowed to use. The condition uses the **s3:RequestObjectTagKeys** condition key to specify the set of tag keys.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "s3:PutObjectTagging"
            ],
            "Resource": [
                "arn:aws:s3:::awsexamplebucket1/*"
            ],
            "Principal": {
                "CanonicalUser": [
                    "64-digit-alphanumeric-value"
                ]
            },
            "Condition": {
                "ForAllValues:StringLike": {
                    "s3:RequestObjectTagKeys": [
                        "Owner",
                        "CreationDate"
                    ]
                }
            }
        }
    ]
}
```

```

        }
    ]
}
```

The policy ensures that the tag set, if specified in the request, has the specified keys. A user might send an empty tag set in `PutObjectTagging`, which is allowed by this policy (an empty tag set in the request removes any existing tags on the object). If you want to prevent a user from removing the tag set, you can add another condition to ensure that the user provides at least one value. The `ForAnyValue` in the condition ensures at least one of the specified values must be present in the request.

```

{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "s3:PutObjectTagging"
            ],
            "Resource": [
                "arn:aws:s3:::awsexamplebucket1/*"
            ],
            "Principal": {
                "AWS": [
                    "arn:aws:iam::account-number-without-hyphens:user/username"
                ]
            },
            "Condition": {
                "ForAllValues:StringLike": {
                    "s3:RequestObjectTagKeys": [
                        "Owner",
                        "CreationDate"
                    ]
                },
                "ForAnyValue:StringLike": {
                    "s3:RequestObjectTagKeys": [
                        "Owner",
                        "CreationDate"
                    ]
                }
            }
        ]
    ]
}
```

For more information, see [Creating a Condition That Tests Multiple Key Values \(Set Operations\)](#) in the *IAM User Guide*.

Example 3: Allow a user to add object tags that include a specific tag key and value

The following user policy grants a user permissions to perform the `s3:PutObjectTagging` action, which allows user to add tags on an existing object. The condition requires the user to include a specific tag (`Project`) with value set to `X`.

```

{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "s3:PutObjectTagging"
            ],
            "Condition": {
                "StringLike": {
                    "s3:RequestObjectTagKey": "Project"
                }
            }
        }
    ]
}
```

```
],
"Resource": [
    "arn:aws:s3:::awsexamplebucket1/*"
],
"Principal": {
    "AWS": [
        "arn:aws:iam::account-number-without-hyphens:user/username"
    ]
},
"Condition": {
    "StringEquals": {
        "s3:RequestObjectTag/Project": "X"
    }
}
}
```

Managing object tags

This section explains how you can manage object tags using the AWS SDKs for Java and .NET or the Amazon S3 console.

Object tagging gives you a way to categorize storage. Each tag is a key-value pair that adheres to the following rules:

- You can associate up to 10 tags with an object. Tags associated with an object must have unique tag keys.
- A tag key can be up to 128 Unicode characters in length and tag values can be up to 256 Unicode characters in length.
- Key and tag values are case sensitive.

For more information about object tags, see [Categorizing your storage using tags \(p. 825\)](#). For more information about tag restrictions, see [User-Defined Tag Restrictions](#) in the *AWS Billing and Cost Management User Guide*.

Using the S3 console

To add tags to 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 **Buckets** list, choose the name of the bucket that contains the objects that you want to add tags to.

You can also optionally navigate to a folder.
3. In the **Objects** list, select the checkbox next to the names of the objects that you want to add tags to.
4. In the **Actions** menu, choose **Edit tags**.
5. Review the objects listed, and choose **Add tags**.
6. Each object tag is a key-value pair. Enter a **Key** and a **Value**. To add another tag, choose **Add Tag**.

You can enter up to 10 tags for an object.
7. Choose **Save changes**.

Amazon S3 adds the tags to the specified objects.

For more information, see also [Viewing object properties in the Amazon S3 console \(p. 256\)](#) and [Uploading objects \(p. 158\)](#) in this guide.

Using the AWS SDKs

Java

The following example shows how to use the AWS SDK for Java to set tags for a new object and retrieve or replace tags for an existing object. For more information about object tagging, see [Categorizing your storage using tags \(p. 825\)](#). For instructions on creating and testing a working sample, see [Testing the Amazon S3 Java Code Examples \(p. 1192\)](#).

```
import com.amazonaws.AmazonServiceException;
import com.amazonaws.SdkClientException;
import com.amazonaws.auth.profile.ProfileCredentialsProvider;
import com.amazonaws.regions.Regions;
import com.amazonaws.services.s3.AmazonS3;
import com.amazonaws.services.s3.AmazonS3ClientBuilder;
import com.amazonaws.services.s3.model.*;

import java.io.File;
import java.util.ArrayList;
import java.util.List;

public class ManagingObjectTags {

    public static void main(String[] args) {
        Regions clientRegion = Regions.DEFAULT_REGION;
        String bucketName = "**** Bucket name ****";
        String keyName = "**** Object key ****";
        String filePath = "**** File path ****";

        try {
            AmazonS3 s3Client = AmazonS3ClientBuilder.standard()
                .withCredentials(new ProfileCredentialsProvider())
                .withRegion(clientRegion)
                .build();

            // Create an object, add two new tags, and upload the object to Amazon S3.
            PutObjectRequest putRequest = new PutObjectRequest(bucketName, keyName, new
File(filePath));
            List<Tag> tags = new ArrayList<Tag>();
            tags.add(new Tag("Tag 1", "This is tag 1"));
            tags.add(new Tag("Tag 2", "This is tag 2"));
            putRequest.setTagging(new ObjectTagging(tags));
            PutObjectResult putResult = s3Client.putObject(putRequest);

            // Retrieve the object's tags.
            GetObjectTaggingRequest getTaggingRequest = new
GetObjectTaggingRequest(bucketName, keyName);
            GetObjectTaggingResult getTagsResult =
s3Client.getObjectTagging(getTaggingRequest);

            // Replace the object's tags with two new tags.
            List<Tag> newTags = new ArrayList<Tag>();
            newTags.add(new Tag("Tag 3", "This is tag 3"));
            newTags.add(new Tag("Tag 4", "This is tag 4"));
            s3Client.setObjectTagging(new SetObjectTaggingRequest(bucketName, keyName,
new ObjectTagging(newTags)));
        } catch (AmazonServiceException e) {
            // The call was transmitted successfully, but Amazon S3 couldn't process
            // it, so it returned an error response.
            e.printStackTrace();
        }
    }
}
```

```
        } catch (SdkClientException e) {
            // Amazon S3 couldn't be contacted for a response, or the client
            // couldn't parse the response from Amazon S3.
            e.printStackTrace();
        }
    }
}
```

.NET

The following example shows how to use the AWS SDK for .NET to set the tags for a new object and retrieve or replace the tags for an existing object. For more information about object tagging, see [Categorizing your storage using tags \(p. 825\)](#).

For instructions on how to create and test a working sample, see [Running the Amazon S3 .NET Code Examples \(p. 1193\)](#).

```
using Amazon;
using Amazon.S3;
using Amazon.S3.Model;
using System;
using System.Collections.Generic;
using System.Threading.Tasks;

namespace Amazon.DocSamples.S3
{
    public class ObjectTagsTest
    {
        private const string bucketName = "*** bucket name ***";
        private const string keyName = "*** key name for the new object ***";
        private const string filePath = @ "*** file path ***";
        // Specify your bucket region (an example region is shown).
        private static readonly RegionEndpoint bucketRegion = RegionEndpoint.USWest2;
        private static IAmazonS3 client;

        public static void Main()
        {
            client = new AmazonS3Client(bucketRegion);
            PutObjectWithTagsTestAsync().Wait();
        }

        static async Task PutObjectWithTagsTestAsync()
        {
            try
            {
                // 1. Put an object with tags.
                var putRequest = new PutObjectRequest
                {
                    BucketName = bucketName,
                    Key = keyName,
                    FilePath = filePath,
                    TagSet = new List<Tag>{
                        new Tag { Key = "Keyx1", Value = "Value1" },
                        new Tag { Key = "Keyx2", Value = "Value2" }
                    }
                };

                PutObjectResponse response = await client.PutObjectAsync(putRequest);
                // 2. Retrieve the object's tags.
                GetObjectTaggingRequest getTagsRequest = new GetObjectTaggingRequest
                {
                    BucketName = bucketName,
                    Key = keyName
                };
            }
        }
    }
}
```

```
};

GetObjectTaggingResponse objectTags = await
client.GetObjectTaggingAsync(getTagsRequest);
for (int i = 0; i < objectTags.Tagging.Count; i++)
    Console.WriteLine("Key: {0}, Value: {1}",
objectTags.Tagging[i].Key, objectTags.Tagging[i].Value);

// 3. Replace the tagset.

Tagging newTagSet = new Tagging();
newTagSet.TagSet = new List<Tag>{
    new Tag { Key = "Key3", Value = "Value3" },
    new Tag { Key = "Key4", Value = "Value4" }
};

PutObjectTaggingRequest putObjTagsRequest = new
PutObjectTaggingRequest()
{
    BucketName = bucketName,
    Key = keyName,
    Tagging = newTagSet
};
PutObjectTaggingResponse response2 = await
client.PutObjectTaggingAsync(putObjTagsRequest);

// 4. Retrieve the object's tags.
GetObjectTaggingRequest getTagsRequest2 = new
GetObjectTaggingRequest();
getTagsRequest2.BucketName = bucketName;
getTagsRequest2.Key = keyName;
GetObjectTaggingResponse objectTags2 = await
client.GetObjectTaggingAsync(getTagsRequest2);
for (int i = 0; i < objectTags2.Tagging.Count; i++)
    Console.WriteLine("Key: {0}, Value: {1}",
objectTags2.Tagging[i].Key, objectTags2.Tagging[i].Value);

}
catch (AmazonS3Exception e)
{
    Console.WriteLine(
        "Error encountered ***. Message:'{0}' when writing an object"
        , e.Message);
}
catch (Exception e)
{
    Console.WriteLine(
        "Encountered an error. Message:'{0}' when writing an object"
        , e.Message);
}
}

}
```

Using cost allocation S3 bucket tags

To track the storage cost or other criteria for individual projects or groups of projects, label your Amazon S3 buckets using cost allocation tags. A *cost allocation tag* is a key-value pair that you associate with an S3 bucket. After you activate cost allocation tags, AWS uses the tags to organize your resource costs on

your cost allocation report. Cost allocation tags can only be used to label buckets. For information about tags used for labeling objects, see [Categorizing your storage using tags \(p. 825\)](#).

The *cost allocation report* lists the AWS usage for your account by product category and AWS Identity and Access Management (IAM) user. The report contains the same line items as the detailed billing report (see [Understanding your AWS billing and usage reports for Amazon S3 \(p. 839\)](#)) and additional columns for your tag keys.

AWS provides two types of cost allocation tags, an AWS-generated tag and user-defined tags. AWS defines, creates, and applies the AWS-generated `createdBy` tag for you after an Amazon S3 `CreateBucket` event. You define, create, and apply *user-defined* tags to your S3 bucket.

You must activate both types of tags separately in the Billing and Cost Management console before they can appear in your billing reports. For more information about AWS-generated tags, see [AWS-Generated Cost Allocation Tags](#).

- To create tags in the console, see [Viewing the properties for an S3 bucket \(p. 124\)](#).
- To create tags using the Amazon S3 API, see [PUT Bucket tagging](#) in the *Amazon Simple Storage Service API Reference*.
- To create tags using the AWS CLI, see [put-bucket-tagging](#) in the AWS CLI Command Reference.
- For more information about activating tags, see [Using cost allocation tags](#) in the *AWS Billing User Guide*.

User-defined cost allocation tags

A user-defined cost allocation tag has the following components:

- The tag key. The tag key is the name of the tag. For example, in the tag project/Trinity, project is the key. The tag key is a case-sensitive string that can contain 1 to 128 Unicode characters.
- The tag value. The tag value is a required string. For example, in the tag project/Trinity, Trinity is the value. The tag value is a case-sensitive string that can contain from 0 to 256 Unicode characters.

For details on the allowed characters for user-defined tags and other restrictions, see [User-Defined Tag Restrictions](#) in the *AWS Billing User Guide*. For more information about user-defined tags, see [User-Defined Cost Allocation Tags](#) in the *AWS Billing User Guide*.

S3 bucket tags

Each S3 bucket has a tag set. A *tag set* contains all of the tags that are assigned to that bucket. A tag set can contain as many as 50 tags, or it can be empty. Keys must be unique within a tag set, but values in a tag set don't have to be unique. For example, you can have the same value in tag sets named project/Trinity and cost-center/Trinity.

Within a bucket, if you add a tag that has the same key as an existing tag, the new value overwrites the old value.

AWS doesn't apply any semantic meaning to your tags. We interpret tags strictly as character strings.

To add, list, edit, or delete tags, you can use the Amazon S3 console, the AWS Command Line Interface (AWS CLI), or the Amazon S3 API.

More Info

- [Using Cost Allocation Tags](#) in the *AWS Billing User Guide*
- [Understanding your AWS billing and usage reports for Amazon S3 \(p. 839\)](#)
- [AWS Billing reports for Amazon S3 \(p. 836\)](#)

Billing and usage reporting for S3 buckets

When using Amazon Simple Storage Service (Amazon S3), you don't have to pay any upfront fees or commit to how much content you'll store. As with the other Amazon Web Services (AWS) services, you pay as you go and pay only for what you use.

AWS provides the following reports for Amazon S3:

- **Billing reports** – Multiple reports that provide high-level views of all of the activity for the AWS services that you're using, including Amazon S3. AWS always bills the owner of the S3 bucket for Amazon S3 fees, unless the bucket was created as a Requester Pays bucket. For more information about Requester Pays, see [Using Requester Pays buckets for storage transfers and usage \(p. 144\)](#). For more information about billing reports, see [AWS Billing reports for Amazon S3 \(p. 836\)](#).
- **Usage report** – A summary of activity for a specific service, aggregated by hour, day, or month. You can choose which usage type and operation to include. You can also choose how the data is aggregated. For more information, see [AWS usage report for Amazon S3 \(p. 838\)](#).

The following topics provide information about billing and usage reporting for Amazon S3.

Topics

- [AWS Billing reports for Amazon S3 \(p. 836\)](#)
- [AWS usage report for Amazon S3 \(p. 838\)](#)
- [Understanding your AWS billing and usage reports for Amazon S3 \(p. 839\)](#)

AWS Billing reports for Amazon S3

Your monthly bill from AWS separates your usage information and cost by AWS service and function. There are several AWS billing reports available, the monthly report, the cost allocation report, and detailed billing reports. For information about how to see your billing reports, see [Viewing Your Bill](#) in the [AWS Billing User Guide](#).

You may set up AWS Cost and Usage Report to track your AWS usage and provide estimated charges associated with your account. For more information, see [What are AWS Cost and Usage Report?](#) in the [AWS Cost and Usage Reports Guide](#).

You can also download a usage report that gives more detail about your Amazon S3 storage usage than the billing reports. For more information, see [AWS usage report for Amazon S3 \(p. 838\)](#).

The following table lists the charges associated with Amazon S3 usage.

Amazon S3 usage charges

Charge	Comments
Storage	You pay for storing objects in your S3 buckets. The rate you're charged depends on your objects' size, how long you stored the objects during the month, and the storage class—S3 Standard, S3 Intelligent-Tiering, S3 Standard-IA (IA for infrequent access), S3 One Zone-IA, S3 Glacier Instant Retrieval, S3 Glacier Flexible Retrieval, S3 Glacier Deep Archive or Reduced Redundancy Storage (RRS). For more information about storage classes, see Using Amazon S3 storage classes (p. 688) .

Charge	Comments
	Be aware that if you have versioning enabled, you're charged for each version of an object that is retained. For more information about versioning, see How S3 Versioning works (p. 640) .
Monitoring and Automation	You pay a monthly monitoring and automation fee per object stored in the S3 Intelligent-Tiering storage class to monitor access patterns and move objects between access tiers in S3 Intelligent-Tiering.
Requests	You pay for requests, for example, GET requests, made against your S3 buckets and objects. This includes lifecycle requests. The rates for requests depend on what kind of request you're making. For information about request pricing, see Amazon S3 Pricing .
Retrievals	You pay for retrieving objects that are stored in S3 Standard-IA, S3 One Zone-IA, S3 Glacier Instant Retrieval, S3 Glacier Flexible Retrieval, and S3 Glacier Deep Archive storage.
Early Deletes	If you delete an object stored in S3 Standard-IA, S3 One Zone-IA, S3 Glacier Instant Retrieval, S3 Glacier Flexible Retrieval, or S3 Glacier Deep Archive storage before the minimum storage commitment has passed, you pay an early deletion fee for that object.
Storage Management	You pay for the storage management features (Amazon S3 Inventory, analytics, and object tagging) that are enabled on your account's buckets.
Bandwidth	<p>You pay for all bandwidth into and out of Amazon S3, except for the following:</p> <ul style="list-style-type: none"> • Data transferred in from the internet • Data transferred out to an Amazon Elastic Compute Cloud (Amazon EC2) instance, when the instance is in the same AWS Region as the S3 bucket • Data transferred out to Amazon CloudFront (CloudFront) <p>You also pay a fee for any data transferred using Amazon S3 Transfer Acceleration.</p>

For detailed information on Amazon S3 usage charges for storage, data transfer, and services, see [Amazon S3 Pricing](#) and the [Amazon S3 FAQ](#).

For information on understanding codes and abbreviations used in the billing and usage reports for Amazon S3, see [Understanding your AWS billing and usage reports for Amazon S3 \(p. 839\)](#).

More Info

- [AWS usage report for Amazon S3 \(p. 838\)](#)
- [Using cost allocation S3 bucket tags \(p. 834\)](#)
- [AWS Billing and Cost Management](#)
- [Amazon S3 Pricing](#)
- [Amazon S3 FAQ](#)
- [S3 Glacier Pricing](#)

AWS usage report for Amazon S3

When you download a usage report, you can choose to aggregate usage data by hour, day, or month. The Amazon S3 usage report lists operations by usage type and AWS Region. For more detailed reports about your Amazon S3 storage usage, download dynamically generated AWS usage reports. You can choose which usage type, operation, and time period to include. You can also choose how the data is aggregated.

The Amazon S3 usage report includes the following information:

- **Service** – Amazon S3
- **Operation** – The operation performed on your bucket or object. For a detailed explanation of Amazon S3 operations, see [Tracking Operations in Your Usage Reports \(p. 851\)](#).
- **UsageType** – One of the following values:
 - A code that identifies the type of storage
 - A code that identifies the type of request
 - A code that identifies the type of retrieval
 - A code that identifies the type of data transfer
 - A code that identifies early deletions from S3 Intelligent-Tiering, S3 Standard-IA, S3 One Zone-Infrequent Access (S3 One Zone-IA), S3 Glacier Flexible Retrieval, or S3 Glacier Deep Archive storage
- **StorageObjectCount** – The count of objects stored within a given bucket

For a detailed explanation of Amazon S3 usage types, see [Understanding your AWS billing and usage reports for Amazon S3 \(p. 839\)](#).

- **Resource** – The name of the bucket associated with the listed usage.
- **StartTime** – Start time of the day that the usage applies to, in Coordinated Universal Time (UTC).
- **EndTime** – End time of the day that the usage applies to, in Coordinated Universal Time (UTC).
- **UsageValue** – One of the following volume values. The typical unit of measurement for data is gigabytes (GB). However, depending on the service and the report, terabytes (TB) might appear instead.
 - The number of requests during the specified time period
 - The amount of data transferred
 - The amount of data stored in a given hour
 - The amount of data associated with restorations from S3 Standard-IA, S3 One Zone-IA, S3 Glacier Flexible Retrieval, or S3 Glacier Deep Archive storage

Tip

For detailed information about every request that Amazon S3 receives for your objects, turn on server access logging for your buckets. For more information, see [Logging requests using server access logging \(p. 978\)](#).

You can download a usage report as an XML or a comma-separated values (CSV) file. The following is an example CSV usage report opened in a spreadsheet application.

Service	Operation	UsageType	Resource	StartTime	EndTime	UsageValue
AmazonS3	HeadBucket	USW2-C3DataTransfer-Out-Bytes	admin-created3	6/1/2017 0:00	7/1/2017 0:00	15309
AmazonS3	PutObject	USW2-C3DataTransfer-In-Bytes	admin-created3	6/1/2017 0:00	7/1/2017 0:00	19062
AmazonS3	HeadBucket	USW2-Requests-Tier2	admin-created3	6/1/2017 0:00	7/1/2017 0:00	68
AmazonS3	PutObjectForReplica	USW1-Requests-SIA-Tier1	ca-example-bucket	6/1/2017 0:00	7/1/2017 0:00	178294
AmazonS3	PutObjectForReplica	USW1-USW2-AWS-In-Bytes	ca-example-bucket	6/1/2017 0:00	7/1/2017 0:00	387929083
AmazonS3	GetObjectForReplica	USW2-Requests-NoCharge	admin-created3	6/1/2017 0:00	7/1/2017 0:00	108
AmazonS3	GetObjectForReplica	USW2-USW1-AWS-Out-Bytes	my-test-bucket-bash	6/1/2017 0:00	7/1/2017 0:00	387910021

For more information, see [Understanding your AWS billing and usage reports for Amazon S3 \(p. 839\)](#).

Downloading the AWS Usage Report

You can download a usage report as an .xml or a .csv file.

To download the usage report

1. Sign in to the AWS Management Console and open the Amazon S3 console at <https://console.aws.amazon.com/s3/>.
2. In the title bar, choose your AWS Identity and Access Management (IAM) user name, and then choose **My Billing Dashboard**.
3. In the navigation pane, choose **AWS Cost & Usage Reports**.
4. In the **Other Reports** section, choose **AWS Usage Report**.
5. For **Services**, choose **Amazon Simple Storage Service**.
6. For **Download Usage Report**, choose the following settings:
 - **Usage Types** – For a detailed explanation of Amazon S3 usage types, see [Understanding your AWS billing and usage reports for Amazon S3 \(p. 839\)](#).
 - **Operation** – For a detailed explanation of Amazon S3 operations, see [Tracking Operations in Your Usage Reports \(p. 851\)](#).
 - **Time Period** – The time period that you want the report to cover.
 - **Report Granularity** – Whether you want the report to include subtotals by the hour, by the day, or by the month.
7. Choose the **Download** format and follow the prompts to open or save the report.

More Info

- [Understanding your AWS billing and usage reports for Amazon S3 \(p. 839\)](#)
- [AWS Billing reports for Amazon S3 \(p. 836\)](#)

Understanding your AWS billing and usage reports for Amazon S3

Amazon S3 billing and usage reports use codes and abbreviations. For usage types in the table that follows, replace **region**, **region1**, and **region2** with abbreviations from this list:

- **AP1**: Asia Pacific (Hong Kong)
- **APN1**: Asia Pacific (Tokyo)
- **APN2**: Asia Pacific (Seoul)
- **APN3**: Asia Pacific (Osaka)

- **APS1:** Asia Pacific (Singapore)
- **APS2:** Asia Pacific (Sydney)
- **APS3:** Asia Pacific (Mumbai)
- **APS4:** Asia Pacific (Jakarta)
- **CAN1:** Canada (Central)
- **CPT:** Africa (Cape Town)
- **EUN1:** Europe (Stockholm)
- **EUC1:** Europe (Frankfurt)
- **EU:** Europe (Ireland)
- **EUS1:** Europe (Milan)
- **EUW2:** Europe (London)
- **EUW3:** Europe (Paris)
- **MES1:** Middle East (Bahrain)
- **SAE1:** South America (São Paulo)
- **UGW1:** AWS GovCloud (US-West)
- **UGE1:** AWS GovCloud (US-East)
- **USE1 (or no prefix):** US East (N. Virginia)
- **USE2:** US East (Ohio)
- **USW1:** US West (N. California)
- **USW2:** US West (Oregon)

For information about pricing by AWS Region, see [Amazon S3 Pricing](#).

The first column in the following table lists usage types that appear in your billing and usage reports. The typical unit of measurement for data is gigabytes (GB). However, depending on the service and the report, terabytes (TB) might appear instead.

Usage Types

Usage Type	Units	Granularity	Description
<i>region1-region2-AWS-In-ABytes</i>	GB	Hourly	The amount of accelerated data transferred to AWS Region1 from AWS Region2
<i>region1-region2-AWS-In-ABytes-T1</i>	GB	Hourly	The amount of T1 accelerated data transferred to AWS Region1 from AWS Region2, where T1 refers to CloudFront requests to POPs in the United States, Europe, and Japan
<i>region1-region2-AWS-In-ABytes-T2</i>	GB	Hourly	The amount of T2 accelerated data transferred to AWS Region1 from AWS Region2, where T2 refers to CloudFront requests to POPs in all other AWS edge locations

Usage Type	Units	Granularity	Description
<i>region1-region2-AWS-In-Bytes</i>	GB	Hourly	The amount of data transferred to AWS Region1 from AWS Region2
<i>region1-region2-AWS-Out-ABytes</i>	GB	Hourly	The amount of accelerated data transferred from AWS Region1 to AWS Region2
<i>region1-region2-AWS-Out-ABytes-T1</i>	GB	Hourly	The amount of T1 accelerated data transferred from AWS Region1 from AWS Region2, where T1 refers to CloudFront requests to POPs in the United States, Europe, and Japan
<i>region1-region2-AWS-Out-ABytes-T2</i>	GB	Hourly	The amount of T2 accelerated data transferred from AWS Region1 to AWS Region2, where T2 refers to CloudFront requests to POPs in all other AWS edge locations
<i>region1-region2-AWS-Out-Bytes</i>	GB	Hourly	The amount of data transferred from AWS Region1 to AWS Region2
<i>region-BatchOperations-Jobs</i>	Count	Hourly	The number of S3 Batch Operations jobs performed
<i>region-BatchOperations-Objects</i>	Count	Hourly	The number of object operations performed by S3 Batch Operations
<i>region-Bulk-Retrieval-Bytes</i>	GB	Hourly	The amount of data retrieved with Bulk S3 Glacier Flexible Retrieval or S3 Glacier Deep Archive requests
<i>region-BytesDeleted-GDA</i>	GB	Monthly	The amount of data deleted by a DeleteObject operation from S3 Glacier Deep Archive storage
<i>region-BytesDeleted-GIR</i>	GB	Monthly	The amount of data deleted by a DeleteObject operation from S3 Glacier Instant Retrieval storage.
<i>region-BytesDeleted-GLACIER</i>	GB	Monthly	The amount of data deleted by a DeleteObject operation from S3 Glacier Flexible Retrieval storage

Usage Type	Units	Granularity	Description
<i>region</i> -BytesDeleted-INT	GB	Monthly	The amount of data deleted by a DeleteObject operation from S3 Intelligent-Tiering storage
<i>region</i> -BytesDeleted-RRS	GB	Monthly	The amount of data deleted by a DeleteObject operation from Reduced Redundancy Storage (RRS) storage
<i>region</i> -BytesDeleted-SIA	GB	Monthly	The amount of data deleted by a DeleteObject operation from S3 Standard-IA storage
<i>region</i> -BytesDeleted-STANDARD	GB	Monthly	The amount of data deleted by a DeleteObject operation from S3 Standard storage
<i>region</i> -BytesDeleted-ZIA	GB	Monthly	The amount of data deleted by a DeleteObject operation from S3 One Zone-IA storage
<i>region</i> -C3DataTransfer-In-Bytes	GB	Hourly	The amount of data transferred into Amazon S3 from Amazon EC2 within the same AWS Region
<i>region</i> -C3DataTransfer-Out-Bytes	GB	Hourly	The amount of data transferred from Amazon S3 to Amazon EC2 within the same AWS Region
<i>region</i> -CloudFront-In-Bytes	GB	Hourly	The amount of data transferred into an AWS Region from a CloudFront distribution
<i>region</i> -CloudFront-Out-Bytes	GB	Hourly	The amount of data transferred from an AWS Region to a CloudFront distribution
<i>region</i> -DataTransfer-In-Bytes	GB	Hourly	The amount of data transferred into Amazon S3 from the internet
<i>region</i> -DataTransfer-Out-Bytes	GB	Hourly	The amount of data transferred from Amazon S3 to the internet ¹

Usage Type	Units	Granularity	Description
<i>region</i> -DataTransfer-Regional-Bytes	GB	Hourly	The amount of data transferred from Amazon S3 to AWS resources within the same AWS Region
<i>region</i> -EarlyDelete-ByteHrs	GB-Hours	Hourly	Prorated storage usage for objects deleted from S3 Glacier Flexible Retrieval storage before the 90-day minimum commitment ended ²
<i>region</i> -EarlyDelete-GDA	GB-Hours	Hourly	Prorated storage usage for objects deleted from S3 Glacier Deep Archive storage before the 180-day minimum commitment ended ²
<i>region</i> -EarlyDelete-GIR	GB-Hours	Hourly	Prorated storage usage for objects deleted from S3 Glacier Instant Retrieval before the 90-day minimum commitment ended.
<i>region</i> -EarlyDelete-GIR-SmObjects	GB-Hours	Hourly	Prorated storage usage for small objects (smaller than 128 KB) that were deleted from S3 Glacier Instant Retrieval before the 90-day minimum commitment ended.
<i>region</i> -EarlyDelete-SIA	GB-Hours	Hourly	Prorated storage usage for objects deleted from S3 Standard-IA before the 30-day minimum commitment ended ³
<i>region</i> -EarlyDelete-SIA-SmObjects	GB-Hours	Hourly	Prorated storage usage for small objects (smaller than 128 KB) that were deleted from S3 Standard-IA before the 30-day minimum commitment ended ³
<i>region</i> -EarlyDelete-ZIA	GB-Hours	Hourly	Prorated storage usage for objects deleted from S3 One Zone-IA before the 30-day minimum commitment ended ³

Usage Type	Units	Granularity	Description
<i>region</i> -EarlyDelete-ZIA-SmObjects	GB-Hours	Hourly	Prorated storage usage for small objects (smaller than 128 KB) that were deleted from S3 One Zone-IA before the 30-day minimum commitment ended ³
<i>region</i> -Expedited-Retrieval-Bytes	GB	Hourly	The amount of data retrieved with Expedited S3 Glacier Flexible Retrieval requests
<i>region</i> -Inventory-ObjectsListed	Objects	Hourly	The number of objects listed for an object group (objects are grouped by bucket or prefix) with an inventory list
<i>region</i> -Monitoring-Automation-INT	Objects	Hourly	The number of unique objects monitored and auto-tiered in the S3 Intelligent-Tiering storage class
<i>region</i> -OverwriteBytes-Copy-GDA	GB	Monthly	The amount of data overwritten by a CopyObject operation from S3 Glacier Deep Archive storage
<i>region</i> -OverwriteBytes-Copy-GIR	GB	Monthly	The amount of data overwritten by a CopyObject operation from S3 Glacier Instant Retrieval storage.
<i>region</i> -OverwriteBytes-Copy-GLACIER	GB	Monthly	The amount of data overwritten by a CopyObject operation from S3 Glacier Flexible Retrieval storage
<i>region</i> -OverwriteBytes-Copy-INT	GB	Monthly	The amount of data overwritten by a CopyObject operation from S3 Intelligent-Tiering storage
<i>region</i> -OverwriteBytes-Copy-RRS	GB	Monthly	The amount of data overwritten by a CopyObject operation from Reduced Redundancy Storage (RRS) storage

Usage Type	Units	Granularity	Description
<i>region</i> -OverwriteBytes-Copy-SIA	GB	Monthly	The amount of data overwritten by a CopyObject operation from S3 Standard-IA storage
<i>region</i> -OverwriteBytes-Copy-STANDARD	GB	Monthly	The amount of data overwritten by a CopyObject operation from S3 Standard storage
<i>region</i> -OverwriteBytes-Copy-ZIA	GB	Monthly	The amount of data overwritten by a CopyObject operation from S3 One Zone-IA storage
<i>region</i> -OverwriteBytes-Put-GDA	GB	Monthly	The amount of data overwritten by a PutObject operation from S3 Glacier Deep Archive storage
<i>region</i> -OverwriteBytes-Put-GIR	GB	Monthly	The amount of data overwritten by a PutObject operation from S3 Glacier Instant Retrieval storage.
<i>region</i> -OverwriteBytes-Put-GLACIER	GB	Monthly	The amount of data overwritten by a PutObject operation from S3 Glacier Flexible Retrieval storage
<i>region</i> -OverwriteBytes-Put-INT	GB	Monthly	The amount of data overwritten by a PutObject operation from S3 Intelligent-Tiering storage
<i>region</i> -OverwriteBytes-Put-RRS	GB	Monthly	The amount of data overwritten by a PutObject operation from Reduced Redundancy Storage (RRS) storage
<i>region</i> -OverwriteBytes-Put-SIA	GB	Monthly	The amount of data overwritten by a PutObject operation from S3 Standard-IA storage
<i>region</i> -OverwriteBytes-Put-STANDARD	GB	Monthly	The amount of data overwritten by a PutObject operation from S3 Standard storage
<i>region</i> -OverwriteBytes-Put-ZIA	GB	Monthly	The amount of data overwritten by a PutObject operation from S3 One Zone-IA storage

Usage Type	Units	Granularity	Description
<i>region</i> -Requests-GDA-Tier1	Count	Hourly	The number of PUT, COPY, POST, InitiateMultipartUpload, UploadPart, or CompleteMultipartUpload requests on DEEP Archive objects
<i>region</i> -Requests-GDA-Tier2	Count	Hourly	The number of GET and HEAD requests
<i>region</i> -Requests-GDA-Tier3	Count	Hourly	The number of S3 Glacier Deep Archive standard restore requests
<i>region</i> -Requests-GDA-Tier5	Count	Hourly	The number of Bulk S3 Glacier Deep Archive restore requests
<i>region</i> -Requests-GIR-Tier1	Count	Hourly	The number of PUT, COPY, or POST requests on S3 Glacier Instant Retrieval objects.
<i>region</i> -Requests-GIR-Tier2	Count	Hourly	The number of GET and all other non-GIR-Tier1 requests on S3 Glacier Instant Retrieval objects.
<i>region</i> -Requests-GLACIER-Tier1	Count	Hourly	The number of PUT, COPY, POST, InitiateMultipartUpload, UploadPart, or CompleteMultipartUpload requests on S3 Glacier Flexible Retrieval objects
<i>region</i> -Requests-GLACIER-Tier2	Count	Hourly	The number of GET and all other requests not listed on S3 Glacier Flexible Retrieval objects
<i>region</i> -Requests-INT-Tier1	Count	Hourly	The number of PUT, COPY, or POST requests on S3 Intelligent-Tiering objects
<i>region</i> -Requests-INT-Tier2	Count	Hourly	The number of GET and all other non-Tier1 requests for S3 Intelligent-Tiering objects
<i>region</i> -Requests-SIA-Tier1	Count	Hourly	The number of PUT, COPY, or POST requests on S3 Standard-IA objects

Usage Type	Units	Granularity	Description
<i>region</i> -Requests-SIA-Tier2	Count	Hourly	The number of GET and all other non-SIA-Tier1 requests on S3 Standard-IA objects
<i>region</i> -Requests-Tier1	Count	Hourly	The number of PUT, COPY, or POST requests for STANDARD, RRS, and tags, plus LIST requests for all buckets and objects
<i>region</i> -Requests-Tier2	Count	Hourly	The number of GET and all other non-Tier1 requests
<i>region</i> -Requests-Tier3	Count	Hourly	The number of lifecycle requests to S3 Glacier Flexible Retrieval or S3 Glacier Deep Archive and standard S3 Glacier Flexible Retrieval restore requests
<i>region</i> -Requests-Tier4	Count	Hourly	The number of lifecycle transitions to S3 Glacier Instant Retrieval, S3 Intelligent-Tiering, S3 Standard-IA, or S3 One Zone-IA storage
<i>region</i> -Requests-Tier5	Count	Hourly	The number of Bulk S3 Glacier Flexible Retrieval restore requests
<i>region</i> -Requests-Tier6	Count	Hourly	The number of Expedited S3 Glacier Flexible Retrieval restore requests
<i>region</i> -Requests-ZIA-Tier1	Count	Hourly	The number of PUT, COPY, or POST requests on S3 One Zone-IA objects
<i>region</i> -Requests-ZIA-Tier2	Count	Hourly	The number of GET and all other non-ZIA-Tier1 requests on S3 One Zone-IA objects
<i>region</i> -Retrieval-GIR	GB	Hourly	The amount of data retrieved from S3 Glacier Instant Retrieval storage.
<i>region</i> -Retrieval-SIA	GB	Hourly	The amount of data retrieved from S3 Standard-IA storage
<i>region</i> -Retrieval-ZIA	GB	Hourly	The amount of data retrieved from S3 One Zone-IA storage

Usage Type	Units	Granularity	Description
<i>region</i> -S3G-DataTransfer-In-Bytes	GB	Hourly	The amount of data transferred into Amazon S3 to restore objects from S3 Glacier Flexible Retrieval or S3 Glacier Deep Archive storage
<i>region</i> -S3G-DataTransfer-Out-Bytes	GB	Hourly	The amount of data transferred from Amazon S3 to transition objects to S3 Glacier Flexible Retrieval or S3 Glacier Deep Archive storage
<i>region</i> -Select-Returned-Bytes	GB	Hourly	The amount of data returned with Select requests from S3 Standard storage
<i>region</i> -Select-Returned-GIR-Bytes	GB	Hourly	The amount of data returned with Select requests from S3 Glacier Instant Retrieval storage.
<i>region</i> -Select-Returned-INT-Bytes	GB	Hourly	The amount of data returned with Select requests from S3 Intelligent-Tiering storage
<i>region</i> -Select-Returned-SIA-Bytes	GB	Hourly	The amount of data returned with Select requests from S3 Standard-IA storage
<i>region</i> -Select-Returned-ZIA-Bytes	GB	Hourly	The amount of data returned with Select requests from S3 One Zone-IA storage
<i>region</i> -Select-Scanned-Bytes	GB	Hourly	The amount of data scanned with Select requests from S3 Standard storage
<i>region</i> -Select-Scanned-GIR-Bytes	GB	Hourly	The amount of data scanned with Select requests from S3 Glacier Instant Retrieval storage.
<i>region</i> -Select-Scanned-INT-Bytes	GB	Hourly	The amount of data scanned with Select requests from S3 Intelligent-Tiering storage

Usage Type	Units	Granularity	Description
<i>region</i> -Select-Scanned-SIA-Bytes	GB	Hourly	The amount of data scanned with Select requests from S3 Standard-IA storage
<i>region</i> -Select-Scanned-ZIA-Bytes	GB	Hourly	The amount of data scanned with Select requests from S3 One Zone-IA storage
<i>region</i> -Standard-Retrieval-Bytes	GB	Hourly	The amount of data retrieved with standard S3 Glacier Flexible Retrieval or S3 Glacier Deep Archive requests
<i>region</i> -StorageAnalytics-ObjCount	Objects	Hourly	The number of unique objects monitored in each Storage Class Analysis configuration.
<i>region</i> -StorageLens-ObjCount	Objects	Daily	The number of unique objects in each S3 Storage Lens dashboard that are tracked by S3 Storage Lens advanced metrics and recommendations.
<i>region</i> -StorageLensFreeTier-ObjCount	Objects	Daily	The number of unique objects in each S3 Storage Lens dashboard that are tracked by S3 Storage Lens usage metrics.
<i>region</i> -TagStorage-TagHrs	Tag-Hours	Daily	The total of tags on all objects in the bucket reported by hour
<i>region</i> -TimedStorage-ByteHrs	GB-Hours	Daily	The number of GB-hours that data was stored in S3 Standard storage
<i>region</i> -TimedStorage-GDA-ByteHrs	GB-Hours	Daily	The number of GB-hours that data was stored in S3 Glacier Deep Archive storage
<i>region</i> -TimedStorage-GDA-Staging	GB-Hours	Daily	The number of GB-hours that data was stored in S3 Glacier Deep Archive staging storage
<i>region</i> -TimedStorage-GIR-ByteHrs	GB-Hours	Daily	The number of GB-hours that data was stored in S3 Glacier Instant Retrieval storage.

Usage Type	Units	Granularity	Description
<i>region</i> -TimedStorage-GIR-SmObjects	GB-Hours	Daily	The number of GB-hours that small objects (smaller than 128 KB) were stored in S3 Glacier Instant Retrieval storage.
<i>region</i> -TimedStorage-GlacierByteHrs	GB-Hours	Daily	The number of GB-hours that data was stored in S3 Glacier Flexible Retrieval storage
<i>region</i> -TimedStorage-GlacierStaging	GB-Hours	Daily	The number of GB-hours that data was stored in S3 Glacier Flexible Retrieval staging storage
<i>region</i> -TimedStorage-INT-FA-ByteHrs	GB-Hours	Daily	The number of GB-hours that data was stored in the frequent access tier of S3 Intelligent-Tiering storage ⁵
<i>region</i> -TimedStorage-INT-IA-ByteHrs	GB-Hours	Daily	The number of GB-hours that data was stored in the Infrequent Access tier of S3 Intelligent-Tiering storage
<i>region</i> -TimedStorage-INT-AA-ByteHrs	GB-Hours	Daily	The number of GB-hours that data was stored in the Archive Access tier of S3 Intelligent-Tiering storage
<i>region</i> -TimedStorage-INT-AIA-ByteHrs	GB-Hours	Daily	The number of GB-hours that data was stored in the Archive Instant Access tier of S3 Intelligent-Tiering storage
<i>region</i> -TimedStorage-INT-DAA-ByteHrs	GB-Hours	Daily	The number of GB-hours that data was stored in the Deep Archive Access tier of S3 Intelligent-Tiering storage
<i>region</i> -TimedStorage-RRS-ByteHrs	GB-Hours	Daily	The number of GB-hours that data was stored in Reduced Redundancy Storage (RRS) storage
<i>region</i> -TimedStorage-SIA-ByteHrs	GB-Hours	Daily	The number of GB-hours that data was stored in S3 Standard-IA storage
<i>region</i> -TimedStorage-SIA-SmObjects	GB-Hours	Daily	The number of GB-hours that small objects (smaller than 128 KB) were stored in S3 Standard-IA storage ⁴

Usage Type	Units	Granularity	Description
<code>region-TimedStorage-ZIA-ByteHrs</code>	GB-Hours	Daily	The number of GB-hours that data was stored in S3 One Zone-IA storage
<code>region-TimedStorage-ZIA-SmObjects</code>	GB-Hours	Daily	The number of GB-hours that small objects (smaller than 128 KB) were stored in One Zone-IA storage
<code>StorageObjectCount</code>	Count	Daily	The number of objects stored within a given bucket

Notes:

1. If you terminate a transfer before completion, the amount of data that is transferred might exceed the amount of data that your application receives. This discrepancy can occur because a transfer termination request cannot be executed instantaneously, and some amount of data might be in transit pending execution of the termination request. This data in transit is billed as data transferred "out."
2. When objects that are archived to the S3 Glacier Instant Retrieval, S3 Glacier Flexible Retrieval, or S3 Glacier Deep Archive storage class are deleted, overwritten, or transitioned to a different storage class before the minimum storage commitment has passed, which is 90 days for S3 Glacier Instant Retrieval and S3 Glacier Flexible Retrieval, or 180-days for S3 Glacier Deep Archive, there is a prorated charge per gigabyte for the remaining days.
3. For objects that are in S3 Standard-IA or S3 One Zone-IA storage, when they are deleted, overwritten, or transitioned to a different storage class prior to 30 days, there is a prorated charge per gigabyte for the remaining days.
4. For small objects (smaller than 128 KB) that are in S3 Standard-IA or S3 One Zone-IA storage, when they are deleted, overwritten, or transitioned to a different storage class prior to 30 days, there is a prorated charge per gigabyte for the remaining days.
5. There is no minimum billable object size for objects in the S3 Intelligent-Tiering storage class. Objects that are smaller than 128 KB are not monitored or eligible for auto-tiering. Smaller objects are always stored in the S3 Intelligent-Tiering Frequent Access tier.

Tracking Operations in Your Usage Reports

Operations describe the action taken on your AWS object or bucket by the specified usage type. Operations are indicated by self-explanatory codes, such as `PutObject` or `ListBucket`. To see which actions on your bucket generated a specific type of usage, use these codes. When you create a usage report, you can choose to include **All Operations**, or a specific operation, for example, `GetObject`, to report on.

More Info

- [AWS usage report for Amazon S3 \(p. 838\)](#)
- [AWS Billing reports for Amazon S3 \(p. 836\)](#)
- [Amazon S3 Pricing](#)
- [Amazon S3 FAQ](#)
- [S3 Glacier Pricing](#)
- [S3 Glacier FAQs](#)

Filtering and retrieving data using Amazon S3 Select

With Amazon S3 Select, you can use simple structured query language (SQL) statements to filter the contents of an Amazon S3 object and retrieve just the subset of data that you need. By using Amazon S3 Select to filter this data, you can reduce the amount of data that Amazon S3 transfers, which reduces the cost and latency to retrieve this data.

Amazon S3 Select works on objects stored in CSV, JSON, or Apache Parquet format. It also works with objects that are compressed with GZIP or BZIP2 (for CSV and JSON objects only), and server-side encrypted objects. You can specify the format of the results as either CSV or JSON, and you can determine how the records in the result are delimited.

You pass SQL expressions to Amazon S3 in the request. Amazon S3 Select supports a subset of SQL. For more information about the SQL elements that are supported by Amazon S3 Select, see [SQL reference for Amazon S3 Select and S3 Glacier Select \(p. 856\)](#).

You can perform SQL queries using AWS SDKs, the SELECT Object Content REST API, the AWS Command Line Interface (AWS CLI), or the Amazon S3 console. The Amazon S3 console limits the amount of data returned to 40 MB. To retrieve more data, use the AWS CLI or the API.

Requirements and limits

The following are requirements for using Amazon S3 Select:

- You must have `s3:GetObject` permission for the object you are querying.
- If the object you are querying is encrypted with a customer-provided encryption key (SSE-C), you must use `https`, and you must provide the encryption key in the request.

The following limits apply when using Amazon S3 Select:

- The maximum length of a SQL expression is 256 KB.
- The maximum length of a record in the input or result is 1 MB.
- Amazon S3 Select can only emit nested data using the JSON output format.
- You cannot specify the S3 Glacier Flexible Retrieval, S3 Glacier Deep Archive, or REDUCED_REDUNDANCY storage classes. For more information, about storage classes see [Storage Classes](#).

Additional limitations apply when using Amazon S3 Select with Parquet objects:

- Amazon S3 Select supports only columnar compression using GZIP or Snappy. Amazon S3 Select doesn't support whole-object compression for Parquet objects.
- Amazon S3 Select doesn't support Parquet output. You must specify the output format as CSV or JSON.
- The maximum uncompressed row group size is 512 MB.
- You must use the data types specified in the object's schema.
- Selecting on a repeated field returns only the last value.

Constructing a request

When you construct a request, you provide details of the object that is being queried using an `InputSerialization` object. You provide details of how the results are to be returned using an `OutputSerialization` object. You also include the SQL expression that Amazon S3 uses to filter the request.

For more information about constructing an Amazon S3 Select request, see [SELECTObjectContent](#) in the *Amazon Simple Storage Service API Reference*. You can also see one of the SDK code examples in the following sections.

Requests using scan ranges

With Amazon S3 Select, you can scan a subset of an object by specifying a range of bytes to query. This capability lets you parallelize scanning the whole object by splitting the work into separate Amazon S3 Select requests for a series of non-overlapping scan ranges. Scan ranges don't need to be aligned with record boundaries. An Amazon S3 Select scan range request runs across the byte range that you specify. A record that starts within the scan range specified but extends beyond the scan range will be processed by the query. For example; the following shows an Amazon S3 object containing a series of records in a line-delimited CSV format:

```
A,B  
C,D  
D,E  
E,F  
G,H  
I,J
```

Use the Amazon S3 Select `ScanRange` parameter and `Start at (Byte) 1` and `End at (Byte) 4`. So the scan range would start at "" and scan till the end of record starting at "C" and return the result **C, D** because that is the end of the record.

Amazon S3 Select scan range requests support Parquet, CSV (without quoted delimiters), and JSON objects (in LINES mode only). CSV and JSON objects must be uncompressed. For line-based CSV and JSON objects, when a scan range is specified as part of the Amazon S3 Select request, all records that start within the scan range are processed. For Parquet objects, all of the row groups that start within the scan range requested are processed.

Amazon S3 Select scan range requests are available to use on the Amazon S3 CLI, API and SDK. You can use the `ScanRange` parameter in the Amazon S3 Select request for this feature. For more information, see the [Amazon S3 SELECT Object Content](#) in the *Amazon Simple Storage Service API Reference*.

Errors

Amazon S3 Select returns an error code and associated error message when an issue is encountered while attempting to run a query. For a list of error codes and descriptions, see the [List of SELECT Object Content Error Codes](#) section of the *Error Responses* page in the *Amazon Simple Storage Service API Reference*.

For more information about Amazon S3 Select, see the topics below:

Topics

- [Examples of using Amazon S3 Select on objects \(p. 854\)](#)
- [SQL reference for Amazon S3 Select and S3 Glacier Select \(p. 856\)](#)

Examples of using Amazon S3 Select on objects

You can use S3 Select with the Amazon S3 REST API and the AWS SDK to select content from objects.

Using the REST API

You can use the AWS SDK to select content from objects. However, if your application requires it, you can send REST requests directly. For more information about the request and response format, see [SELECT Object Content](#).

Using the AWS SDKs

You can use Amazon S3 Select to select contents of an object using the `selectObjectContent` method, which on success returns the results of the SQL expression.

Java

The following Java code returns the value of the first column for each record that is stored in an object that contains data stored in CSV format. It also requests Progress and Stats messages to be returned. You must provide a valid bucket name and an object that contains data in CSV format.

For instructions on creating and testing a working sample, see [Testing the Amazon S3 Java Code Examples \(p. 1192\)](#).

```
package com.amazonaws;

import com.amazonaws.services.s3.AmazonS3;
import com.amazonaws.services.s3.AmazonS3ClientBuilder;
import com.amazonaws.services.s3.model.CSVInput;
import com.amazonaws.services.s3.model.CSVOutput;
import com.amazonaws.services.s3.model.CompressionType;
import com.amazonaws.services.s3.model.ExpressionType;
import com.amazonaws.services.s3.model.InputSerialization;
import com.amazonaws.services.s3.model.OutputSerialization;
import com.amazonaws.services.s3.model.SelectObjectContentEvent;
import com.amazonaws.services.s3.model.SelectObjectContentEventVisitor;
import com.amazonaws.services.s3.model.SelectObjectContentRequest;
import com.amazonaws.services.s3.model.SelectObjectContentResult;

import java.io.File;
import java.io.FileOutputStream;
import java.io.InputStream;
import java.io.OutputStream;
import java.util.concurrent.atomic.AtomicBoolean;

import static com.amazonaws.util.IOUtils.copy;

/**
 * This example shows how to query data from S3Select and consume the response in the
 * form of an
 * InputStream of records and write it to a file.
 */

public class RecordInputStreamExample {

    private static final String BUCKET_NAME = "${my-s3-bucket}";
    private static final String CSV_OBJECT_KEY = "${my-csv-object-key}";
    private static final String S3_SELECT_RESULTS_PATH = "${my-s3-select-results-path}";
    private static final String QUERY = "select s._1 from S3Object s";
```

```

public static void main(String[] args) throws Exception {
    final AmazonS3 s3Client = AmazonS3ClientBuilder.defaultClient();

    SelectObjectContentRequest request = generateBaseCSVRequest(BUCKET_NAME,
CSV_OBJECT_KEY, QUERY);
    final AtomicBoolean isResultComplete = new AtomicBoolean(false);

    try (OutputStream fileOutputStream = new FileOutputStream(new File
(S3_SELECT_RESULTS_PATH));
         SelectObjectContentResult result = s3Client.selectObjectContent(request))
{
    InputStream resultInputStream = result.getPayload().getRecordsInputStream(
        new SelectObjectContentEventVisitor() {
            @Override
            public void visit(SelectObjectContentEvent.StatsEvent event)
            {
                System.out.println(
                    "Received Stats, Bytes Scanned: " +
                event.getDetails().getBytesScanned()
                    + " Bytes Processed: " +
                event.getDetails().getBytesProcessed());
            }

            /*
             * An End Event informs that the request has finished
             successfully.
             */
            @Override
            public void visit(SelectObjectContentEvent.EndEvent event)
            {
                isResultComplete.set(true);
                System.out.println("Received End Event. Result is
complete.");
            }
        });
    copy(resultInputStream, fileOutputStream);
}

/*
 * The End Event indicates all matching records have been transmitted.
 * If the End Event is not received, the results may be incomplete.
 */
if (!isResultComplete.get()) {
    throw new Exception("S3 Select request was incomplete as End Event was not
received.");
}

private static SelectObjectContentRequest generateBaseCSVRequest(String bucket,
String key, String query) {
    SelectObjectContentRequest request = new SelectObjectContentRequest();
    request.setBucketName(bucket);
    request.setKey(key);
    request.setExpression(query);
    request.setExpressionType(ExpressionType.SQL);

    InputSerialization inputSerialization = new InputSerialization();
    inputSerialization.setCsv(new CSVInput());
    inputSerialization.setCompressionType(CompressionType.NONE);
    request.setInputSerialization(inputSerialization);

    OutputSerialization outputSerialization = new OutputSerialization();
    outputSerialization.setCsv(new CSVOutput());
    request.setOutputSerialization(outputSerialization);
}

```

```
        return request;  
    }  
}
```

JavaScript

For a JavaScript example using the AWS SDK for JavaScript with the S3 SelectObjectContent API to select records from JSON and CSV files stored in Amazon S3, see the blog post [Introducing support for Amazon S3 Select in the AWS SDK for JavaScript](#).

Python

For a Python example on using structured query language (SQL) queries to search through data loaded to Amazon S3 as a comma-separated value (CSV) file using S3 Select, see the blog post [Querying data without servers or databases using Amazon S3 Select](#).

SQL reference for Amazon S3 Select and S3 Glacier Select

This reference contains a description of the structured query language (SQL) elements that are supported by Amazon S3 Select and S3 Glacier Select.

Topics

- [SELECT Command \(p. 856\)](#)
- [Data Types \(p. 862\)](#)
- [Operators \(p. 863\)](#)
- [Reserved Keywords \(p. 865\)](#)
- [SQL Functions \(p. 869\)](#)

SELECT Command

Amazon S3 Select and S3 Glacier Select support only the `SELECT` SQL command. The following ANSI standard clauses are supported for `SELECT`:

- `SELECT` list
- `FROM` clause
- `WHERE` clause
- `LIMIT` clause (Amazon S3 Select only)

Note

Amazon S3 Select and S3 Glacier Select queries currently do not support subqueries or joins.

SELECT List

The `SELECT` list names the columns, functions, and expressions that you want the query to return. The list represents the output of the query.

```
SELECT *  
SELECT projection [ AS column_alias | column_alias ] [, ...]
```

The first form with * (asterisk) returns every row that passed the WHERE clause, as-is. The second form creates a row with user-defined output scalar expressions **projection** for each column.

FROM Clause

Amazon S3 Select and S3 Glacier Select support the following forms of the FROM clause:

```
FROM table_name
FROM table_name alias
FROM table_name AS alias
```

Where table_name is one of S3Object (for Amazon S3 Select) or ARCHIVE or OBJECT (for S3 Glacier Select) referring to the archive being queried over. Users coming from traditional relational databases can think of this as a database schema that contains multiple views over a table.

Following standard SQL, the FROM clause creates rows that are filtered in the WHERE clause and projected in the SELECT list.

For JSON objects that are stored in Amazon S3 Select, you can also use the following forms of the FROM clause:

```
FROM S3Object[*].path
FROM S3Object[*].path alias
FROM S3Object[*].path AS alias
```

Using this form of the FROM clause, you can select from arrays or objects within a JSON object. You can specify path using one of the following forms:

- By name (in an object): .name or ['name']
- By index (in an array): [index]
- By wildcard (in an object): .*
- By wildcard (in an array): [*]

Note

- This form of the FROM clause works only with JSON objects.
- Wildcards always emit at least one record. If no record matches, then Amazon S3 Select emits the value MISSING. During output serialization (after the query is complete), Amazon S3 Select replaces MISSING values with empty records.
- Aggregate functions (AVG, COUNT, MAX, MIN, and SUM) skip MISSING values.
- If you don't provide an alias when using a wildcard, you can refer to the row using the last element in the path. For example, you could select all prices from a list of books using the query `SELECT price FROM S3Object[*].books[*].price`. If the path ends in a wildcard rather than a name, then you can use the value _1 to refer to the row. For example, instead of `SELECT price FROM S3Object[*].books[*].price`, you could use the query `SELECT _1.price FROM S3Object[*].books[*]`.
- Amazon S3 Select always treats a JSON document as an array of root-level values. Thus, even if the JSON object that you are querying has only one root element, the FROM clause must begin with S3Object[*]. However, for compatibility reasons, Amazon S3 Select allows you to omit the wildcard if you don't include a path. Thus, the complete clause FROM S3Object is equivalent to FROM S3Object[*] as S3Object. If you include a path, you must also use the wildcard. So FROM S3Object and FROM S3Object[*].path are both valid clauses, but FROM S3Object.path is not.

Example

Examples:

Example #1

This example shows results using the following dataset and query:

```
{ "Rules": [ {"id": "1"}, {"expr": "y > x"}, {"id": "2", "expr": "z = DEBUG"} ]}  
{ "created": "June 27", "modified": "July 6" }
```

```
SELECT id FROM S3Object[*].Rules[*].id
```

```
{"id":"1"}  
{}  
{"id":"2"}  
{}
```

Amazon S3 Select produces each result for the following reasons:

- `{"id":"id-1"}` — `S3Object[0].Rules[0].id` produced a match.
- `{}` — `S3Object[0].Rules[1].id` did not match a record, so Amazon S3 Select emitted `MISSING`, which was then changed to an empty record during output serialization and returned.
- `{"id":"id-2"}` — `S3Object[0].Rules[2].id` produced a match.
- `{}` — `S3Object[1]` did not match on `Rules`, so Amazon S3 Select emitted `MISSING`, which was then changed to an empty record during output serialization and returned.

If you don't want Amazon S3 Select to return empty records when it doesn't find a match, you can test for the value `MISSING`. The following query returns the same results as the previous query, but with the empty values omitted:

```
SELECT id FROM S3Object[*].Rules[*].id WHERE id IS NOT MISSING
```

```
{"id":"1"}  
{"id":"2"}
```

Example #2

This example shows results using the following dataset and queries:

```
{ "created": "936864000", "dir_name": "important_docs", "files": [ { "name": "." },  
{ "name": ".." }, { "name": ".aws" }, { "name": "downloads" } ], "owner": "AWS S3" }  
{ "created": "936864000", "dir_name": "other_docs", "files": [ { "name": "." }, { "name": ".." }, { "name": "my stuff" }, { "name": "backup" } ], "owner": "User" }
```

```
SELECT d.dir_name, d.files FROM S3Object[*] d
```

```
{"dir_name":"important_docs","files":[{"name":"."}, {"name":".."}, {"name": ".aws"},  
 {"name": "downloads"}]}  
{"dir_name":"other_docs","files":[{"name":"."}, {"name":".."}, {"name": "my stuff"},  
 {"name": "backup"}]}
```

```
SELECT _1.dir_name, _1.owner FROM S3Object[*]
```

```
{"dir_name": "important_docs", "owner": "AWS S3"}  
{"dir_name": "other_docs", "owner": "User"}
```

WHERE Clause

The `WHERE` clause follows this syntax:

```
WHERE condition
```

The `WHERE` clause filters rows based on the *condition*. A condition is an expression that has a Boolean result. Only rows for which the condition evaluates to `TRUE` are returned in the result.

LIMIT Clause (Amazon S3 Select only)

The `LIMIT` clause follows this syntax:

```
LIMIT number
```

The `LIMIT` clause limits the number of records that you want the query to return based on *number*.

Note

S3 Glacier Select does not support the `LIMIT` clause.

Attribute Access

The `SELECT` and `WHERE` clauses can refer to record data using one of the methods in the following sections, depending on whether the file that is being queried is in CSV or JSON format.

CSV

- **Column Numbers** – You can refer to the *Nth* column of a row with the column name `_N`, where *N* is the column position. The position count starts at 1. For example, the first column is named `_1` and the second column is named `_2`.

You can refer to a column as `_N` or `alias._N`. For example, `_2` and `myAlias._2` are both valid ways to refer to a column in the `SELECT` list and `WHERE` clause.

- **Column Headers** – For objects in CSV format that have a header row, the headers are available to the `SELECT` list and `WHERE` clause. In particular, as in traditional SQL, within `SELECT` and `WHERE` clause expressions, you can refer to the columns by `alias.column_name` or `column_name`.

JSON (Amazon S3 Select only)

- **Document** – You can access JSON document fields as `alias.name`. Nested fields can also be accessed; for example, `alias.name1.name2.name3`.
- **List** – You can access elements in a JSON list using zero-based indexes with the `[]` operator. For example, you can access the second element of a list as `alias[1]`. Accessing list elements can be combined with fields as `alias.name1.name2[1].name3`.
- **Examples:** Consider this JSON object as a sample dataset:

```
{"name": "Susan Smith",  
 "org": "engineering",  
 "projects":
```

```
[  
    {"project_name": "project1", "completed": false},  
    {"project_name": "project2", "completed": true}  
]
```

Example #1

The following query returns these results:

```
Select s.name from S3Object s
```

```
{"name": "Susan Smith"}
```

Example #2

The following query returns these results:

```
Select s.projects[0].project_name from S3Object s
```

```
{"project_name": "project1"}
```

Case Sensitivity of Header/Attribute Names

With Amazon S3 Select and S3 Glacier Select, you can use double quotation marks to indicate that column headers (for CSV objects) and attributes (for JSON objects) are case sensitive. Without double quotation marks, object headers/attributes are case insensitive. An error is thrown in cases of ambiguity.

The following examples are either 1) Amazon S3 or S3 Glacier objects in CSV format with the specified column header(s), and with `FileHeaderInfo` set to "Use" for the query request; or 2) Amazon S3 objects in JSON format with the specified attributes.

Example #1: The object being queried has header/attribute "NAME".

- The following expression successfully returns values from the object (no quotation marks: case insensitive):

```
SELECT s.name from S3Object s
```

- The following expression results in a 400 error `MissingHeaderName` (quotation marks: case sensitive):

```
SELECT s."name" from S3Object s
```

Example #2: The Amazon S3 object being queried has one header/attribute with "NAME" and another header/attribute with "name".

- The following expression results in a 400 error `AmbiguousFieldName` (no quotation marks: case insensitive, but there are two matches):

```
SELECT s.name from S3Object s
```

- The following expression successfully returns values from the object (quotation marks: case sensitive, so it resolves the ambiguity).

```
SELECT s."NAME" from S3Object s
```

Using Reserved Keywords as User-Defined Terms

Amazon S3 Select and S3 Glacier Select have a set of reserved keywords that are needed to run the SQL expressions used to query object content. Reserved keywords include function names, data types, operators, and so on. In some cases, user-defined terms like the column headers (for CSV files) or attributes (for JSON object) may clash with a reserved keyword. When this happens, you must use double quotation marks to indicate that you are intentionally using a user-defined term that clashes with a reserved keyword. Otherwise a 400 parse error will result.

For the full list of reserved keywords see [Reserved Keywords \(p. 865\)](#).

The following example is either 1) an Amazon S3 or S3 Glacier object in CSV format with the specified column headers, with `FileHeaderInfo` set to "Use" for the query request, or 2) an Amazon S3 object in JSON format with the specified attributes.

Example: The object being queried has header/attribute named "CAST", which is a reserved keyword.

- The following expression successfully returns values from the object (quotation marks: use user-defined header/attribute):

```
SELECT s."CAST" from S3Object s
```

- The following expression results in a 400 parse error (no quotation marks: clash with reserved keyword):

```
SELECT s.CAST from S3Object s
```

Scalar Expressions

Within the `WHERE` clause and the `SELECT` list, you can have SQL *scalar expressions*, which are expressions that return scalar values. They have the following form:

- ***literal***

An SQL literal.

- ***column_reference***

A reference to a column in the form `column_name` or `alias.column_name`.

- ***unary_op expression***

Where `unary_op` unary is an SQL unary operator.

- ***expression binary_op expression***

Where `binary_op` is an SQL binary operator.

- ***func_name***

Where `func_name` is the name of a scalar function to invoke.

- ***expression [NOT] BETWEEN expression AND expression***

- ***expression LIKE expression [ESCAPE expression]***

Data Types

Amazon S3 Select and S3 Glacier Select support several primitive data types.

Data Type Conversions

The general rule is to follow the `CAST` function if defined. If `CAST` is not defined, then all input data is treated as a string. It must be cast into the relevant data types when necessary.

For more information about the `CAST` function, see [CAST \(p. 872\)](#).

Supported Data Types

Amazon S3 Select and S3 Glacier Select support the following set of primitive data types.

Name	Description	Examples
bool	TRUE or FALSE	FALSE
int, integer	8-byte signed integer in the range -9,223,372,036,854,775,808 to 9,223,372,036,854,775,807.	100000
string	UTF8-encoded variable-length string. The default limit is one character. The maximum character limit is 2,147,483,647.	'xyz'
float	8-byte floating point number.	<code>CAST(0.456 AS FLOAT)</code>
decimal, numeric	Base-10 number, with maximum precision of 38 (that is, the maximum number of significant digits), and with scale within the range of -2^{31} to $2^{31}-1$ (that is, the base-10 exponent).	123.456
	<p>Note Amazon S3 Select ignores scale and precision when you provide both at the same time.</p>	
timestamp	<p>Time stamps represent a specific moment in time, always include a local offset, and are capable of arbitrary precision.</p> <p>In the text format, time stamps follow the W3C note on date and time formats, but they must end with the literal "T" if not at least whole-day precision. Fractional seconds are allowed, with at least one digit of precision, and an unlimited maximum. Local-time offsets can be represented as either hour:minute offsets from UTC, or as the literal "Z" to denote a local time of UTC. They are required on time stamps with time and are not allowed on date values.</p>	<code>CAST('2007-04-05T14:30Z' AS TIMESTAMP)</code>

Supported Parquet types

Amazon S3 Select supports the following Parquet types.

- DATE
- DECIMAL
- ENUM
- INT(8)
- INT(16)

- INT(32)
- INT(64)
- LIST

Note

For LIST Parquet type output, Amazon S3 Select only supports JSON format. However, if the query limits the data to simple values, the LIST Parquet type can also be queried in CSV format.

- STRING
- TIMESTAMP supported precision (MILLIS/MICROS/NANOS)

Note

Timestamps saved as an INT(96) are unsupported.

Due to the range of the INT(64) type, timestamps using the NANOS unit can only represent values between 1677-09-21 00:12:43 and 2262-04-11 23:47:16. Values outside of this range cannot be represented with the NANOS unit.

Mapping of Parquet types to supported data types in Amazon S3 Select

Parquet types	Supported data types
DATE	timestamp
DECIMAL	decimal, numeric
ENUM	string
INT(8)	int, integer
INT(16)	int, integer
INT(32)	int, integer
INT(64)	decimal, numeric
LIST	Each Parquet type in list is mapped to the corresponding data type
STRING	string
TIMESTAMP	timestamp

Operators

Amazon S3 Select and S3 Glacier Select support the following operators.

Logical Operators

- AND
- NOT
- OR

Comparison Operators

- <

- >
- <=
- >=
- =
- <>
- !=
- BETWEEN
- IN – For example: IN ('a', 'b', 'c')

Pattern Matching Operators

- LIKE
- _ (Matches any character)
- % (Matches any sequence of characters)

Unitary Operators

- IS NULL
- IS NOT NULL

Math Operators

Addition, subtraction, multiplication, division, and modulo are supported.

- +
- -
- *
- /
- %

Operator Precedence

The following table shows the operators' precedence in decreasing order.

Operator/ Element	Associativity	Required
-	right	unary minus
*, /, %	left	multiplication, division, modulo
+, -	left	addition, subtraction
IN		set membership
BETWEEN		range containment

Operator/ Element	Associativity	Required
LIKE		string pattern matching
<>		less than, greater than
=	right	equality, assignment
NOT	right	logical negation
AND	left	logical conjunction
OR	left	logical disjunction

Reserved Keywords

Below is the list of reserved keywords for Amazon S3 Select and S3 Glacier Select. These include function names, data types, operators, etc., that needed to run the SQL expressions used to query object content.

```
absolute
action
add
all
allocate
alter
and
any
are
as
asc
assertion
at
authorization
avg
bag
begin
between
bit
bit_length
blob
bool
boolean
both
by
cascade
cascaded
case
cast
catalog
char
char_length
character
character_length
check
clob
```

```
close
coalesce
collate
collation
column
commit
connect
connection
constraint
constraints
continue
convert
corresponding
count
create
cross
current
current_date
current_time
current_timestamp
current_user
cursor
date
day
deallocate
dec
decimal
declare
default
deferrable
deferred
delete
desc
describe
descriptor
diagnostics
disconnect
distinct
domain
double
drop
else
end
end-exec
escape
except
exception
exec
execute
exists
external
extract
false
fetch
first
float
for
foreign
found
from
full
get
global
go
goto
grant
```

```
group
having
hour
identity
immediate
in
indicator
initially
inner
input
insensitive
insert
int
integer
intersect
interval
into
is
isolation
join
key
language
last
leading
left
level
like
limit
list
local
lower
match
max
min
minute
missing
module
month
names
national
natural
nchar
next
no
not
null
nullif
numeric
octet_length
of
on
only
open
option
or
order
outer
output
overlaps
pad
partial
pivot
position
precision
prepare
preserve
```

```
primary
prior
privileges
procedure
public
read
real
references
relative
restrict
revoke
right
rollback
rows
schema
scroll
second
section
select
session
session_user
set
sexp
size
smallint
some
space
sql
sqlcode
sqlerror
sqlstate
string
struct
substring
sum
symbol
system_user
table
temporary
then
time
timestamp
timezone_hour
timezone_minute
to
trailing
transaction
translate
translation
trim
true
tuple
union
unique
unknown
unpivot
update
upper
usage
user
using
value
values
varchar
varying
view
```

```
when
whenever
where
with
work
write
year
zone
```

SQL Functions

Amazon S3 Select and S3 Glacier Select support several SQL functions.

Topics

- [Aggregate Functions \(Amazon S3 Select only\) \(p. 869\)](#)
- [Conditional Functions \(p. 870\)](#)
- [Conversion Functions \(p. 872\)](#)
- [Date Functions \(p. 873\)](#)
- [String Functions \(p. 878\)](#)

Aggregate Functions (Amazon S3 Select only)

Amazon S3 Select supports the following aggregate functions.

Note

S3 Glacier Select does not support aggregate functions.

Function	Argument Type	Return Type
AVG(expression)	INT, FLOAT, DECIMAL	DECIMAL for an INT argument, FLOAT for a floating-point argument; otherwise the same as the argument data type.
COUNT	-	INT
MAX(expression)	INT, DECIMAL	Same as the argument type.
MIN(expression)	INT, DECIMAL	Same as the argument type.
SUM(expression)	INT, FLOAT, DOUBLE, DECIMAL	INT for INT argument, FLOAT for a floating-point argument; otherwise, the same as the argument data type.

Conditional Functions

Amazon S3 Select and S3 Glacier Select support the following conditional functions.

Topics

- [CASE \(p. 870\)](#)
- [COALESCE \(p. 871\)](#)
- [NULLIF \(p. 871\)](#)

CASE

The CASE expression is a conditional expression, similar to if/then/else statements found in other languages. CASE is used to specify a result when there are multiple conditions. There are two types of CASE expressions: simple and searched.

In simple CASE expressions, an expression is compared with a value. When a match is found, the specified action in the THEN clause is applied. If no match is found, the action in the ELSE clause is applied.

In searched CASE expressions, each CASE is evaluated based on a Boolean expression, and the CASE statement returns the first matching CASE. If no matching CASEs are found among the WHEN clauses, the action in the ELSE clause is returned.

Syntax

Note

Currently, Amazon S3 Select doesn't support ORDER BY or queries that contain new lines. Make sure that you use queries with no line breaks.

Simple CASE statement used to match conditions:

```
CASE expression WHEN value THEN result [WHEN...] [ELSE result] END
```

Searched CASE statement used to evaluate each condition:

```
CASE WHEN boolean condition THEN result [WHEN ...] [ELSE result] END
```

Examples

Note

If you use the Amazon S3 console to run the following examples and your CSV file contains a header row, please select **Exclude the first line of CSV data**.

Example 1: Use a simple CASE expression to replace New York City with Big Apple in a query. Replace all other city names with other.

```
SELECT venuecity, CASE venuecity WHEN 'New York City' THEN 'Big Apple' ELSE 'other' END
FROM S3Object;
```

Query Result:

venuecity	case
Los Angeles	other

```
New York City | Big Apple
San Francisco | other
Baltimore | other
...
```

Example 2: Use a searched CASE expression to assign group numbers based on the PRICEPAID value for individual ticket sales:

```
SELECT pricepaid, CASE WHEN CAST(pricepaid as FLOAT) < 10000 THEN 'group 1' WHEN
CAST(pricepaid as FLOAT) > 10000 THEN 'group 2' ELSE 'group 3' END FROM S3Object;
```

Query Result:

```
pricepaid | case
-----+-----
12624.00 | group 2
10000.00 | group 3
10000.00 | group 3
9996.00 | group 1
9988.00 | group 1
...
```

COALESCE

Evaluates the arguments in order and returns the first non-unknown, that is, the first non-null or non-missing. This function does not propagate null and missing.

Syntax

```
COALESCE ( expression, expression, ... )
```

Parameters

expression

The target expression that the function operates on.

Examples

```
COALESCE(1)          -- 1
COALESCE(null)       -- null
COALESCE(null, null) -- null
COALESCE(missing)    -- null
COALESCE(missing, missing) -- null
COALESCE(1, null)    -- 1
COALESCE(null, null, 1) -- 1
COALESCE(null, 'string') -- 'string'
COALESCE(missing, 1)  -- 1
```

NULLIF

Given two expressions, returns NULL if the two expressions evaluate to the same value; otherwise, returns the result of evaluating the first expression.

Syntax

```
NULLIF ( expression1, expression2 )
```

Parameters

expression1, expression2

The target expressions that the function operates on.

Examples

```
NULLIF(1, 1)          -- null
NULLIF(1, 2)          -- 1
NULLIF(1.0, 1)        -- null
NULLIF(1, '1')        -- 1
NULLIF([1], [1])      -- null
NULLIF(1, NULL)       -- 1
NULLIF(NULL, 1)       -- null
NULLIF(null, null)    -- null
NULLIF(missing, null) -- null
NULLIF(missing, missing) -- null
```

Conversion Functions

Amazon S3 Select and S3 Glacier Select support the following conversion functions.

Topics

- [CAST \(p. 872\)](#)

CAST

The `CAST` function converts an entity, such as an expression that evaluates to a single value, from one type to another.

Syntax

```
CAST ( expression AS data_type )
```

Parameters

expression

A combination of one or more values, operators, and SQL functions that evaluate to a value.

data_type

The target data type, such as `INT`, to cast the expression to. For a list of supported data types, see [Data Types \(p. 862\)](#).

Examples

```
CAST('2007-04-05T14:30Z' AS TIMESTAMP)
CAST(0.456 AS FLOAT)
```

Date Functions

Amazon S3 Select and S3 Glacier Select support the following date functions.

Topics

- [DATE_ADD \(p. 873\)](#)
- [DATE_DIFF \(p. 874\)](#)
- [EXTRACT \(p. 874\)](#)
- [TO_STRING \(p. 875\)](#)
- [TO_TIMESTAMP \(p. 877\)](#)
- [UTCNOW \(p. 878\)](#)

DATE_ADD

Given a date part, a quantity, and a time stamp, returns an updated time stamp by altering the date part by the quantity.

Syntax

```
DATE_ADD( date_part, quantity, timestamp )
```

Parameters

date_part

Specifies which part of the date to modify. This can be one of the following:

- year
- month
- day
- hour
- minute
- second

quantity

The value to apply to the updated time stamp. Positive values for quantity add to the time stamp's date_part, and negative values subtract.

timestamp

The target time stamp that the function operates on.

Examples

```
DATE_ADD(year, 5, `2010-01-01T`)
-- 2015-01-01 (equivalent to 2015-01-01T)
DATE_ADD(month, 1, `2010T`)
-- 2010-02T (result will add precision as
-- necessary)
DATE_ADD(month, 13, `2010T`)
-- 2011-02T
DATE_ADD(day, -1, `2017-01-10T`)
-- 2017-01-09 (equivalent to 2017-01-09T)
DATE_ADD(hour, 1, `2017T`)
-- 2017-01-01T01:00-00:00
DATE_ADD(hour, 1, `2017-01-02T03:04Z`)
-- 2017-01-02T04:04Z
DATE_ADD(minute, 1, `2017-01-02T03:04:05.006Z`)
-- 2017-01-02T03:05:05.006Z
DATE_ADD(second, 1, `2017-01-02T03:04:05.006Z`)
-- 2017-01-02T03:04:06.006Z
```

DATE_DIFF

Given a date part and two valid time stamps, returns the difference in date parts. The return value is a negative integer when the `date_part` value of `timestamp1` is greater than the `date_part` value of `timestamp2`. The return value is a positive integer when the `date_part` value of `timestamp1` is less than the `date_part` value of `timestamp2`.

Syntax

```
DATE_DIFF( date_part, timestamp1, timestamp2 )
```

Parameters

date_part

Specifies which part of the time stamps to compare. For the definition of `date_part`, see [DATE_ADD \(p. 873\)](#).

timestamp1

The first time stamp to compare.

timestamp2

The second time stamp to compare.

Examples

```
DATE_DIFF(year, `2010-01-01T`, `2011-01-01T`) -- 1
DATE_DIFF(year, `2010T`, `2010-05T`)
  2010-01-01T00:00:00.000Z) -- 4 (2010T is equivalent to
DATE_DIFF(month, `2010T`, `2011T`) -- 12
DATE_DIFF(month, `2011T`, `2010T`) -- -12
DATE_DIFF(day, `2010-01-01T23:00`, `2010-01-02T01:00`) -- 0 (need to be at least 24h apart
  to be 1 day apart)
```

EXTRACT

Given a date part and a time stamp, returns the time stamp's date part value.

Syntax

```
EXTRACT( date_part FROM timestamp )
```

Parameters

date_part

Specifies which part of the time stamps to extract. This can be one of the following:

- year
- month
- day
- hour
- minute
- second
- timezone_hour
- timezone_minute

timestamp

The target time stamp that the function operates on.

Examples

```

EXTRACT(YEAR FROM `2010-01-01T`)
EXTRACT(MONTH FROM `2010T`)
  2010-01-01T00:00:00.000Z)          -- 2010
                                         -- 1 (equivalent to
EXTRACT(MONTH FROM `2010-10T`)
EXTRACT(HOUR FROM `2017-01-02T03:04:05+07:08`)
EXTRACT(MINUTE FROM `2017-01-02T03:04:05+07:08`)
EXTRACT(TIMEZONE_HOUR FROM `2017-01-02T03:04:05+07:08`)
EXTRACT(TIMEZONE_MINUTE FROM `2017-01-02T03:04:05+07:08`) -- 10
                                         -- 3
                                         -- 4
                                         -- 7
                                         -- 8

```

TO_STRING

Given a time stamp and a format pattern, returns a string representation of the time stamp in the given format.

Syntax

```
TO_STRING ( timestamp time_format_pattern )
```

Parameters

timestamp

The target time stamp that the function operates on.

time_format_pattern

A string that has the following special character interpretations.

Format	Example	Description
yy	69	2-digit year
y	1969	4-digit year
yyyy	1969	Zero-padded 4-digit year
M	1	Month of year
MM	01	Zero-padded month of year
MMM	Jan	Abbreviated month year name
MMMM	January	Full month of year name
MMMMM	J	Month of year first letter (NOTE: not valid for use with

Format	Example	Description
		to_timestamp function)
d	2	Day of month (1-31)
dd	02	Zero-padded day of month (01-31)
a	AM	AM or PM of day
h	3	Hour of day (1-12)
hh	03	Zero-padded hour of day (01-12)
H	3	Hour of day (0-23)
HH	03	Zero-padded hour of day (00-23)
m	4	Minute of hour (0-59)
mm	04	Zero-padded minute of hour (00-59)
s	5	Second of minute (0-59)
ss	05	Zero-padded second of minute (00-59)
S	0	Fraction of second (precision: 0.1, range: 0.0-0.9)
SS	6	Fraction of second (precision: 0.01, range: 0.0-0.99)
SSS	60	Fraction of second (precision: 0.001, range: 0.0-0.999)
...

Format	Example	Description
SSSSSSSS	60000000	Fraction of second (maximum precision: 1 nanosecond, range: 0.0-0.999999999)
n	60000000	Nano of second
X	+07 or Z	Offset in hours or "Z" if the offset is 0
XX or XXXX	+0700 or Z	Offset in hours and minutes or "Z" if the offset is 0
XXX or XXXXX	+07:00 or Z	Offset in hours and minutes or "Z" if the offset is 0
x	7	Offset in hours
xx or xxxx	700	Offset in hours and minutes
xxx or xxxxx	+07:00	Offset in hours and minutes

Examples

```

TO_STRING(`1969-07-20T20:18Z`, 'MMMM d, y')                                -- "July 20, 1969"
TO_STRING(`1969-07-20T20:18Z`, 'MMM d, yyyy')                            -- "Jul 20, 1969"
TO_STRING(`1969-07-20T20:18Z`, 'M-d-yy')                                 -- "7-20-69"
TO_STRING(`1969-07-20T20:18Z`, 'MM-d-y')                                -- "07-20-1969"
TO_STRING(`1969-07-20T20:18Z`, 'MMMM d, y h:m a')                         -- "July 20, 1969 8:18 PM"
TO_STRING(`1969-07-20T20:18Z`, 'y-MM-dd''T''H:m:ssX')                      -- "1969-07-20T20:18:00Z"
TO_STRING(`1969-07-20T20:18+08:00Z`, 'y-MM-dd''T''H:m:ssX')                  -- "1969-07-20T20:18:00Z"
TO_STRING(`1969-07-20T20:18+08:00`, 'y-MM-dd''T''H:m:ssXXXX')                --
"1969-07-20T20:18:00+0800"
TO_STRING(`1969-07-20T20:18+08:00`, 'y-MM-dd''T''H:m:ssXXXX')                --
"1969-07-20T20:18:00+08:00"

```

TO_TIMESTAMP

Given a string, converts it to a time stamp. This is the inverse operation of TO_STRING.

Syntax

```
TO_TIMESTAMP ( string )
```

Parameters

string

The target string that the function operates on.

Examples

```
TO_TIMESTAMP('2007T')          -- `2007T`  
TO_TIMESTAMP('2007-02-23T12:14:33.079-08:00') -- `2007-02-23T12:14:33.079-08:00`
```

UTCNOW

Returns the current time in UTC as a time stamp.

Syntax

```
UTCNOW()
```

Parameters

none

Examples

```
UTCNOW() -- 2017-10-13T16:02:11.123Z
```

String Functions

Amazon S3 Select and S3 Glacier Select support the following string functions.

Topics

- [CHAR_LENGTH, CHARACTER_LENGTH \(p. 878\)](#)
- [LOWER \(p. 879\)](#)
- [SUBSTRING \(p. 879\)](#)
- [TRIM \(p. 880\)](#)
- [UPPER \(p. 880\)](#)

CHAR_LENGTH, CHARACTER_LENGTH

Counts the number of characters in the specified string.

Note

`CHAR_LENGTH` and `CHARACTER_LENGTH` are synonyms.

Syntax

```
CHAR_LENGTH ( string )
```

Parameters

string

The target string that the function operates on.

Examples

```
CHAR_LENGTH('')          -- 0
CHAR_LENGTH('abcdefg')   -- 7
```

LOWER

Given a string, converts all uppercase characters to lowercase characters. Any non-uppercased characters remain unchanged.

Syntax

```
LOWER ( string )
```

Parameters

string

The target string that the function operates on.

Examples

```
LOWER('AbCdEfG!@#$') -- 'abcdefg!@#$'
```

SUBSTRING

Given a string, a start index, and optionally a length, returns the substring from the start index up to the end of the string, or up to the length provided.

Note

The first character of the input string has index 1.

- If *start* is < 1, with no length specified then it is set to 1.
- If *start* is < 1, with length specified then it is set to *start* + length -1.
- If *start* + length -1 < 0 then an empty string is returned.
- If *start* + length -1 > = 0 then the sub-string starting at index 1 with length *start* + length - 1 is returned.

Syntax

```
SUBSTRING( string FROM start [ FOR length ] )
```

Parameters

string

The target string that the function operates on.

start

The start position of the string.

length

The length of the substring to return. If not present, proceed to the end of the string.

Examples

```
SUBSTRING("123456789", 0)      -- "123456789"  
SUBSTRING("123456789", 1)      -- "123456789"  
SUBSTRING("123456789", 2)      -- "23456789"  
SUBSTRING("123456789", -4)     -- "123456789"  
SUBSTRING("123456789", 0, 999)  -- "123456789"  
SUBSTRING("123456789", 1, 5)    -- "12345"
```

TRIM

Trims leading or trailing characters from a string. The default character to remove is ''.

Syntax

```
TRIM ( [[LEADING | TRAILING | BOTH remove_chars] FROM] string )
```

Parameters

string

The target string that the function operates on.

LEADING | TRAILING | BOTH

Whether to trim leading or trailing characters, or both leading and trailing characters.

remove_chars

The set of characters to remove. Note that *remove_chars* can be a string with length > 1. This function returns the string with any character from *remove_chars* found at the beginning or end of the string that was removed.

Examples

```
TRIM('      foobar      ')          -- 'foobar'  
TRIM('      \tfoobar\t      ')       -- '\tfoobar\t'  
TRIM(LEADING FROM '      foobar      ') -- 'foobar      '  
TRIM(TRAILING FROM '      foobar      ') -- '      foobar'  
TRIM(BOTH FROM '      foobar      ')  -- 'foobar'  
TRIM(BOTH '12' FROM '1112211foobar22211122') -- 'foobar'
```

UPPER

Given a string, converts all lowercase characters to uppercase characters. Any non-lowercased characters remain unchanged.

Syntax

```
UPPER ( string )
```

Parameters

string

The target string that the function operates on.

Examples

```
UPPER( 'AbCdEfG!@#$' ) -- 'ABCDEFG!@#$'
```

Performing large-scale batch operations on Amazon S3 objects

You can use S3 Batch Operations to perform large-scale batch operations on Amazon S3 objects. S3 Batch Operations can perform a single operation on lists of Amazon S3 objects that you specify. A single job can perform a specified operation on billions of objects containing exabytes of data. Amazon S3 tracks progress, sends notifications, and stores a detailed completion report of all actions, providing a fully managed, auditable, and serverless experience. You can use S3 Batch Operations through the AWS Management Console, AWS CLI, Amazon SDKs, or REST API.

Use S3 Batch Operations to copy objects and set object tags or access control lists (ACLs). You can also initiate object restores from S3 Glacier Flexible Retrieval or invoke an AWS Lambda function to perform custom actions using your objects. You can perform these operations on a custom list of objects, or you can use an Amazon S3 Inventory report to easily generate lists of objects. Amazon S3 Batch Operations use the same Amazon S3 APIs that you already use with Amazon S3, so you'll find the interface familiar.

S3 Batch Operations basics

You can use S3 Batch Operations to perform large-scale batch operations on Amazon S3 objects. S3 Batch Operations can run a single operation or action on lists of Amazon S3 objects that you specify.

Terminology

This section uses the terms *jobs*, *operations*, and *tasks*, which are defined as follows:

Job

A job is the basic unit of work for S3 Batch Operations. A job contains all of the information necessary to run the specified operation on the objects listed in the manifest. After you provide this information and request that the job begin, the job performs the operation for each object in the manifest.

Operation

The operation is the type of API [action](#), such as copying objects, that you want the Batch Operations job to run. Each job performs a single type of operation across all objects that are specified in the manifest.

Task

A task is the unit of execution for a job. A task represents a single call to an Amazon S3 or AWS Lambda API operation to perform the job's operation on a single object. Over the course of a job's lifetime, S3 Batch Operations create one task for each object specified in the manifest.

How an S3 Batch Operations job works

A job is the basic unit of work for S3 Batch Operations. A job contains all of the information necessary to run the specified operation on a list of objects. To create a job, you give S3 Batch Operations a list of objects and specify the action to perform on those objects.

For information about the operations that S3 Batch Operations supports, see [Operations supported by S3 Batch Operations \(p. 895\)](#).

A batch job performs a specified operation on every object that is included in its *manifest*. A manifest lists the objects that you want a batch job to process and it is stored as an object in a bucket. You can use a comma-separated values (CSV)-formatted [Amazon S3 Inventory \(p. 739\)](#) report as a manifest, which makes it easy to create large lists of objects located in a bucket. You can also specify a manifest in a simple CSV format that enables you to perform batch operations on a customized list of objects contained within a single bucket.

After you create a job, Amazon S3 processes the list of objects in the manifest and runs the specified operation against each object. While a job is running, you can monitor its progress programmatically or through the Amazon S3 console. You can also configure a job to generate a completion report when it finishes. The completion report describes the results of each task that was performed by the job. For more information about monitoring jobs, see [Managing S3 Batch Operations jobs \(p. 920\)](#).

Granting permissions for Amazon S3 Batch Operations

Before creating and running S3 Batch Operations jobs, you must grant required permissions. To create an Amazon S3 Batch Operations job, the `s3:CreateJob` user permission is required. The same entity that creates the job must also have the `iam:PassRole` permission to pass the AWS Identity and Access Management (IAM) role that is specified for the job to Batch Operations.

For general information about specifying IAM resources, see [IAM JSON policy, Resource elements](#) in the *IAM User Guide*. The following sections provide information about creating an IAM role and attaching policies.

Topics

- [Creating an S3 Batch Operations IAM role \(p. 882\)](#)
- [Attaching permissions policies \(p. 883\)](#)

Creating an S3 Batch Operations IAM role

Amazon S3 must have permissions to perform S3 Batch Operations on your behalf. You grant these permissions through an AWS Identity and Access Management (IAM) role. This section provides examples of the trust and permissions policies you use when creating an IAM role. For more information, see [IAM roles](#) in the *IAM User Guide*. For examples, see [Controlling permissions for S3 Batch Operations using job tags \(p. 937\)](#) and [Copying objects using S3 Batch Operations \(p. 896\)](#).

In your IAM policies, you can also use condition keys to filter access permissions for S3 Batch Operations jobs. For more information and a complete list of Amazon S3 specific condition keys, see [Actions, resources, and condition keys for Amazon S3 \(p. 431\)](#).

Trust policy

To allow the S3 Batch Operations service principal to assume the IAM role, attach the following trust policy to the role.

```
{  
    "Version": "2012-10-17",  
    "Statement": [  
        {  
            "Effect": "Allow",  
            "Principal": {  
                "Service": "batchoperations.s3.amazonaws.com"  
            }  
        }  
    ]  
}
```

```
        },
        "Action": "sts:AssumeRole"
    ]
}
```

Attaching permissions policies

Depending on the type of operations, you can attach one of the following policies.

Before you configure permissions, note the following:

- Regardless of the operation, Amazon S3 needs permissions to read your manifest object from your S3 bucket and optionally write a report to your bucket. Therefore, all of the following policies include these permissions.
- For Amazon S3 Inventory report manifests, S3 Batch Operations requires permission to read the manifest.json object and all associated CSV data files.
- Version-specific permissions such as s3:GetObjectVersion are only required when you are specifying the version ID of the objects.
- If you are running S3 Batch Operations on encrypted objects, the IAM role must also have access to the AWS KMS keys used to encrypt them.

Copy objects: PutObject

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Action": [
                "s3:PutObject",
                "s3:PutObjectAcl",
                "s3:PutObjectTagging"
            ],
            "Effect": "Allow",
            "Resource": "arn:aws:s3:::DestinationBucket/*"
        },
        {
            "Action": [
                "s3:GetObject",
                "s3:GetObjectAcl",
                "s3:GetObjectTagging",
                "s3>ListBucket"
            ],
            "Effect": "Allow",
            "Resource": [
                "arn:aws:s3:::SourceBucket",
                "arn:aws:s3:::SourceBucket/*"
            ]
        },
        {
            "Effect": "Allow",
            "Action": [
                "s3:GetObject",
                "s3:GetObjectVersion"
            ],
            "Resource": [
                "arn:aws:s3:::ManifestBucket/*"
            ]
        }
    ]
}
```

```
        "Effect": "Allow",
        "Action": [
            "s3:PutObject"
        ],
        "Resource": [
            "arn:aws:s3:::ReportBucket/*"
        ]
    }
]
```

Replace object tagging: PutObjectTagging

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "s3:PutObjectTagging",
                "s3:PutObjectVersionTagging"
            ],
            "Resource": "arn:aws:s3:::TargetResource/*"
        },
        {
            "Effect": "Allow",
            "Action": [
                "s3:GetObject",
                "s3:GetObjectVersion"
            ],
            "Resource": [
                "arn:aws:s3:::ManifestBucket/*"
            ]
        },
        {
            "Effect": "Allow",
            "Action": [
                "s3:PutObject"
            ],
            "Resource": [
                "arn:aws:s3:::ReportBucket/*"
            ]
        }
    ]
}
```

Delete object tagging: DeleteObjectTagging

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "s3>DeleteObjectTagging",
                "s3>DeleteObjectVersionTagging"
            ],
            "Resource": [
                "arn:aws:s3:::TargetResource/*"
            ]
        },
        {
            "Effect": "Allow",
            "
```

```
        "Action": [
            "s3:GetObject",
            "s3:GetObjectVersion"
        ],
        "Resource": [
            "arn:aws:s3:::ManifestBucket/*"
        ]
    },
    {
        "Effect": "Allow",
        "Action": [
            "s3:PutObject"
        ],
        "Resource": [
            "arn:aws:s3:::ReportBucket/*"
        ]
    }
]
```

Replace access control list: PutObjectAcl

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "s3:PutObjectAcl",
                "s3:PutObjectVersionAcl"
            ],
            "Resource": "arn:aws:s3:::TargetResource/*"
        },
        {
            "Effect": "Allow",
            "Action": [
                "s3:GetObject",
                "s3:GetObjectVersion"
            ],
            "Resource": [
                "arn:aws:s3:::ManifestBucket/*"
            ]
        },
        {
            "Effect": "Allow",
            "Action": [
                "s3:PutObject"
            ],
            "Resource": [
                "arn:aws:s3:::ReportBucket/*"
            ]
        }
    ]
}
```

Restore objects: RestoreObject

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [

```

```
        "s3:RestoreObject"
    ],
    "Resource": "arn:aws:s3:::TargetResource/*"
},
{
    "Effect": "Allow",
    "Action": [
        "s3:GetObject",
        "s3:GetObjectVersion"
    ],
    "Resource": [
        "arn:aws:s3:::ManifestBucket/*"
    ]
},
{
    "Effect": "Allow",
    "Action": [
        "s3:PutObject"
    ],
    "Resource": [
        "arn:aws:s3:::ReportBucket/*"
    ]
}
]
```

Apply Object Lock retention: PutObjectRetention

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": "s3:GetBucketObjectLockConfiguration",
            "Resource": [
                "arn:aws:s3:::TargetResource"
            ]
        },
        {
            "Effect": "Allow",
            "Action": [
                "s3:PutObjectRetention",
                "s3:BypassGovernanceRetention"
            ],
            "Resource": [
                "arn:aws:s3:::TargetResource/*"
            ]
        },
        {
            "Effect": "Allow",
            "Action": [
                "s3:GetObject",
                "s3:GetObjectVersion"
            ],
            "Resource": [
                "arn:aws:s3:::ManifestBucket/*"
            ]
        },
        {
            "Effect": "Allow",
            "Action": [
                "s3:PutObject"
            ],
            "Resource": [

```

```

        "arn:aws:s3:::ReportBucket/*"
    ]
}
]
```

Apply Object Lock legal hold: PutObjectLegalHold

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": "s3:GetBucketObjectLockConfiguration",
            "Resource": [
                "arn:aws:s3:::TargetResource"
            ]
        },
        {
            "Effect": "Allow",
            "Action": "s3:PutObjectLegalHold",
            "Resource": [
                "arn:aws:s3:::TargetResource/*"
            ]
        },
        {
            "Effect": "Allow",
            "Action": [
                "s3:GetObject",
                "s3:GetObjectVersion"
            ],
            "Resource": [
                "arn:aws:s3:::ManifestBucket/*"
            ]
        },
        {
            "Effect": "Allow",
            "Action": [
                "s3:PutObject"
            ],
            "Resource": [
                "arn:aws:s3:::ReportBucket/*"
            ]
        }
    ]
}
```

Replicate existing objects: InitiateReplication with a S3 generated manifest

Use this policy if using and storing a S3 generated manifest. For more information about using Batch Operations for replicate existing objects see, [Replicating existing objects with S3 Batch Replication \(p. 798\)](#).

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Action": [
                "s3:InitiateReplication"
            ],
            "Effect": "Allow",
            "Resource": [

```

```

        "arn:aws:s3:::*** replication source bucket ***/*"
    ],
},
{
    "Action":[
        "s3:GetReplicationConfiguration",
        "s3:PutInventoryConfiguration"
    ],
    "Effect":"Allow",
    "Resource":[
        "arn:aws:s3:::*** replication source bucket ***"
    ]
},
{
    "Action":[
        "s3:GetObject",
        "s3:GetObjectVersion"
    ],
    "Effect":"Allow",
    "Resource":[
        "arn:aws:s3:::*** manifest bucket ***/*"
    ]
},
{
    "Effect":"Allow",
    "Action":[
        "s3:PutObject"
    ],
    "Resource":[
        "arn:aws:s3:::*** completion report bucket ****/*",
        "arn:aws:s3:::*** manifest bucket ****/*"
    ]
}
]
}
}

```

Replicate existing objects: InitiateReplication with a user manifest

Use this policy if using a user supplied manifest. For more information about using Batch Operations for replicate existing objects see, [Replicating existing objects with S3 Batch Replication \(p. 798\)](#).

```

{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Action":[
                "s3:InitiateReplication"
            ],
            "Effect":"Allow",
            "Resource":[
                "arn:aws:s3:::*** replication source bucket ***/*"
            ]
        },
        {
            "Action":[
                "s3:GetObject",
                "s3:GetObjectVersion"
            ],
            "Effect":"Allow",
            "Resource":[
                "arn:aws:s3:::*** manifest bucket ***/*"
            ]
        }
    ]
}

```

```
    "Effect": "Allow",
    "Action": [
        "s3:PutObject"
    ],
    "Resource": [
        "arn:aws:s3:::*** completion report bucket ***/*"
    ]
}
}
```

Creating an S3 Batch Operations job

With S3 Batch Operations, you can perform large-scale batch operations on a list of specific Amazon S3 objects. This section describes the information that you need to create an S3 Batch Operations job and the results of a `Create Job` request. It also provides instructions for creating a Batch Operations job using the AWS Management Console, AWS Command Line Interface (AWS CLI), and AWS SDK for Java.

When you create an S3 Batch Operations job, you can request a completion report for all tasks or only for failed tasks. As long as at least one task has been invoked successfully, S3 Batch Operations generates a report for jobs that have completed, failed, or been canceled. For more information, see [Examples: S3 Batch Operations completion reports \(p. 930\)](#).

Topics

- [Batch Operations job request elements \(p. 889\)](#)
- [Specifying a manifest \(p. 890\)](#)
- [Creating a job \(p. 891\)](#)
- [Job responses \(p. 895\)](#)

Batch Operations job request elements

To create an S3 Batch Operations job, you must provide the following information:

Operation

Specify the operation that you want S3 Batch Operations to run against the objects in the manifest. Each operation type accepts parameters that are specific to that operation. This enables you to perform the same tasks as if you performed the operation one-by-one on each object.

Manifest

The manifest is a list of all of the objects that you want S3 Batch Operations to run the specified action on. You can use a CSV-formatted [Amazon S3 Inventory \(p. 739\)](#) report as a manifest or use your own customized CSV list of objects.

If the objects in your manifest are in a versioned bucket, you must specify the version IDs for the objects. For more information, see [Specifying a manifest \(p. 890\)](#).

Priority

Use job priorities to indicate the relative priority of this job to others running in your account. A higher number indicates higher priority.

Job priorities only have meaning relative to the priorities that are set for other jobs in the same account and Region. You can choose whatever numbering system works for you. For example, you might want to assign all `Initiate Restore Object` jobs a priority of 1, all `PUT Object Copy` jobs a priority of 2, and all `Put Object ACL` jobs a priority of 3.

S3 Batch Operations prioritize jobs according to priority numbers, but strict ordering isn't guaranteed. Therefore, you shouldn't use job priorities to ensure that any one job starts or finishes before any other job. If you need to ensure strict ordering, wait until one job has finished before starting the next.

RoleArn

Specify an AWS Identity and Access Management (IAM) role to run the job. The IAM role that you use must have sufficient permissions to perform the operation that is specified in the job. For example, to run a `PUT Object Copy` job, the IAM role must have `s3:GetObject` permissions for the source bucket and `s3:PutObject` permissions for the destination bucket. The role also needs permissions to read the manifest and write the job-completion report.

For more information about IAM roles, see [IAM Roles](#) in the *IAM User Guide*.

For more information about Amazon S3 permissions, see [Amazon S3 actions \(p. 415\)](#).

Report

Specify whether you want S3 Batch Operations to generate a completion report. If you request a job-completion report, you must also provide the parameters for the report in this element. The necessary information includes:

- The bucket where you want to store the report
- The format of the report
- Whether you want the report to include the details of all tasks or only failed tasks
- An optional prefix string

Tags (optional)

You can label and control access to your S3 Batch Operations jobs by adding *tags*. Tags can be used to identify who is responsible for a Batch Operations job. You can create jobs with tags attached to them, and you can add tags to jobs after you create them. For example, you could grant an IAM user permission to invoke `CreateJob` provided that the job is created with the tag "Department=Finance".

For more information, see [the section called "Using tags" \(p. 932\)](#).

Description (optional)

To track and monitor your job, you can also provide a description of up to 256 characters. Amazon S3 includes this description whenever it returns information about a job or displays job details on the Amazon S3 console. You can then easily sort and filter jobs according to the descriptions that you assigned. Descriptions don't need to be unique, so you can use descriptions as categories (for example, "Weekly Log Copy Jobs") to help you track groups of similar jobs.

Specifying a manifest

A manifest is an Amazon S3 object that contains object keys that you want Amazon S3 to act upon. To create a manifest for a job, you specify the manifest object key, ETag, and optional version ID. The contents of the manifest must be URL encoded. Manifests that use server-side encryption with customer-provided keys (SSE-C) are not supported. Manifests that use server-side encryption with AWS Key Management Service (SSE-KMS) AWS KMS keys are only supported when using CSV-formatted inventory reports. Your manifest must contain the bucket name, object key, and optionally, the object version for each object. Any other fields in the manifest are not used by S3 Batch Operations.

You can specify a manifest in a create job request using one of the following two formats.

- Amazon S3 Inventory report — Must be a CSV-formatted Amazon S3 Inventory report. You must specify the `manifest.json` file that is associated with the inventory report. For more information

about inventory reports, see [Amazon S3 Inventory \(p. 739\)](#). If the inventory report includes version IDs, S3 Batch Operations operates on the specific object versions.

Note

S3 Batch Operations supports CSV *inventory reports* that are AWS KMS-encrypted.

- CSV file — Each row in the file must include the bucket name, object key, and optionally, the object version. Object keys must be URL-encoded, as shown in the following examples. The manifest must either include version IDs for all objects or omit version IDs for all objects. For more information about the CSV manifest format, see [JobManifestSpec](#) in the *Amazon Simple Storage Service API Reference*.

Note

S3 Batch Operations does not support CSV *manifest files* that are AWS KMS-encrypted.

The following is an example manifest in CSV format without version IDs.

```
Examplebucket,objectkey1
Examplebucket,objectkey2
Examplebucket,objectkey3
Examplebucket,photos/jpgs/objectkey4
Examplebucket,photos/jpgs/newjersey/objectkey5
Examplebucket,object%20key%20with%20spaces
```

The following is an example manifest in CSV format including version IDs.

```
Examplebucket,objectkey1,PZ9ibn9D5lP6p298B7S9_ceqx1n5EJ0p
Examplebucket,objectkey2,YY_ouuAJByNW1LRBffMfxMge7XQWxMBF
Examplebucket,objectkey3,jbo9_jhdPEyB4RrmOxWSokU0EoNrU_OI
Examplebucket,photos/jpgs/objectkey4,6EqlikJJxLTsHsnbZbSRffn24_eh5Ny4
Examplebucket,photos/jpgs/newjersey/objectkey5,imHf3FAiRsvBW_EHB8GOu.NHunHO1gVs
Examplebucket,object%20key%20with%20spaces,9HkPvDaZY5MVbMhn6TMn1YTb5ArQAO3w
```

Important

When using a user supplied manifest and a versioned bucket, we recommend that you specify the version IDs for the objects. When you create a job, S3 Batch Operations parses the entire manifest before running the job. However, it doesn't take a "snapshot" of the state of the bucket.

Because manifests can contain billions of objects, jobs might take a long time to run. If you overwrite an object with a new version while a job is running and you didn't specify a version ID for that object, Amazon S3 performs the operation on the latest version of the object, not on the version that existed when you created the job. The only way to avoid this behavior is to specify version IDs for the objects that are listed in the manifest.

Note

Amazon S3 gives you the option to create a manifest for the S3 Batch Replication job. Batch Replication is an on-demand operation that replicates existing objects. For more information about Batch Replication, see [Replicating existing objects with S3 Batch Replication \(p. 798\)](#).

Creating a job

You can create S3 Batch Operations jobs using the AWS Management Console, AWS CLI, Amazon SDKs, or REST API.

For more information about creating a job request, see [Batch Operations job request elements \(p. 889\)](#).

Prerequisite

Before you create a Batch Operations job, confirm that you have configured relevant permissions. For more information, see [Granting permissions for Amazon S3 Batch Operations \(p. 882\)](#).

Using the S3 console

To create a batch job

1. Sign in to the AWS Management Console and open the Amazon S3 console at <https://console.aws.amazon.com/s3/>.
2. Choose **Batch Operations** on the navigation pane of the Amazon S3 console.
3. Choose **Create job**.
4. Choose the **Region** where you want to create your job.
5. Under **Manifest format**, choose the type of manifest object to use.
 - If you choose **S3 inventory report**, enter the path to the manifest.json object that Amazon S3 generated as part of the CSV-formatted Inventory report, and optionally the version ID for the manifest object if you want to use a version other than the most recent.
 - If you choose **CSV**, enter the path to a CSV-formatted manifest object. The manifest object must follow the format described in the console. You can optionally include the version ID for the manifest object if you want to use a version other than the most recent.
6. Choose **Next**.
7. Under **Operation**, choose the operation that you want to perform on all objects listed in the manifest. Fill out the information for the operation you chose and then choose **Next**.
8. Fill out the information for **Configure additional options** and then choose **Next**.
9. For **Review**, verify the settings. If you need to make changes, choose **Previous**. Otherwise, choose **Create Job**.

Using the AWS CLI

The following example creates an S3 Batch Operations **S3PutObjectTagging** job using the AWS CLI.

To create a Batch Operations **S3PutObjectTagging** job

1. Create an AWS Identity and Access Management (IAM) role, and assign permissions. This role grants Amazon S3 permission to add object tags, for which you create a job in the next step.
 - a. Create an IAM role as follows.

```
aws iam create-role \
--role-name S3BatchJobRole \
--assume-role-policy-document '{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Principal": {
                "Service": "batchoperations.s3.amazonaws.com"
            },
            "Action": "sts:AssumeRole"
        }
    ]
}'
```

Record the role's Amazon Resource Name (ARN). You need the ARN when you create a job.

- b. Create an IAM policy with permissions, and attach it to the IAM role that you created in the previous step. For more information about permissions, see [Granting permissions for Amazon S3 Batch Operations \(p. 882\)](#).

```
aws iam put-role-policy \
```

```
--role-name S3BatchJobRole \
--policy-name PutObjectTaggingBatchJobPolicy \
--policy-document '{
"Version":"2012-10-17",
"Statement": [
  {
    "Effect": "Allow",
    "Action": [
      "s3:PutObjectTagging",
      "s3:PutObjectVersionTagging"
    ],
    "Resource": "arn:aws:s3:::{${TargetResource}}/*"
  },
  {
    "Effect": "Allow",
    "Action": [
      "s3:GetObject",
      "s3:GetObjectVersion",
      "s3:GetBucketLocation"
    ],
    "Resource": [
      "arn:aws:s3:::{${ManifestBucket}}",
      "arn:aws:s3:::{${ManifestBucket}}/*"
    ]
  },
  {
    "Effect": "Allow",
    "Action": [
      "s3:PutObject",
      "s3:GetBucketLocation"
    ],
    "Resource": [
      "arn:aws:s3:::{${ReportBucket}}",
      "arn:aws:s3:::{${ReportBucket}}/*"
    ]
  }
]
}'
```

2. Create an S3PutObjectTagging job.

The `manifest.csv` file provides a list of bucket and object key values. The job applies the specified tags to objects identified in the manifest. The ETag is the ETag of the `manifest.csv` object, which you can get from the Amazon S3 console. The request specifies the `no-confirmation-required` parameter. Therefore, Amazon S3 makes the job eligible for execution without you having to confirm it using the `update-job-status` command.

```
aws s3control create-job \
  --region us-west-2 \
  --account-id acct-id \
  --operation '{"S3PutObjectTagging": { "TagSet": [{"Key": "keyOne", "Value": "ValueOne"}] }}' \
  --manifest '{"Spec":{"Format": "S3BatchOperations_CSV_20180820", "Fields": ["Bucket", "Key"]}, "Location": {"ObjectArn": "arn:aws:s3:::my_manifests/manifest.csv", "ETag": "60e460c9d1046e73f7dde5043ac3ae85"} }' \
  --report '{"Bucket": "arn:aws:s3:::bucket-where-completion-report-goes", "Prefix": "final-reports", "Format": "Report_CSV_20180820", "Enabled": true, "ReportScope": "AllTasks"}' \
  --priority 42 \
  --role-arn IAM-role \
  --client-request-token $(uuidgen) \
  --description "job Description" \
  --no-confirmation-required
```

In response, Amazon S3 returns a job ID (for example, 00e123a4-c0d8-41f4-a0eb-b46f9ba5b07c). You need the ID in the next commands.

Using the AWS SDK for Java

The following example creates an S3 Batch Operations job using the AWS SDK for Java.

Example

```
package aws.example.s3control;

import com.amazonaws.AmazonServiceException;
import com.amazonaws.SdkClientException;
import com.amazonaws.auth.profile.ProfileCredentialsProvider;
import com.amazonaws.services.s3control.AWSS3Control;
import com.amazonaws.services.s3control.AWSS3ControlClient;
import com.amazonaws.services.s3control.model.*;

import java.util.UUID;
import java.util.ArrayList;

import static com.amazonaws.regions.Regions.US_WEST_2;

public class CreateJob {
    public static void main(String[] args) {
        String accountId = "Account ID";
        String iamRoleArn = "IAM Role ARN";
        String reportBucketName = "arn:aws:s3:::bucket-where-completion-report-goes";
        String uuid = UUID.randomUUID().toString();

        ArrayList<S3Tag> tagSet = new ArrayList<S3Tag>();
        tagSet.add(new S3Tag().withKey("keyOne").withValue("ValueOne"));

        try {
            JobOperation jobOperation = new JobOperation()
                .withS3PutObjectTagging(new S3SetObjectTaggingOperation()
                    .withTagSet(tagSet)
                );
            JobManifest manifest = new JobManifest()
                .withSpec(new JobManifestSpec()
                    .withFormat("S3BatchOperations_CSV_20180820")
                    .withFields(new String[]{
                        "Bucket", "Key"
                    })
                )
                .withLocation(new JobManifestLocation()
                    .withObjectArn("arn:aws:s3:::my_manifests/manifest.csv")
                    .withETag("60e460c9d1046e73f7dde5043ac3ae85"));
            JobReport jobReport = new JobReport()
                .withBucket(reportBucketName)
                .withPrefix("reports")
                .withFormat("Report_CSV_20180820")
                .withEnabled(true)
                .withReportScope("AllTasks");

            AWSS3Control s3ControlClient = AWSS3ControlClient.builder()
                .withCredentials(new ProfileCredentialsProvider())
                .withRegion(US_WEST_2)
                .build();
        }
    }
}
```

```
s3ControlClient.createJob(new CreateJobRequest()
    .withAccountId(accountId)
    .withOperation(jobOperation)
    .withManifest(manifest)
    .withReport(jobReport)
    .withPriority(42)
    .withRoleArn(iamRoleArn)
    .withClientRequestToken(uuid)
    .withDescription("job description")
    .withConfirmationRequired(false)
);

} catch (AmazonServiceException e) {
    // The call was transmitted successfully, but Amazon S3 couldn't process
    // it and returned an error response.
    e.printStackTrace();
} catch (SdkClientException e) {
    // Amazon S3 couldn't be contacted for a response, or the client
    // couldn't parse the response from Amazon S3.
    e.printStackTrace();
}
}
```

Using the REST API

You can use the REST API to create a Batch Operations job. For more information, see [CreateJob REST API](#) in the *Amazon Simple Storage Service API Reference*.

Job responses

If the `Create Job` request succeeds, Amazon S3 returns a job ID. The job ID is a unique identifier that Amazon S3 generates automatically so that you can identify your Batch Operations job and monitor its status.

When you create a job through the AWS CLI, Amazon SDKs, or REST API, you can set S3 Batch Operations to begin processing the job automatically. The job runs as soon as it's ready and not waiting behind higher-priority jobs.

When you create a job through the AWS Management Console, you must review the job details and confirm that you want to run it before Batch Operations can begin to process it. After you confirm that you want to run the job, it progresses as though you created it through one of the other methods. If a job remains in the suspended state for over 30 days, it will fail.

Operations supported by S3 Batch Operations

S3 Batch Operations supports several different operations. The topics in this section describe each of these operations.

Copy objects

The **Copy** operation copies each object that is specified in the manifest. You can copy objects to a bucket in the same AWS Region or to a bucket in a different Region. S3 Batch Operations supports most options available through Amazon S3 for copying objects. These options include setting object metadata, setting permissions, and changing an object's storage class.

You can also use the Copy operation to copy existing unencrypted objects and write them back to the same bucket as encrypted objects. For more information, see [Encrypting objects with Amazon S3 Batch Operations](#).

When you copy objects, you can change the checksum algorithm used to calculate the checksum of the object. If objects don't have an additional checksum calculated, you can also add one by specifying the checksum algorithm for Amazon S3 to use. For more information, see [Checking object integrity \(p. 215\)](#).

For more information about copying objects in Amazon S3 and the required and optional parameters, see [Copying objects \(p. 201\)](#) in this guide and [CopyObject](#) in the *Amazon Simple Storage Service API Reference*.

Restrictions and limitations

- All source objects must be in one bucket.
- All destination objects must be in one bucket.
- You must have read permissions for the source bucket and write permissions for the destination bucket.
- Objects to be copied can be up to 5 GB in size.
- If you try to copy objects from the S3 Glacier Flexible Retrieval or S3 Glacier Deep Archive classes to the S3 Standard storage class, you need to first restore these objects. For more information, see [Restoring an archived object \(p. 673\)](#).
- Copy jobs must be created in the destination Region, which is the Region you intend to copy the objects to.
- All Copy options are supported except for conditional checks on ETags and server-side encryption with customer-provided encryption keys (SSE-C).
- If the buckets are unversioned, you will overwrite objects with the same key names.
- Objects are not necessarily copied in the same order as they appear in the manifest. For versioned buckets, if preserving current/non-current version order is important, you should copy all noncurrent versions first. Then, after the first job is complete, copy the current versions in a subsequent job.
- Copying objects to the Reduced Redundancy Storage (RRS) class is not supported.

Copying objects using S3 Batch Operations

You can use S3 Batch Operations to create a PUT copy job to copy objects within the same account or to a different destination account. The following sections contain examples of how to store and use a manifest that is in a different account. In the first section, you can use Amazon S3 Inventory to deliver the inventory report to the destination account for use during job creation or, you can use a comma-separated values (CSV) manifest in the source or destination account, as shown in the second example. The third example shows how to use the Copy operation to activate S3 Bucket Key encryption on existing objects.

Copy Operation Examples

- [Using an inventory report delivered to the destination account to copy objects across AWS accounts \(p. 896\)](#)
- [Using a CSV manifest stored in the source account to copy objects across AWS accounts \(p. 899\)](#)
- [Using S3 Batch Operations to encrypt objects with S3 Bucket Keys \(p. 901\)](#)

Using an inventory report delivered to the destination account to copy objects across AWS accounts

You can use Amazon S3 Inventory to create an inventory report and use the report to create a list of objects to copy with S3 Batch Operations. For more information about using a CSV manifest in the source or destination account, see [the section called "Using a CSV manifest to copy objects across AWS accounts" \(p. 899\)](#).

Amazon S3 Inventory generates inventories of the objects in a bucket. The resulting list is published to an output file. The bucket that is inventoried is called the *source bucket*, and the bucket where the inventory report file is stored is called the *destination bucket*.

The Amazon S3 Inventory report can be configured to be delivered to another AWS account. This allows S3 Batch Operations to read the inventory report when the job is created in the destination account.

For more information about Amazon S3 Inventory source and destination buckets, see [Source and destination buckets \(p. 739\)](#).

The easiest way to set up an inventory is by using the AWS Management Console, but you can also use the REST API, AWS Command Line Interface (AWS CLI), or AWS SDKs.

The following console procedure contains the high-level steps for setting up permissions for an S3 Batch Operations job. In this procedure, you copy objects from a source account to a destination account, with the inventory report stored in the destination account.

To set up Amazon S3 Inventory for source and destination buckets owned by different accounts

1. Sign in to the AWS Management Console and open the Amazon S3 console at <https://console.aws.amazon.com/s3/>.
2. Choose a destination bucket to store the inventory report in.

Decide on a destination manifest bucket for storing the inventory report. In this procedure, the *destination account* is the account that owns both the destination manifest bucket and the bucket that the objects are copied to.

3. Configure an inventory to list the objects in a source bucket and publish the list to the destination manifest bucket.

Configure an inventory list for a source bucket. When you do this, you specify the destination bucket where you want the list to be stored. The inventory report for the source bucket is published to the destination bucket. In this procedure, the *source account* is the account that owns the source bucket.

For information about how to use the console to configure an inventory, see [Configuring Amazon S3 Inventory \(p. 741\)](#).

Choose **CSV** for the output format.

When you enter information for the destination bucket, choose **Buckets in another account**. Then enter the name of the destination manifest bucket. Optionally, you can enter the account ID of the destination account.

After the inventory configuration is saved, the console displays a message similar to the following:

Amazon S3 could not create a bucket policy on the destination bucket. Ask the destination bucket owner to add the following bucket policy to allow Amazon S3 to place data in that bucket.

The console then displays a bucket policy that you can use for the destination bucket.

4. Copy the destination bucket policy that appears on the console.
5. In the destination account, add the copied bucket policy to the destination manifest bucket where the inventory report is stored.
6. Create a role in the destination account that is based on the S3 Batch Operations trust policy. For more information about the trust policy, see [Trust policy \(p. 882\)](#).

For more information about creating a role, see [Creating a role to delegate permissions to an AWS service](#) in the *IAM User Guide*.

Enter a name for the role (the example role uses the name `BatchOperationsDestinationRoleCOPY`). Choose the **S3** service, and then choose the **S3 bucket Batch Operations** use case, which applies the trust policy to the role.

Then choose **Create policy** to attach the following policy to the role.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "AllowBatchOperationsDestinationObjectCOPY",
            "Effect": "Allow",
            "Action": [
                "s3:PutObject",
                "s3:PutObjectVersionAcl",
                "s3:PutObjectAcl",
                "s3:PutObjectVersionTagging",
                "s3:PutObjectTagging",
                "s3:GetObject",
                "s3:GetObjectVersion",
                "s3:GetObjectAcl",
                "s3:GetObjectTagging",
                "s3:GetObjectVersionAcl",
                "s3:GetObjectVersionTagging"
            ],
            "Resource": [
                "arn:aws:s3:::ObjectDestinationBucket/*",
                "arn:aws:s3:::ObjectSourceBucket/*",
                "arn:aws:s3:::ObjectDestinationManifestBucket/*"
            ]
        }
    ]
}
```

The role uses the policy to grant `batchoperations.s3.amazonaws.com` permission to read the manifest in the destination bucket. It also grants permissions to GET objects, access control lists (ACLs), tags, and versions in the source object bucket. And it grants permissions to PUT objects, ACLs, tags, and versions into the destination object bucket.

7. In the source account, create a bucket policy for the source bucket that grants the role that you created in the previous step to GET objects, ACLs, tags, and versions in the source bucket. This step allows S3 Batch Operations to get objects from the source bucket through the trusted role.

The following is an example of the bucket policy for the source account.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "AllowBatchOperationsSourceObjectCOPY",
            "Effect": "Allow",
            "Principal": {
                "AWS": "arn:aws:iam::DestinationAccountNumber:role/BatchOperationsDestinationRoleCOPY"
            },
            "Action": [
                "s3:GetObject",
                "s3:GetObjectVersion",
                "s3:GetObjectAcl",
                "s3:GetObjectTagging",
                "s3:GetObjectVersionAcl",
                "s3:GetObjectVersionTagging"
            ]
        }
    ]
}
```

```
        ],
        "Resource": "arn:aws:s3:::ObjectSourceBucket/*"
    }
}
```

8. After the inventory report is available, create an S3 Batch Operations PUT object copy job in the destination account, choosing the inventory report from the destination manifest bucket. You need the ARN for the role that you created in the destination account.

For general information about creating a job, see [Creating an S3 Batch Operations job \(p. 889\)](#).

For information about creating a job using the console, see [Creating an S3 Batch Operations job \(p. 889\)](#).

Using a CSV manifest stored in the source account to copy objects across AWS accounts

You can use a CSV file that is stored in a different AWS account as a manifest for an S3 Batch Operations job. For using an S3 Inventory Report, see the section called [“Using an inventory report to copy objects across AWS accounts” \(p. 896\)](#).

The following procedure shows how to set up permissions when using an S3 Batch Operations job to copy objects from a source account to a destination account with the CSV manifest file stored in the source account.

To set up a CSV manifest stored in a different AWS account

1. Create a role in the destination account that is based on the S3 Batch Operations trust policy. In this procedure, the *destination account* is the account that the objects are being copied to.

For more information about the trust policy, see [Trust policy \(p. 882\)](#).

For more information about creating a role, see [Creating a role to delegate permissions to an AWS service](#) in the *IAM User Guide*.

If you create the role using the console, enter a name for the role (the example role uses the name `BatchOperationsDestinationRoleCOPY`). Choose the **S3** service, and then choose the **S3 bucket Batch Operations** use case, which applies the trust policy to the role.

Then choose **Create policy** to attach the following policy to the role.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "AllowBatchOperationsDestinationObjectCOPY",
            "Effect": "Allow",
            "Action": [
                "s3:PutObject",
                "s3:PutObjectVersionAcl",
                "s3:PutObjectAcl",
                "s3:PutObjectVersionTagging",
                "s3:PutObjectTagging",
                "s3:GetObject",
                "s3:GetObjectVersion",
                "s3:GetObjectAcl",
                "s3:GetObjectTagging",
                "s3:GetObjectVersionAcl",
                "s3:GetObjectVersionTagging"
            ],
            "Resource": [
                "arn:aws:s3:::ObjectSourceBucket/*"
            ]
        }
    ]
}
```

```

        "arn:aws:s3:::ObjectDestinationBucket/*",
        "arn:aws:s3:::ObjectSourceBucket/*",
        "arn:aws:s3:::ObjectSourceManifestBucket/*"
    ]
}
]
}
}

```

Using the policy, the role grants `batchoperations.s3.amazonaws.com` permission to read the manifest in the source manifest bucket. It grants permissions to GET objects, ACLs, tags, and versions in the source object bucket. It also grants permissions to PUT objects, ACLs, tags, and versions into the destination object bucket.

2. In the source account, create a bucket policy for the bucket that contains the manifest to grant the role that you created in the previous step to GET objects and versions in the source manifest bucket.

This step allows S3 Batch Operations to read the manifest using the trusted role. Apply the bucket policy to the bucket that contains the manifest.

The following is an example of the bucket policy to apply to the source manifest bucket.

```

{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "AllowBatchOperationsSourceManifestRead",
            "Effect": "Allow",
            "Principal": {
                "AWS": [
                    "arn:aws:iam::DestinationAccountNumber:user/ConsoleUserCreatingJob",
                    "arn:aws:iam::DestinationAccountNumber:role/
BatchOperationsDestinationRoleCOPY"
                ]
            },
            "Action": [
                "s3:GetObject",
                "s3:GetObjectVersion"
            ],
            "Resource": "arn:aws:s3:::ObjectSourceManifestBucket/*"
        }
    ]
}

```

This policy also grants permissions to allow a console user who is creating a job in the destination account the same permissions in the source manifest bucket through the same bucket policy.

3. In the source account, create a bucket policy for the source bucket that grants the role you created to GET objects, ACLs, tags, and versions in the source object bucket. S3 Batch Operations can then get objects from the source bucket through the trusted role.

The following is an example of the bucket policy for the bucket that contains the source objects.

```

{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "AllowBatchOperationsSourceObjectCOPY",
            "Effect": "Allow",
            "Principal": {
                "AWS": "arn:aws:iam::DestinationAccountNumber:role/
BatchOperationsDestinationRoleCOPY"
            },
            "Action": [

```

```
        "s3:GetObject",
        "s3:GetObjectVersion",
        "s3:GetObjectAcl",
        "s3:GetObjectTagging",
        "s3:GetObjectVersionAcl",
        "s3:GetObjectVersionTagging"
    ],
    "Resource": "arn:aws:s3:::ObjectSourceBucket/*"
}
]
}
```

4. Create an S3 Batch Operations job in the destination account. You need the Amazon Resource Name (ARN) for the role that you created in the destination account.

For general information about creating a job, see [Creating an S3 Batch Operations job \(p. 889\)](#).

For information about creating a job using the console, see [Creating an S3 Batch Operations job \(p. 889\)](#).

Using S3 Batch Operations to encrypt objects with S3 Bucket Keys

In this section, you use the Amazon S3 Batch Operations Copy operation to identify and activate S3 Bucket Keys encryption on existing objects. For more information about S3 Bucket Keys, see [Reducing the cost of SSE-KMS with Amazon S3 Bucket Keys \(p. 347\)](#) and [Configuring your bucket to use an S3 Bucket Key with SSE-KMS for new objects \(p. 350\)](#).

Topics covered in this example include the following:

Topics

- [Prerequisites \(p. 901\)](#)
- [Step 1: Get your list of objects using Amazon S3 Inventory \(p. 901\)](#)
- [Step 2: Filter your object list with S3 Select \(p. 902\)](#)
- [Step 3: Set up and run your S3 Batch Operations job \(p. 904\)](#)
- [Summary \(p. 907\)](#)

Prerequisites

To follow along with the steps in this procedure, you need an AWS account and at least one S3 bucket to hold your working files and encrypted results. You might also find much of the existing S3 Batch Operations documentation useful, including the following topics:

- [S3 Batch Operations basics \(p. 881\)](#)
- [Creating an S3 Batch Operations job \(p. 889\)](#)
- [Operations supported by S3 Batch Operations \(p. 895\)](#)
- [Managing S3 Batch Operations jobs \(p. 920\)](#)

Step 1: Get your list of objects using Amazon S3 Inventory

To get started, identify the S3 bucket that contains the objects to encrypt, and get a list of its contents. An Amazon S3 Inventory report is the most convenient and affordable way to do this. The report provides the list of the objects in a bucket along with associated metadata. The **source bucket** refers to the inventoried bucket, and the **destination bucket** refers to the bucket where you store the inventory report file. For more information about Amazon S3 Inventory source and destination buckets, see [Amazon S3 Inventory \(p. 739\)](#).

The easiest way to set up an inventory is by using the AWS Management Console. But you can also use the REST API, AWS Command Line Interface (AWS CLI), or AWS SDKs. Before following these steps, be sure to sign in to the console and open the Amazon S3 console at <https://console.aws.amazon.com/s3/>. If you encounter permission denied errors, add a bucket policy to your destination bucket. For more information, see [Granting permissions for Amazon S3 Inventory and Amazon S3 analytics \(p. 497\)](#).

To get a list of objects using S3 Inventory

1. Open the Amazon S3 console at <https://console.aws.amazon.com/s3/>.
2. In the navigation pane, choose **Buckets**, and choose a bucket that contains objects to encrypt.
3. On the **Management** tab, navigate to the **Inventory configurations** section, and choose **Create inventory configuration**.
4. Give your new inventory a name, enter the name of the destination S3 bucket, and optionally create a destination prefix for Amazon S3 to assign objects in that bucket.
5. For **Output format**, choose **CSV**.
6. In the **Additional fields - optional** section, choose **Encryption** and any other report fields that interest you. Set the frequency for report deliveries to **Daily** so that the first report is delivered to your bucket sooner.
7. Choose **Create** to save your configuration.

Amazon S3 can take up to 48 hours to deliver the first report, so check back when the first report arrives. After you receive your first report, proceed to the next section to filter your S3 Inventory report's contents. If you no longer want to receive inventory reports for this bucket, delete your S3 Inventory configuration. Otherwise, S3 delivers reports on a daily or weekly schedule.

An inventory list isn't a single point-in-time view of all objects. Inventory lists are a rolling snapshot of bucket items, which are eventually consistent (for example, the list might not include recently added or deleted objects). Combining S3 Inventory and S3 Batch Operations works best when you work with static objects, or with an object set that you created two or more days ago. To work with more recent data, use the [ListObjectsV2](#) (GET Bucket) API operation to build your list of objects manually. If needed, repeat the process for the next few days or until your inventory report shows the desired status for all keys.

Step 2: Filter your object list with S3 Select

After you receive your S3 Inventory report, you can filter the report's contents to list only the objects that aren't encrypted with Bucket Keys. If you want all your bucket's objects encrypted with Bucket Keys, you can ignore this step. However, filtering your S3 Inventory report at this stage saves you the time and expense of re-encrypting objects that you previously encrypted.

Although the following steps show how to filter using [Amazon S3 Select](#), you can also use [Amazon Athena](#). To decide which tool to use, look at your S3 Inventory report's `manifest.json` file. This file lists the number of data files that are associated with that report. If the number is large, use Amazon Athena because it runs across multiple S3 objects, whereas S3 Select works on one object at a time. For more information about using Amazon S3 and Athena together, see [Querying Amazon S3 Inventory with Amazon Athena \(p. 749\)](#) and [Using Athena](#) in the blog post [Encrypting objects with Amazon S3 Batch Operations](#).

To filter your S3 Inventory report using S3 Select

1. Open the `manifest.json` file from your inventory report and look at the `fileSchema` section of the JSON. This informs the query that you run on the data.

The following JSON is an example `manifest.json` file for a CSV-formatted inventory on a bucket with versioning enabled. Depending on how you configured your inventory report, your manifest might look different.

```
{
```

```
"sourceBucket": "batchoperationsdemo",
"destinationBucket": "arn:aws:s3:::testbucket",
"version": "2021-05-22",
"creationTimestamp": "1558656000000",
"fileFormat": "CSV",
"fileSchema": "Bucket, Key, VersionId, IsLatest, IsDeleteMarker, BucketKeyStatus",
"files": [
    {
        "key": "demoinv/batchoperationsdemo/DemoInventory/data/009a40e4-f053-4c16-8c75-6100f8892202.csv.gz",
        "size": 72691,
        "MD5checksum": "c24c831717a099f0ebe4a9d1c5d3935c"
    }
]
```

If versioning isn't activated on the bucket, or if you choose to run the report for the latest versions, the `fileSchema` is `Bucket`, `Key`, and `BucketKeyStatus`.

If versioning *is* activated, depending on how you set up the inventory report, the `fileSchema` might include the following: `Bucket`, `Key`, `VersionId`, `IsLatest`, `IsDeleteMarker`, `BucketKeyStatus`. So pay attention to columns 1, 2, 3, and 6 when you run your query.

S3 Batch Operations needs the bucket, key, and version ID as inputs to perform the job, in addition to the field to search by, which is Bucket Key status. You don't need the version ID field, but it helps to specify it when you operate on a versioned bucket. For more information, see [Working with objects in a versioning-enabled bucket \(p. 649\)](#).

2. Locate the data files for the inventory report. The `manifest.json` object lists the data files under **files**.
3. After you locate and select the data file in the S3 console, choose **Actions**, and then choose **Query with S3 Select**.
4. Keep the preset **CSV**, **Comma**, and **GZIP** fields selected, and choose **Next**.
5. To review your inventory report's format before proceeding, choose **Show file preview**.
6. Enter the columns to reference in the SQL expression field, and choose **Run SQL**. The following expression returns columns 1–3 for all objects without Bucket Key configured.

```
select s._1, s._2, s._3 from s3object s where s._6 = 'DISABLED'
```

The following are example results.

```
batchoperationsdemo,0100059%7Ethumb.jpg,lsrtIxksLu0R0ZkYPL.LhgD5caTYn6vu
batchoperationsdemo,0100074%7Ethumb.jpg,sd2M60g6Fdazoi6D5kNARIE7KzUibmHR
batchoperationsdemo,0100075%7Ethumb.jpg,TLYESLn1l1mXD5c4BwiOIinqFrktddkOL
batchoperationsdemo,0200147%7Ethumb.jpg,amufzfMi_fEw0Rs99rxR_HrDF1E.13Y0
batchoperationsdemo,0301420%7Ethumb.jpg,9qGU2SEscL.C.c_sK89trmXYIwooABSh
batchoperationsdemo,0401524%7Ethumb.jpg,ORnEWNuB1QhHrrYAGFsZhbyvEYJ3DUor
batchoperationsdemo,200907200065HQ%7Ethumb.jpg,d8LgvIVjbDR5mUvW6pu9ahTfReyn5V4
batchoperationsdemo,200907200076HQ%7Ethumb.jpg,XUT25d7.gK40u_Gmnupdazg3BVx2jN40
batchoperationsdemo,201103190002HQ%7Ethumb.jpg,z.2sVRh0myqVi0BuIrngWlsRPQdb7qOS
```

7. Download the results, save them into a CSV format, and upload them to Amazon S3 as your list of objects for the S3 Batch Operations job.
8. If you have multiple manifest files, run **Query with S3 Select** on those also. Depending on the size of the results, you could combine the lists and run a single S3 Batch Operations job or run each list as a separate job.

Consider the [price](#) of running each S3 Batch Operations job when you decide the number of jobs to run.

Step 3: Set up and run your S3 Batch Operations job

Now that you have your filtered CSV lists of S3 objects, you can begin the S3 Batch Operations job to encrypt the objects with S3 Bucket Keys.

A *job* refers collectively to the list (manifest) of objects provided, the operation performed, and the specified parameters. The easiest way to encrypt this set of objects is by using the `PUT` copy operation and specifying the same destination prefix as the objects listed in the manifest. This either overwrites the existing objects in an unversioned bucket or, with versioning turned on, creates a newer, encrypted version of the objects.

As part of copying the objects, specify that Amazon S3 should encrypt the object with SSE-KMS encryption and S3. This job copies the objects, so all your objects show an updated creation date upon completion, regardless of when you originally added them to S3. Also specify the other properties for your set of objects as part of the S3 Batch Operations job, including object tags and storage class.

Substeps

- [Set up your IAM policy \(p. 904\)](#)
- [Set up your Batch Operations IAM role \(p. 905\)](#)
- [Turn on S3 Bucket Keys for an existing bucket \(p. 906\)](#)
- [Create your Batch Operations job \(p. 906\)](#)
- [Run your Batch Operations job \(p. 906\)](#)
- [Things to note \(p. 907\)](#)

Set up your IAM policy

1. Open the IAM console at <https://console.aws.amazon.com/iam/>.
2. In the navigation pane, choose **Policy**, and then choose **Create Policy**.
3. Choose the **JSON** tab. Choose **Edit policy** and add the example IAM policy that appears in the following code block.

After copying the policy example into your IAM console, replace the following:

- a. Replace `{SOURCE_BUCKET_FOR_COPY}` with the name of your source bucket.
- b. Replace `{DESTINATION_BUCKET_FOR_COPY}` with the name of your destination bucket.
- c. Replace `{MANIFEST_KEY}` with the name of your manifest object.
- d. Replace `{REPORT_BUCKET}` with the name of the bucket where you want to save reports.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "CopyObjectsToEncrypt",
            "Effect": "Allow",
            "Action": [
                "s3:PutObject",
                "s3:PutObjectTagging",
                "s3:PutObjectAcl",
                "s3:PutObjectVersionTagging",
                "s3:PutObjectVersionAcl",
                "s3:GetObject",
                "s3:GetObjectAcl",
                "s3:GetObjectTagging",
                "s3:GetObjectVersion",
                "s3:GetObjectVersionAcl",
                "s3:DeleteObject"
            ],
            "Resource": [
                "arn:aws:s3:::{SOURCE_BUCKET_FOR_COPY}/*",
                "arn:aws:s3:::{DESTINATION_BUCKET_FOR_COPY}/*"
            ]
        }
    ]
}
```

```

        "s3:GetObjectVersionTagging"
    ],
    "Resource": [
        "arn:aws:s3:::{SOURCE_BUCKET_FOR_COPY}/*",
        "arn:aws:s3:::{DESTINATION_BUCKET_FOR_COPY}/*"
    ]
},
{
    "Sid": "ReadManifest",
    "Effect": "Allow",
    "Action": [
        "s3:GetObject",
        "s3:GetObjectVersion"
    ],
    "Resource": "arn:aws:s3:::{MANIFEST_KEY}"
},
{
    "Sid": "WriteReport",
    "Effect": "Allow",
    "Action": [
        "s3:PutObject"
    ],
    "Resource": "arn:aws:s3:::{REPORT_BUCKET}/*"
}
]
}

```

4. Choose **Next: Tags**.
5. Add any tags that you want (optional), and choose **Next: Review**.
6. Add a policy name, optionally add a description, and choose **Create policy**.
7. Choose **Review policy and Save changes**.
8. With your S3 Batch Operations policy now complete, the console returns you to the **IAM Policies** page. Filter on the policy name, choose the button to the left of the policy name, choose **Policy actions**, and choose **Attach**.

To attach the newly created policy to an IAM role, select the appropriate users, groups, or roles in your account and choose **Attach policy**. This takes you back to the IAM console.

Set up your Batch Operations IAM role

1. On the IAM console, in the navigation pane, choose **Roles**, and then choose **Create role**.
2. Choose **AWS service, S3, and S3 Batch Operations**. Then choose **Next: Permissions**.
3. Start entering the name of the IAM **policy** that you just created. Select the check box by the policy name when it appears, and choose **Next: Tags**.
4. (Optional) Add tags or keep the key and value fields blank for this exercise. Choose **Next: Review**.
5. Enter a role name, and accept the default description or add your own. Choose **Create role**.
6. Ensure that the user creating the job has the permissions in the following example.

Replace **{ACCOUNT-ID}** with your AWS account ID and **{IAM_ROLE_NAME}** with the name that you plan to apply to the IAM role that you will create in the Batch Operations job creation step later. For more information, see [Granting permissions for Amazon S3 Batch Operations \(p. 882\)](#).

```

{
    "Sid": "AddIamPermissions",
    "Effect": "Allow",
    "Action": [
        "iam:GetRole",
        "iam:PassRole"
    ],
}

```

```
"Resource": "arn:aws:iam::${ACCOUNT-ID}:role/${IAM_ROLE_NAME}"  
}
```

Turn on S3 Bucket Keys for an existing bucket

1. Open the Amazon S3 console at <https://console.aws.amazon.com/s3/>.
2. In the **Buckets** list, choose the bucket that you want to turn on an S3 Bucket Keys for.
3. Choose **Properties**.
4. Under **Default encryption**, choose **Edit**.
5. Under **Server-side encryption** options, choose **Enable**.
6. Under **Encryption key type**, choose **AWS KMS key (SSE-KMS)** and choose the AWS KMS key format that you prefer:
 - **AWS managed key (aws/s3)**.
 - **Choose from your AWS KMS keys**, and choose a symmetric encryption KMS key in the same Region as your bucket.
 - **AWS KMS key ARN**
7. Under **Bucket Key**, choose **Enable**, and then choose **Save changes**.

Now that Bucket Key is turned on at the bucket level, objects that are uploaded, modified, or copied into this bucket will inherit this encryption configuration by default. This includes objects copied using Amazon S3 Batch Operations.

Create your Batch Operations job

1. Open the Amazon S3 console at <https://console.aws.amazon.com/s3/>.
2. In the navigation pane, choose **Batch Operations**, and then choose **Create Job**.
3. Choose the **Region** where you store your objects, and choose **CSV** as the manifest type.
4. Enter the path or navigate to the CSV manifest file that you created earlier from S3 Select (or Athena) results. If your manifest contains version IDs, select that box. Choose **Next**.
5. Choose the **Copy** operation, and choose the copy destination bucket. You can keep server-side encryption disabled. As long as the bucket destination has Bucket Key enabled, the copy operation applies Bucket Key at the destination bucket.
6. (Optional) Choose a storage class and the other parameters as desired. The parameters that you specify in this step apply to all operations performed on the objects that are listed in the manifest. Choose **Next**.
7. Give your job a description (or keep the default), set its priority level, choose a report type, and specify the **Path to completion report destination**.
8. In the **Permissions** section, be sure to choose the Batch Operations IAM role that you defined earlier. Choose **Next**.
9. Under **Review**, verify the settings. If you want to make changes, choose **Previous**. After confirming the Batch Operations settings, choose **Create job**.

For more information, see [Creating an S3 Batch Operations job \(p. 889\)](#).

Run your Batch Operations job

The setup wizard automatically returns you to the S3 Batch Operations section of the Amazon S3 console. Your new job transitions from the **New** state to the **Preparing** state as S3 begins the process. During the Preparing state, S3 reads the job's manifest, checks it for errors, and calculates the number of objects.

1. Choose the refresh button in the Amazon S3 console to check progress. Depending on the size of the manifest, reading can take minutes or hours.
2. After S3 finishes reading the job's manifest, the job moves to the **Awaiting your confirmation** state. Choose the option button to the left of the Job ID, and choose **Run job**.
3. Check the settings for the job, and choose **Run job** in the bottom-right corner.

After the job begins running, you can choose the refresh button to check progress through the console dashboard view or by selecting the specific job.

4. When the job is complete, you can view the **Successful** and **Failed** object counts to confirm that everything performed as expected. If you enabled job reports, check your job report for the exact cause of any failed operations.

You can also perform these steps using the AWS CLI, SDKs, or APIs. For more information about tracking job status and completion reports, see [Tracking job status and completion reports \(p. 922\)](#).

Things to note

Consider the following issues when you use S3 Batch Operations to encrypt objects with Bucket Keys:

- You will be charged for S3 Batch Operations jobs, objects, and requests in addition to any charges associated with the operation that S3 Batch Operations performs on your behalf, including data transfer, requests, and other charges. For more information, see [Amazon S3 pricing](#).
- If you use a versioned bucket, each S3 Batch Operations job performed creates new encrypted versions of your objects. It also maintains the previous versions without Bucket Key configured. To delete the old versions, set up an S3 Lifecycle expiration policy for noncurrent versions as described in [Lifecycle configuration elements \(p. 721\)](#).
- The copy operation creates new objects with new creation dates, which can affect lifecycle actions like archiving. If you copy all objects in your bucket, all the new copies have identical or similar creation dates. To further identify these objects and create different lifecycle rules for various data subsets, consider using object tags.

Summary

In this section, you sorted existing objects to filter out already encrypted data. Then you applied the Bucket Key feature on unencrypted objects by using S3 Batch Operations to copy existing data to a bucket with Bucket Key activated. This process can save you time and money while allowing you to complete operations such as encrypting all existing objects.

For more information about S3 Batch Operations, see [Performing large-scale batch operations on Amazon S3 objects \(p. 881\)](#).

For examples that show the copy operation with tags using the AWS CLI and AWS SDK for Java, see [Creating a Batch Operations job with job tags used for labeling \(p. 933\)](#).

Invoke AWS Lambda function

The **Invoke AWS Lambda function** initiates AWS Lambda functions to perform custom actions on objects that are listed in a manifest. This section describes how to create a Lambda function to use with S3 Batch Operations and how to create a job to invoke the function. The S3 Batch Operations job uses the `LambdaInvoke` operation to run a Lambda function on every object listed in a manifest.

You can work with S3 Batch Operations for Lambda using the AWS Management Console, AWS Command Line Interface (AWS CLI), AWS SDKs, or REST APIs. For more information about using Lambda, see [Getting Started with AWS Lambda](#) in the *AWS Lambda Developer Guide*.

The following sections explain how you can get started using S3 Batch Operations with Lambda.

Topics

- [Using Lambda with Amazon S3 batch operations \(p. 908\)](#)
- [Creating a Lambda function to use with S3 Batch Operations \(p. 908\)](#)
- [Creating an S3 Batch Operations job that invokes a Lambda function \(p. 912\)](#)
- [Providing task-level information in Lambda manifests \(p. 912\)](#)

Using Lambda with Amazon S3 batch operations

When using S3 Batch Operations with AWS Lambda, you must create new Lambda functions specifically for use with S3 Batch Operations. You can't reuse existing Amazon S3 event-based functions with S3 Batch Operations. Event functions can only receive messages; they don't return messages. The Lambda functions that are used with S3 Batch Operations must accept and return messages. For more information about using Lambda with Amazon S3 events, see [Using AWS Lambda with Amazon S3](#) in the [AWS Lambda Developer Guide](#).

You create an S3 Batch Operations job that invokes your Lambda function. The job runs the same Lambda function on all of the objects listed in your manifest. You can control what versions of your Lambda function to use while processing the objects in your manifest. S3 Batch Operations support unqualified Amazon Resource Names (ARNs), aliases, and specific versions. For more information, see [Introduction to AWS Lambda Versioning](#) in the [AWS Lambda Developer Guide](#).

If you provide the S3 Batch Operations job with a function ARN that uses an alias or the `$LATEST` qualifier, and you update the version that either of those points to, S3 Batch Operations starts calling the new version of your Lambda function. This can be useful when you want to update functionality part of the way through a large job. If you don't want S3 Batch Operations to change the version that is used, provide the specific version in the `FunctionARN` parameter when you create your job.

Response and result codes

There are two levels of codes that S3 Batch Operations expect from Lambda functions. The first is the response code for the entire request, and the second is a per-task result code. The following table contains the response codes.

Response code	Description
Succeeded	The task completed normally. If you requested a job completion report, the task's result string is included in the report.
TemporaryFailure	The task suffered a temporary failure and will be redriven before the job completes. The result string is ignored. If this is the final redrive, the error message is included in the final report.
PermanentFailure	The task suffered a permanent failure. If you requested a job-completion report, the task is marked as Failed and includes the error message string. Result strings from failed tasks are ignored.

Creating a Lambda function to use with S3 Batch Operations

This section provides example AWS Identity and Access Management (IAM) permissions that you must use with your Lambda function. It also contains an example Lambda function to use with S3 Batch

Operations. If you have never created a Lambda function before, see [Tutorial: Using AWS Lambda with Amazon S3](#) in the *AWS Lambda Developer Guide*.

You must create Lambda functions specifically for use with S3 Batch Operations. You can't reuse existing Amazon S3 event-based Lambda functions. This is because Lambda functions that are used for S3 Batch Operations must accept and return special data fields.

Important

AWS Lambda functions written in Java accept either [RequestHandler](#) or [RequestStreamHandler](#) handler interfaces. However, to support S3 Batch Operations request and response format, AWS Lambda requires the [RequestStreamHandler](#) interface for custom serialization and deserialization of a request and response. This interface allows Lambda to pass an [InputStream](#) and [OutputStream](#) to the Java [handleRequest](#) method.

Be sure to use the [RequestStreamHandler](#) interface when using Lambda functions with S3 Batch Operations. If you use a [RequestHandler](#) interface, the batch job will fail with "Invalid JSON returned in Lambda payload" in the completion report.

For more information, see [Handler interfaces](#) in the *AWS Lambda User Guide*.

Example IAM permissions

The following are examples of the IAM permissions that are necessary to use a Lambda function with S3 Batch Operations.

Example — S3 Batch Operations trust policy

The following is an example of the trust policy that you can use for the Batch Operations IAM role. This IAM role is specified when you create the job and gives Batch Operations permission to assume the IAM role.

```
{  
    "Version": "2012-10-17",  
    "Statement": [  
        {  
            "Effect": "Allow",  
            "Principal": {  
                "Service": "batchoperations.s3.amazonaws.com"  
            },  
            "Action": "sts:AssumeRole"  
        }  
    ]  
}
```

Example — Lambda IAM policy

The following is an example of an IAM policy that gives S3 Batch Operations permission to invoke the Lambda function and read the input manifest.

```
{  
    "Version": "2012-10-17",  
    "Statement": [  
        {  
            "Sid": "BatchOperationsLambdaPolicy",  
            "Effect": "Allow",  
            "Action": [  
                "s3:GetObject",  
                "s3:GetObjectVersion",  
                "s3:PutObject",  
                "lambda:InvokeFunction"  
            ],  
            "Resource": "*"  
        }  
    ]  
}
```

```
        }
    ]  
}
```

Example request and response

This section provides request and response examples for the Lambda function.

Example Request

The following is a JSON example of a request for the Lambda function.

```
{
    "invocationSchemaVersion": "1.0",
    "invocationId": "YXNkbGZqYWRmaiBhc2RmdW9hZHNmZGpmaGFzbGtkaGZza2RmaAo",
    "job": {
        "id": "f3cc4f60-61f6-4a2b-8a21-d07600c373ce"
    },
    "tasks": [
        {
            "taskId": "dGFza2lkZ29lc2hlcUK",
            "s3Key": "customerImage1.jpg",
            "s3VersionId": "1",
            "s3BucketArn": "arn:aws:s3:us-east-1:0123456788:awsexamplebucket1"
        }
    ]
}
```

Example Response

The following is a JSON example of a response for the Lambda function.

```
{
    "invocationSchemaVersion": "1.0",
    "treatMissingKeysAs": "PermanentFailure",
    "invocationId": "YXNkbGZqYWRmaiBhc2RmdW9hZHNmZGpmaGFzbGtkaGZza2RmaAo",
    "results": [
        {
            "taskId": "dGFza2lkZ29lc2hlcUK",
            "resultCode": "Succeeded",
            "resultString": "[\"Mary Major\", \"John Stiles\"]"
        }
    ]
}
```

Example Lambda function for S3 Batch Operations

The following example Python Lambda removes a delete marker from a versioned object.

As the example shows, keys from S3 Batch Operations are URL encoded. To use Amazon S3 with other AWS services, it's important that you URL decode the key that is passed from S3 Batch Operations.

```
import logging
from urllib import parse
import boto3
from botocore.exceptions import ClientError

logger = logging.getLogger(__name__)
logger.setLevel('INFO')
```

```

s3 = boto3.client('s3')

def lambda_handler(event, context):
    """
    Removes a delete marker from the specified versioned object.

    :param event: The S3 batch event that contains the ID of the delete marker
                  to remove.
    :param context: Context about the event.
    :return: A result structure that Amazon S3 uses to interpret the result of the
            operation. When the result code is TemporaryFailure, S3 retries the
            operation.
    """
    # Parse job parameters from Amazon S3 batch operations
    invocation_id = event['invocationId']
    invocation_schema_version = event['invocationSchemaVersion']

    results = []
    result_code = None
    result_string = None

    task = event['tasks'][0]
    task_id = task['taskId']

    try:
        obj_key = parse.unquote(task['s3Key'], encoding='utf-8')
        obj_version_id = task['s3VersionId']
        bucket_name = task['s3BucketArn'].split(':')[ -1]

        logger.info("Got task: remove delete marker %s from object %s.",
                   obj_version_id, obj_key)

        try:
            # If this call does not raise an error, the object version is not a delete
            # marker and should not be deleted.
            response = s3.head_object(
                Bucket=bucket_name, Key=obj_key, VersionId=obj_version_id)
            result_code = 'PermanentFailure'
            result_string = f"Object {obj_key}, ID {obj_version_id} is not " \
                           f"a delete marker."

            logger.debug(response)
            logger.warning(result_string)
        except ClientError as error:
            delete_marker = error.response['ResponseMetadata']['HTTPHeaders'] \
                           .get('x-amz-delete-marker', 'false')
            if delete_marker == 'true':
                logger.info("Object %s, version %s is a delete marker.",
                           obj_key, obj_version_id)
                try:
                    s3.delete_object(
                        Bucket=bucket_name, Key=obj_key, VersionId=obj_version_id)
                    result_code = 'Succeeded'
                    result_string = f"Successfully removed delete marker " \
                                   f"{obj_version_id} from object {obj_key}."
                    logger.info(result_string)
                except ClientError as error:
                    # Mark request timeout as a temporary failure so it will be retried.
                    if error.response['Error']['Code'] == 'RequestTimeout':
                        result_code = 'TemporaryFailure'
                        result_string = f"Attempt to remove delete marker from " \
                                       f"object {obj_key} timed out."
                        logger.info(result_string)
                    else:
                        raise
                else:
                    raise
            else:
                raise
        else:
            raise
    except ClientError as error:
        # Mark request timeout as a temporary failure so it will be retried.
        if error.response['Error']['Code'] == 'RequestTimeout':
            result_code = 'TemporaryFailure'
            result_string = f"Attempt to remove delete marker from " \
                           f"object {obj_key} timed out."
            logger.info(result_string)
        else:
            raise
    else:
        raise

```

```
        else:
            raise ValueError(f"The x-amz-delete-marker header is either not "
                            f"present or is not 'true'.")}

    except Exception as error:
        # Mark all other exceptions as permanent failures.
        result_code = 'PermanentFailure'
        result_string = str(error)
        logger.exception(error)

    finally:
        results.append({
            'taskId': task_id,
            'resultCode': result_code,
            'resultString': result_string
        })
return {
    'invocationSchemaVersion': invocation_schema_version,
    'treatMissingKeysAs': 'PermanentFailure',
    'invocationId': invocation_id,
    'results': results
}
```

Creating an S3 Batch Operations job that invokes a Lambda function

When creating an S3 Batch Operations job to invoke a Lambda function, you must provide the following:

- The ARN of your Lambda function (which might include the function alias or a specific version number)
- An IAM role with permission to invoke the function
- The action parameter `LambdaInvokeFunction`

For more information about creating an S3 Batch Operations job, see [Creating an S3 Batch Operations job \(p. 889\)](#) and [Operations supported by S3 Batch Operations \(p. 895\)](#).

The following example creates an S3 Batch Operations job that invokes a Lambda function using the AWS CLI.

```
aws s3control create-job
  --account-id <AccountID>
  --operation '{"LambdaInvoke": { "FunctionArn":
    "arn:aws:lambda:Region:AccountID:function:LambdaFunctionName" } }'
  --manifest '{"Spec":{"Format":"S3BatchOperations_CSV_20180820","Fields":
  ["Bucket","Key"]}}, "Location":
  {"ObjectArn":"arn:aws:s3:::ManifestLocation","ETag":"ManifestETag"}}
  --report
  '{"Bucket":"arn:aws:s3:::awsexamplebucket1","Format":"Report_CSV_20180820","Enabled":true,"Prefix":"
  "Region","Priority":2,"RoleArn":arn:aws:iam::AccountID:role/BatchOperationsRole,"Region":Region,"Description":"
  "Lambda Function"}
```

Providing task-level information in Lambda manifests

When you use AWS Lambda functions with S3 Batch Operations, you might want additional data to accompany each task/key that is operated on. For example, you might want to have both a source object key and new object key provided. Your Lambda function could then copy the source key to a new S3 bucket under a new name. By default, Amazon S3 batch operations let you specify only the destination bucket and a list of source keys in the input manifest to your job. The following describes how you can include additional data in your manifest so that you can run more complex Lambda functions.

To specify per-key parameters in your S3 Batch Operations manifest to use in your Lambda function's code, use the following URL-encoded JSON format. The key field is passed to your Lambda function as if it were an Amazon S3 object key. But it can be interpreted by the Lambda function to contain other values or multiple keys, as shown following.

Note

The maximum number of characters for the key field in the manifest is 1,024.

Example — manifest substituting the "Amazon S3 keys" with JSON strings

The URL-encoded version must be provided to S3 Batch Operations.

```
my-bucket,{"origKey": "object1key", "newKey": "newObject1Key"}  
my-bucket,{"origKey": "object2key", "newKey": "newObject2Key"}  
my-bucket,{"origKey": "object3key", "newKey": "newObject3Key"}
```

Example — manifest URL-encoded

This URL-encoded version must be provided to S3 Batch Operations. The non-URL-encoded version does not work.

```
my-bucket,%7B%22origKey%22%3A%20%22object1key%22%2C%20%22newKey%22%3A%20%22newObject1Key  
%22%7D  
my-bucket,%7B%22origKey%22%3A%20%22object2key%22%2C%20%22newKey%22%3A%20%22newObject2Key  
%22%7D  
my-bucket,%7B%22origKey%22%3A%20%22object3key%22%2C%20%22newKey%22%3A%20%22newObject3Key  
%22%7D
```

Example — Lambda function with manifest format writing results to the job report

This Lambda function shows how to parse a pipe-delimited task that is encoded into the S3 Batch Operations manifest. The task indicates which revision operation is applied to the specified object.

```
import logging  
from urllib import parse  
import boto3  
from botocore.exceptions import ClientError  
  
logger = logging.getLogger(__name__)  
logger.setLevel('INFO')  
  
s3 = boto3.resource('s3')  
  
  
def lambda_handler(event, context):  
    """  
        Applies the specified revision to the specified object.  
  
    :param event: The Amazon S3 batch event that contains the ID of the object to  
                 revise and the revision type to apply.  
    :param context: Context about the event.  
    :return: A result structure that Amazon S3 uses to interpret the result of the  
            operation.  
    """  
    # Parse job parameters from Amazon S3 batch operations  
    invocation_id = event['invocationId']  
    invocation_schema_version = event['invocationSchemaVersion']  
  
    results = []  
    result_code = None  
    result_string = None
```

```

task = event['tasks'][0]
task_id = task['taskId']
# The revision type is packed with the object key as a pipe-delimited string.
obj_key, revision = \
    parse.unquote(task['s3Key'], encoding='utf-8').split('|')
bucket_name = task['s3BucketArn'].split(':')[ -1]

logger.info("Got task: apply revision %s to %s.", revision, obj_key)

try:
    stanza_obj = s3.Bucket(bucket_name).Object(obj_key)
    stanza = stanza_obj.get()['Body'].read().decode('utf-8')
    if revision == 'lower':
        stanza = stanza.lower()
    elif revision == 'upper':
        stanza = stanza.upper()
    elif revision == 'reverse':
        stanza = stanza[::-1]
    elif revision == 'delete':
        pass
    else:
        raise TypeError(f"Can't handle revision type '{revision}'.")

    if revision == 'delete':
        stanza_obj.delete()
        result_string = f"Deleted stanza {stanza_obj.key}."

    else:
        stanza_obj.put(Body=bytes(stanza, 'utf-8'))
        result_string = f"Applied revision type '{revision}' to " \
            f"stanza {stanza_obj.key}."

    logger.info(result_string)
    result_code = 'Succeeded'
except ClientError as error:
    if error.response['Error']['Code'] == 'NoSuchKey':
        result_code = 'Succeeded'
        result_string = f"Stanza {obj_key} not found, assuming it was deleted " \
            f"in an earlier revision."
        logger.info(result_string)
    else:
        result_code = 'PermanentFailure'
        result_string = f"Got exception when applying revision type '{revision}' " \
            f"to {obj_key}: {error}."
        logger.exception(result_string)
finally:
    results.append({
        'taskId': task_id,
        'resultCode': result_code,
        'resultString': result_string
    })
return {
    'invocationSchemaVersion': invocation_schema_version,
    'treatMissingKeysAs': 'PermanentFailure',
    'invocationId': invocation_id,
    'results': results
}

```

Replace all object tags

The **Replace all object tags** operation replaces the Amazon S3 object tags on every object listed in the manifest. An Amazon S3 object tag is a key-value pair of strings that you can use to store metadata about an object.

To create a Replace all object tags job, you provide a set of tags that you want to apply. S3 Batch Operations applies the same set of tags to every object. The tag set that you provide replaces whatever tag sets are already associated with the objects in the manifest. S3 Batch Operations does not support adding tags to objects while leaving the existing tags in place.

If the objects in your manifest are in a versioned bucket, you can apply the tag set to specific versions of every object. You do this by specifying a version ID for every object in the manifest. If you don't include a version ID for any object, then S3 Batch Operations applies the tag set to the latest version of every object.

Restrictions and limitations

- The AWS Identity and Access Management (IAM) role that you specify to run the Batch Operations job must have permissions to perform the underlying Amazon S3 Replace all object tags operation. For more information about the permissions required, see [PutObjectTagging](#) in the *Amazon Simple Storage Service API Reference*.
- S3 Batch Operations uses the Amazon S3 [PutObjectTagging](#) operation to apply tags to each object in the manifest. All restrictions and limitations that apply to the underlying operation also apply to S3 Batch Operations jobs.

For more information about using the console to create jobs, see [Creating an S3 batch operations job](#).

For more information about object tagging, see [Categorizing your storage using tags \(p. 825\)](#) in this guide, and see [PutObjectTagging](#), [GetObjectTagging](#), and [DeleteObjectTagging](#) in the *Amazon Simple Storage Service API Reference*.

Delete all object tags

The **Delete all object tags** operation removes all Amazon S3 object tag sets currently associated with the objects that are listed in the manifest. S3 Batch Operations does not support deleting tags from objects while keeping other tags in place.

If the objects in your manifest are in a versioned bucket, you can remove the tag sets from a specific version of an object. Do this by specifying a version ID for every object in the manifest. If you don't include a version ID for an object, S3 Batch Operations removes the tag set from the latest version of every object.

For more information about Batch Operations manifests, see [Specifying a manifest \(p. 890\)](#).

Warning

Running this job removes all object tag sets on every object listed in the manifest.

Restrictions and limitations

- The AWS Identity and Access Management (IAM) role that you specify to run the job must have permissions to perform the underlying Amazon S3 Delete object tagging operation. For more information, see [DeleteObjectTagging](#) in the *Amazon Simple Storage Service API Reference*.
- S3 Batch Operations uses the Amazon S3 [DeleteObjectTagging](#) operation to remove the tag sets from every object in the manifest. All restrictions and limitations that apply to the underlying operation also apply to S3 Batch Operations jobs.

For more information about creating jobs, see [Creating an S3 Batch Operations job \(p. 889\)](#).

For more details about object tagging, see [Replace all object tags \(p. 914\)](#) in this guide, and [PutObjectTagging](#), [GetObjectTagging](#), and [DeleteObjectTagging](#) in the *Amazon Simple Storage Service API Reference*.

Replace access control list

The **Replace access control list (ACL)** operation replaces the Amazon S3 access control lists (ACLs) for every object that is listed in the manifest. Using ACLs, you can define who can access an object and what actions they can perform.

S3 Batch Operations support custom ACLs that you define and canned ACLs that Amazon S3 provides with a predefined set of access permissions.

If the objects in your manifest are in a versioned bucket, you can apply the ACLs to specific versions of every object. You do this by specifying a version ID for every object in the manifest. If you don't include a version ID for any object, then S3 Batch Operations applies the ACL to the latest version of the object.

For more information about ACLs in Amazon S3, [Access control list \(ACL\) overview \(p. 554\)](#).

S3 Block Public Access

If you want to limit public access to all objects in a bucket, you should use Amazon S3 Block Public Access instead of S3 Batch Operations. Block Public Access can limit public access on a per-bucket or account-wide basis with a single, simple operation that takes effect quickly. This makes it a better choice when your goal is to control public access to all objects in a bucket or account. Use S3 Batch Operations when you need to apply a customized ACL to every object in the manifest. For more information about S3 Block Public Access, see [Blocking public access to your Amazon S3 storage \(p. 584\)](#).

S3 Object Ownership

If the objects in the manifest are in a bucket uses the bucket owner enforced setting for Object Ownership, the **Replace access control list (ACL)** operation can only specify object ACLs that grant full control to the bucket owner. The operation can't grant object ACL permissions to other AWS accounts or groups. For more information, see [Controlling ownership of objects and disabling ACLs for your bucket \(p. 601\)](#).

Restrictions and limitations

- The role that you specify to run the Replace access control list job must have permissions to perform the underlying Amazon S3 PutObjectAcl operation. For more information about the permissions required, see [PutObjectAcl](#) in the *Amazon Simple Storage Service API Reference*.
- S3 Batch Operations uses the Amazon S3 PutObjectAcl operation to apply the specified ACL to every object in the manifest. Therefore, all restrictions and limitations that apply to the underlying PutObjectAcl operation also apply to S3 Batch Operations Replace access control list jobs.

Restore objects

The **Restore** operation initiates restore requests for archived objects on a list of Amazon S3 objects that you specify. The following objects must be restored with an [S3 Initiate Restore Object](#) job before they can be accessed in real time:

- Objects archived in the S3 Glacier Flexible Retrieval or S3 Glacier Deep Archive storage classes
- Objects archived through the S3 Intelligent-Tiering storage class in the Archive Access or Deep Archive Access tiers

Using an S3 Initiate Restore Object operation in your S3 Batch Operations job results in a restore request for every object that is specified in the manifest.

Important

The S3 Initiate Restore Object job only *initiates* the request to restore objects. S3 Batch Operations reports the job as complete for each object after the request is initiated for that

object. Amazon S3 doesn't update the job or otherwise notify you when the objects have been restored. However, you can use event notifications to receive notifications when the objects are available in Amazon S3. For more information, see [Amazon S3 Event Notifications \(p. 1017\)](#).

Restoring archived files from the S3 Glacier Flexible Retrieval or S3 Glacier Deep Archive storage classes differs from restoring files from the S3 Intelligent-Tiering storage class in the Archive Access or Deep Archive Access tiers.

- When you restore from S3 Glacier Flexible Retrieval or S3 Glacier Deep Archive, a temporary copy of the object is created. Amazon S3 deletes this copy after `ExpirationInDays` days have elapsed. After this copy is deleted, you must submit an additional restore request to access it.
- When you restore from the S3 Intelligent-Tiering Archive Access or Deep Archive Access tiers, the object transitions back into the S3 Intelligent-Tiering Frequent Access tier. The object automatically transitions into the Archive Access tier after a minimum of 90 consecutive days of no access. It moves into the Deep Archive Access tier after a minimum of 180 consecutive days of no access. Do *not* specify the `ExpirationInDays` argument when restoring archived S3 Intelligent-Tiering objects.
- Batch Operations jobs can operate either on S3 Glacier Flexible Retrieval and S3 Glacier Deep Archive storage class objects or on S3 Intelligent-Tiering Archive Access and Deep Archive Access storage tier objects. They can't operate on both types in the same job. To restore objects of both types, you *must* create separate Batch Operations jobs.

To create an S3 Initiate Restore Object job, the following arguments are available:

ExpirationInDays

This argument specifies how long the S3 Glacier Flexible Retrieval or S3 Glacier Deep Archive object remains available in Amazon S3. Initiate Restore Object jobs that target S3 Glacier Flexible Retrieval and S3 Glacier Deep Archive objects require `ExpirationInDays` set to 1 or greater.

Conversely, do not set `ExpirationInDays` when creating S3 Initiate Restore Object operation jobs that target S3 Intelligent-Tiering Archive Access and Deep Archive Access tier objects. Objects in S3 Intelligent-Tiering archive access tiers are not subject to restore expiry, so specifying `ExpirationInDays` results in restore request failure.

GlacierJobTier

Amazon S3 can restore objects using one of three different retrieval tiers: EXPEDITED, STANDARD, and BULK. However, the S3 Batch Operations feature supports only the STANDARD and BULK retrieval tiers. For more about the differences between retrieval tiers, see [Archive retrieval options \(p. 671\)](#). For more information about pricing for each tier, see the **Requests & data retrievals** section on [Amazon S3 pricing](#).

Overlapping restores

If your [S3 Initiate Restore Object](#) job tries to restore an object that is already in the process of being restored, S3 Batch Operations proceeds as follows.

The restore operation succeeds for the object if either of the following conditions is true:

- Compared to the restoration request already in progress, this job's `ExpirationInDays` is the same and `GlacierJobTier` is faster.
- The previous restoration request has already completed, and the object is currently available. In this case, Batch Operations updates the expiration date of the restored object to match the `ExpirationInDays` specified in the in-progress restoration request.

The restore operation fails for the object if any of the following conditions are true:

- The restoration request already in progress has not yet completed, and the restoration duration for this job (specified by `ExpirationInDays`) is different from the restoration duration that is specified in the in-progress restoration request.
- The restoration tier for this job (specified by `GlacierJobTier`) is the same or slower than the restoration tier that is specified in the in-progress restoration request.

Limitations

S3 Initiate Restore Object jobs have the following limitations:

- You must create the job in the same Region as the archived objects.
- S3 Batch Operations does not support the `EXPEDITED` retrieval tier.
- S3 Batch Operations does not support restoring subsets of S3 Intelligent-Tiering or S3 Glacier Flexible Retrieval objects. You must call [RestoreObject](#) for this purpose.

For more information about restoring objects, see [Restoring an archived object \(p. 673\)](#).

S3 Object Lock retention

The **Object Lock retention** operation allows you to apply retention dates for your objects using either *governance* mode or *compliance* mode. These retention modes apply different levels of protection. You can apply either retention mode to any object version. Retention dates, like legal holds, prevent an object from being overwritten or deleted. Amazon S3 stores the *retain until date* specified in the object's metadata and protects the specified version of the object version until the retention period expires.

You can use S3 Batch Operations with Object Lock to manage retention dates of many Amazon S3 objects at once. You specify the list of target objects in your manifest and submit it to Batch Operations for completion. For more information, see S3 Object Lock [the section called "Retention periods" \(p. 682\)](#).

Your S3 Batch Operations job with retention dates runs *until completion, until cancellation, or until a failure state* is reached. You should use S3 Batch Operations and S3 Object Lock retention when you want to add, change, or remove the retention date for many objects with a single request.

Batch Operations verifies that Object Lock is enabled on your bucket before processing any keys in the manifest. To perform the operations and validation, Batch Operations needs `s3:GetBucketObjectLockConfiguration` and `s3:PutObjectRetention` permissions in an IAM role to allow Batch Operations to call Object Lock on your behalf. For more information, see [the section called "Managing Object Lock" \(p. 685\)](#).

For information about using this operation with the REST API, see `S3PutObjectRetention` in the [CreateJob](#) operation in the *Amazon Simple Storage Service API Reference*.

For an AWS Command Line Interface example of using this operation, see [the section called "Use Batch Operations with Object Lock retention" \(p. 946\)](#). For an AWS SDK for Java example, see [the section called "Use Batch Operations with Object Lock retention" \(p. 946\)](#).

Restrictions and limitations

- S3 Batch Operations does not make any bucket level changes.
- Versioning and S3 Object Lock must be configured on the bucket where the job is performed.
- All objects listed in the manifest must be in the same bucket.
- The operation works on the latest version of the object unless a version is explicitly specified in the manifest.

- You need `s3:PutObjectRetention` permission in your IAM role to use this.
- `s3:GetBucketObjectLockConfiguration` IAM permission is required to confirm that Object Lock is enabled for the S3 bucket.
- You can only extend the retention period of objects with COMPLIANCE mode retention dates applied, and it cannot be shortened.

S3 Object Lock legal hold

The **Object Lock legal hold** operation enables you to place a legal hold on an object version. Like setting a retention period, a legal hold prevents an object version from being overwritten or deleted. However, a legal hold doesn't have an associated retention period and remains in effect until removed.

You can use S3 Batch Operations with Object Lock to add legal holds to *many* Amazon S3 objects at once. You can do this by listing the target objects in your manifest and submitting that list to Batch Operations. Your S3 Batch Operations job with Object Lock legal hold runs until completion, until cancellation, or until a failure state is reached.

S3 Batch Operations verifies that Object Lock is enabled on your S3 bucket before processing any keys in the manifest. To perform the object operations and bucket level validation, S3 Batch Operations needs `s3:PutObjectLegalHold` and `s3:GetBucketObjectLockConfiguration` in an IAM role allowing S3 Batch Operations to call S3 Object Lock on your behalf.

When you create the S3 Batch Operations job to remove the legal hold, you just need to specify *Off* as the legal hold status. For more information, see [the section called "Managing Object Lock" \(p. 685\)](#).

For information about how to use this operation with the REST API, see `S3PutObjectLegalHold` in the [CreateJob](#) operation in the *Amazon Simple Storage Service API Reference*.

For an example use of this operation, see [Using the AWS SDK for Java \(p. 957\)](#).

Restrictions and limitations

- S3 Batch Operations does not make any bucket level changes.
 - All objects listed in the manifest must be in the same bucket.
 - Versioning and S3 Object Lock must be configured on the bucket where the job is performed.
 - The operation works on the latest version of the object unless a version is explicitly specified in the manifest.
 - `s3:PutObjectLegalHold` permission is required in your IAM role to add or remove legal hold from objects.
 - `s3:GetBucketObjectLockConfiguration` IAM permission is required to confirm that S3 Object Lock is enabled for the S3 bucket.
-
- [Copy objects \(p. 895\)](#)
 - [Invoke AWS Lambda function \(p. 907\)](#)
 - [Replace all object tags \(p. 914\)](#)
 - [Delete all object tags \(p. 915\)](#)
 - [Replace access control list \(p. 916\)](#)
 - [Restore objects \(p. 916\)](#)
 - [S3 Object Lock retention \(p. 918\)](#)
 - [S3 Object Lock legal hold \(p. 919\)](#)

- Replicating existing objects with [S3 Batch Replication](#) (p. 798)

Managing S3 Batch Operations jobs

Amazon S3 provides a robust set of tools to help you manage your S3 Batch Operations jobs after you create them. This section describes the operations that you can use to manage and track your jobs using the AWS Management Console, AWS CLI, AWS SDKs, or REST API.

Topics

- [Using the Amazon S3 console to manage your S3 Batch Operations jobs](#) (p. 920)
- [Listing jobs](#) (p. 920)
- [Viewing job details](#) (p. 921)
- [Assigning job priority](#) (p. 921)

Using the Amazon S3 console to manage your S3 Batch Operations jobs

Using the console, you can manage your S3 Batch Operations jobs. For example, you can:

- View active and queued jobs
- Change a job's priority
- Confirm and run a job
- Clone a job
- Cancel a job

To manage Batch Operations using the console

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 **Batch Operations**.
3. Choose the specific job that you would like to manage.

Listing jobs

You can retrieve a list of your S3 Batch Operations jobs. The list includes jobs that haven't yet finished and jobs that finished within the last 90 days. The job list includes information for each job, such as its ID, description, priority, current status, and the number of tasks that have succeeded and failed. You can filter your job list by status. When you retrieve a job list through the console, you can also search your jobs by description or ID and filter them by AWS Region.

Get a list of Active and Complete jobs

The following AWS CLI example gets a list of Active and Complete jobs.

```
aws s3control list-jobs \
--region us-west-2 \
--account-id acct-id \
--job-statuses '[ "Active", "Complete" ]' \
--max-results 20
```

Viewing job details

If you want more information about a job than you can retrieve by listing jobs, you can view all of the details for a single job. In addition to the information returned in a job list, a single job's details include other items, such as:

- The operation parameters
- Details about the manifest
- Information about the completion report (if you configured one when you created the job)
- The Amazon Resource Name (ARN) of the user role that you assigned to run the job

By viewing an individual job's details, you can access a job's entire configuration.

Get the description of an S3 Batch Operations job

The following example gets the description of an S3 Batch Operations job using the AWS CLI.

```
aws s3control describe-job \
  --region us-west-2 \
  --account-id acct-id \
  --job-id 00e123a4-c0d8-41f4-a0eb-b46f9ba5b07c
```

Assigning job priority

You can assign each job a numeric priority, which can be any positive integer. S3 Batch Operations prioritize jobs according to the assigned priority. Jobs with a higher priority (or a higher numeric value for the priority parameter) are evaluated first. Priority is determined in descending order. For example, a job queue with a priority value of 10 is given scheduling preference over a job queue with a priority value of 1.

You can change a job's priority while it is running. If you submit a new job with a higher priority while a job is running, the lower-priority job can pause to allow the higher-priority job to run.

Changing job priority does not affect job processing speed.

Note

S3 Batch Operations honor job priorities on a best-effort basis. Although jobs with higher priorities generally take precedence over jobs with lower priorities, Amazon S3 does not guarantee strict ordering of jobs.

Using the S3 console

How to update job priority in the AWS Management Console

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 **Batch Operations**.
3. Select the specific job that you would like to manage.
4. Choose **Action**. In the dropdown list, choose **Update priority**.

Using the AWS CLI

The following example updates the job priority using the AWS CLI. A higher number indicates a higher execution priority.

```
aws s3control update-job-priority \
```

```
--region us-west-2 \
--account-id acct-id \
--priority 98 \
--job-id 00e123a4-c0d8-41f4-a0eb-b46f9ba5b07c
```

Using the AWS SDK for Java

The following example updates the priority of an S3 Batch Operations job using the AWS SDK for Java.

For more information about job priority, see [Assigning job priority \(p. 921\)](#).

Example

```
package aws.example.s3control;

import com.amazonaws.AmazonServiceException;
import com.amazonaws.SdkClientException;
import com.amazonaws.auth.profile.ProfileCredentialsProvider;
import com.amazonaws.services.s3control.AWSS3Control;
import com.amazonaws.services.s3control.AWSS3ControlClient;
import com.amazonaws.services.s3control.model.UpdateJobPriorityRequest;

import static com.amazonaws.regions.Regions.US_WEST_2;

public class UpdateJobPriority {
    public static void main(String[] args) {
        String accountId = "Account ID";
        String jobId = "00e123a4-c0d8-41f4-a0eb-b46f9ba5b07c";

        try {
            AWSS3Control s3ControlClient = AWSS3ControlClient.builder()
                .withCredentials(new ProfileCredentialsProvider())
                .withRegion(US_WEST_2)
                .build();

            s3ControlClient.updateJobPriority(new UpdateJobPriorityRequest()
                .withAccountId(accountId)
                .withJobId(jobId)
                .withPriority(98));

        } catch (AmazonServiceException e) {
            // The call was transmitted successfully, but Amazon S3 couldn't process
            // it and returned an error response.
            e.printStackTrace();
        } catch (SdkClientException e) {
            // Amazon S3 couldn't be contacted for a response, or the client
            // couldn't parse the response from Amazon S3.
            e.printStackTrace();
        }
    }
}
```

Tracking job status and completion reports

With S3 Batch Operations, you can view and update job status, add notifications and logging, track job failures, and generate completion reports.

Topics

- [Job statuses \(p. 923\)](#)
- [Updating job status \(p. 925\)](#)
- [Notifications and logging \(p. 926\)](#)
- [Tracking job failure \(p. 926\)](#)
- [Completion reports \(p. 927\)](#)
- [Examples: Tracking an S3 Batch Operations job in Amazon EventBridge through AWS CloudTrail \(p. 927\)](#)
- [Examples: S3 Batch Operations completion reports \(p. 930\)](#)

Job statuses

After you create and run a job, it progresses through a series of statuses. The following table describes the statuses and the possible transitions between them.

Status	Description	Transitions
New	A job begins in the New state when you create it.	A job automatically moves to the Preparing state when Amazon S3 begins processing the manifest object.
Preparing	Amazon S3 is processing the manifest object and other job parameters to set up and run the job.	A job automatically moves to the Ready state after Amazon S3 finishes processing the manifest and other parameters. It is then ready to begin running the specified operation on the objects listed in the manifest. If the job requires confirmation before running, such as when you create a job using the Amazon S3 console, then the job transitions from Preparing to Suspended. It remains in the Suspended state until you confirm that you want to run it.
Suspended	The job requires confirmation, but you have not yet confirmed that you want to run it. Only jobs that you create using the Amazon S3 console require confirmation. A job that is created using the console enters the Suspended state immediately after Preparing. After you confirm that you want to run the job and the job becomes Ready, it never returns to the Suspended state.	After you confirm that you want to run the job, its status changes to Ready.
Ready	Amazon S3 is ready to begin running the requested object operations.	A job automatically moves to Active when Amazon S3 begins to run it. The amount of time

Status	Description	Transitions
		that a job remains in the <code>Ready</code> state depends on whether you have higher-priority jobs running already and how long those jobs take to complete.
Active	Amazon S3 is performing the requested operation on the objects listed in the manifest. While a job is <code>Active</code> , you can monitor its progress using the Amazon S3 console or the <code>DescribeJob</code> operation through the REST API, AWS CLI, or AWS SDKs.	A job moves out of the <code>Active</code> state when it is no longer running operations on objects. This can happen automatically, such as when a job completes successfully or fails. Or it can occur as a result of user actions, such as canceling a job. The state that the job moves to depends on the reason for the transition.
Pausing	The job is transitioning to <code>Paused</code> from another state.	A job automatically moves to <code>Paused</code> when the <code>Pausing</code> stage is finished.
Paused	A job can become <code>Paused</code> if you submit another job with a higher priority while the current job is running.	A <code>Paused</code> job automatically returns to <code>Active</code> after any higher-priority jobs that are blocking the job's execution complete, fail, or are suspended.
Complete	The job has finished performing the requested operation on all objects in the manifest. The operation might have succeeded or failed for every object. If you configured the job to generate a completion report, the report is available as soon as the job is <code>Complete</code> .	<code>Complete</code> is a terminal state. Once a job reaches <code>Complete</code> , it does not transition to any other state.
Cancelling	The job is transitioning to the <code>Cancelled</code> state.	A job automatically moves to <code>Cancelled</code> when the <code>Cancelling</code> stage is finished.
Cancelled	You requested that the job be canceled, and S3 Batch Operations has successfully cancelled the job. The job will not submit any new requests to Amazon S3.	<code>Cancelled</code> is a terminal state. After a job reaches <code>Cancelled</code> , it will not transition to any other state.
Failing	The job is transitioning to the <code>Failed</code> state.	A job automatically moves to <code>Failed</code> once the <code>Failing</code> stage is finished.

Status	Description	Transitions
Failed	The job has failed and is no longer running. For more information about job failures, see Tracking job failure (p. 926) .	Failed is a terminal state. After a job reaches Failed, it will not transition to any other state.

Updating job status

The following AWS CLI and SDK for Java examples update the status of a Batch Operations job. For more information about using the S3 console to manage Batch Operations jobs, see [Using the Amazon S3 console to manage your S3 Batch Operations jobs \(p. 920\)](#).

Using the AWS CLI

- If you didn't specify the `--no-confirmation-required` parameter in the previous `create-job` example, the job remains in a suspended state until you confirm the job by setting its status to Ready. Amazon S3 then makes the job eligible for execution.

```
aws s3control update-job-status \
--region us-west-2 \
--account-id 181572960644 \
--job-id 00e123a4-c0d8-41f4-a0eb-b46f9ba5b07c \
--requested-job-status 'Ready'
```

- Cancel the job by setting the job status to Cancelled.

```
aws s3control update-job-status \
--region us-west-2 \
--account-id 181572960644 \
--job-id 00e123a4-c0d8-41f4-a0eb-b46f9ba5b07c \
--status-update-reason "No longer needed" \
--requested-job-status Cancelled
```

Using the AWS SDK for Java

The following example updates the status of an S3 Batch Operations job using the AWS SDK for Java.

For more information about job status, see [Tracking job status and completion reports \(p. 922\)](#).

Example

```
package aws.example.s3control;

import com.amazonaws.AmazonServiceException;
import com.amazonaws.SdkClientException;
import com.amazonaws.auth.profile.ProfileCredentialsProvider;
import com.amazonaws.services.s3control.AWSS3Control;
import com.amazonaws.services.s3control.AWSS3ControlClient;
import com.amazonaws.services.s3control.model.UpdateJobStatusRequest;

import static com.amazonaws.regions.Regions.US_WEST_2;
```

```

public class UpdateJobStatus {
    public static void main(String[] args) {
        String accountId = "Account ID";
        String jobId = "00e123a4-c0d8-41f4-a0eb-b46f9ba5b07c";

        try {
            AWSS3Control s3ControlClient = AWSS3ControlClient.builder()
                .withCredentials(new ProfileCredentialsProvider())
                .withRegion(US_WEST_2)
                .build();

            s3ControlClient.updateJobStatus(new UpdateJobStatusRequest()
                .withAccountId(accountId)
                .withJobId(jobId)
                .withRequestedJobStatus("Ready"));

        } catch (AmazonServiceException e) {
            // The call was transmitted successfully, but Amazon S3 couldn't process
            // it and returned an error response.
            e.printStackTrace();
        } catch (SdkClientException e) {
            // Amazon S3 couldn't be contacted for a response, or the client
            // couldn't parse the response from Amazon S3.
            e.printStackTrace();
        }
    }
}

```

Notifications and logging

In addition to requesting completion reports, you can also capture, review, and audit Batch Operations activity using AWS CloudTrail. Because Batch Operations use existing Amazon S3 APIs to perform tasks, those tasks also emit the same events that they would if you called them directly. Thus, you can track and record the progress of your job and all of its tasks using the same notification, logging, and auditing tools and processes that you already use with Amazon S3. For more information, see the examples in the following sections.

Note

Amazon S3 Batch Operations generates both management and data events in CloudTrail during job execution. The volume of these events scale with the number of keys in each job's manifest. Refer to the [CloudTrail pricing](#) page for details, which includes examples of how pricing changes depending on the number of trails you have configured in your account. To learn how to configure and log events to fit your needs, see [Create your first trail](#) in the [AWS CloudTrail User Guide](#).

For more information about Amazon S3 events, see [Amazon S3 Event Notifications \(p. 1017\)](#).

Tracking job failure

If an S3 Batch Operations job encounters a problem that prevents it from running successfully, such as not being able to read the specified manifest, the job fails. When a job fails, it generates one or more failure codes or failure reasons. S3 Batch Operations store the failure codes and reasons with the job so that you can view them by requesting the job's details. If you requested a completion report for the job, the failure codes and reasons also appear there.

To prevent jobs from running a large number of unsuccessful operations, Amazon S3 imposes a task-failure threshold on every Batch Operations job. When a job has run at least 1,000 tasks, Amazon S3 monitors the task failure rate. At any point, if the failure rate (the number of tasks that have failed as a proportion of the total number of tasks that have run) exceeds 50 percent, the job fails. If your job fails

because it exceeded the task-failure threshold, you can identify the cause of the failures. For example, you might have accidentally included some objects in the manifest that don't exist in the specified bucket. After fixing the errors, you can resubmit the job.

Note

S3 Batch Operations operate asynchronously and the tasks don't necessarily run in the order that the objects are listed in the manifest. Therefore, you can't use the manifest ordering to determine which objects' tasks succeeded and which ones failed. Instead, you can examine the job's completion report (if you requested one) or view your AWS CloudTrail event logs to help determine the source of the failures.

Completion reports

When you create a job, you can request a completion report. As long as S3 Batch Operations successfully invoke at least one task, Amazon S3 generates a completion report after it finishes running tasks, fails, or is canceled. You can configure the completion report to include all tasks or only failed tasks.

The completion report includes the job configuration and status and information for each task, including the object key and version, status, error codes, and descriptions of any errors. Completion reports provide an easy way to view the results of your tasks in a consolidated format with no additional setup required. For an example of a completion report, see [Examples: S3 Batch Operations completion reports \(p. 930\)](#).

If you don't configure a completion report, you can still monitor and audit your job and its tasks using CloudTrail and Amazon CloudWatch. For more information, see the following section.

Topics

- [Examples: Tracking an S3 Batch Operations job in Amazon EventBridge through AWS CloudTrail \(p. 927\)](#)
- [Examples: S3 Batch Operations completion reports \(p. 930\)](#)

Examples: Tracking an S3 Batch Operations job in Amazon EventBridge through AWS CloudTrail

Amazon S3 Batch Operations job activity is recorded as events in AWS CloudTrail. You can create a custom rule in Amazon EventBridge and send these events to the target notification resource of your choice, such as Amazon Simple Notification Service (Amazon SNS).

Note

Amazon EventBridge is the preferred way to manage your events. Amazon CloudWatch Events and EventBridge are the same underlying service and API, but EventBridge provides more features. Changes that you make in either CloudWatch or EventBridge appear in each console. For more information, see the [Amazon EventBridge User Guide](#).

Tracking Examples

- [S3 Batch Operations events recorded in CloudTrail \(p. 927\)](#)
- [EventBridge rule for tracking S3 Batch Operations job events \(p. 928\)](#)

S3 Batch Operations events recorded in CloudTrail

When a Batch Operations job is created, it is recorded as a `JobCreated` event in CloudTrail. As the job runs, it changes state during processing, and other `JobStatusChanged` events are recorded in CloudTrail. You can view these events on the [CloudTrail console](#). For more information about CloudTrail, see the [AWS CloudTrail User Guide](#).

Note

Only S3 Batch Operations job status-change events are recorded in CloudTrail.

Example S3 Batch Operations job completion event recorded by CloudTrail

```
{  
    "eventVersion": "1.05",  
    "userIdentity": {  
        "accountId": "123456789012",  
        "invokedBy": "s3.amazonaws.com"  
    },  
    "eventTime": "2020-02-05T18:25:30Z",  
    "eventSource": "s3.amazonaws.com",  
    "eventName": "JobStatusChanged",  
    "awsRegion": "us-west-2",  
    "sourceIPAddress": "s3.amazonaws.com",  
    "userAgent": "s3.amazonaws.com",  
    "requestParameters": null,  
    "responseElements": null,  
    "eventID": "f907577b-bf3d-4c53-b9ed-8a83a118a554",  
    "readOnly": false,  
    "eventType": "AwsServiceEvent",  
    "recipientAccountId": "123412341234",  
    "serviceEventDetails": {  
        "jobId": "d6e58ec4-897a-4b6d-975f-10d7f0fb63ce",  
        "jobArn": "arn:aws:s3:us-west-2:181572960644:job/  
d6e58ec4-897a-4b6d-975f-10d7f0fb63ce",  
        "status": "Complete",  
        "jobEventId": "b268784cf0a66749f1a05bce259804f5",  
        "failureCodes": [],  
        "statusChangeReason": []  
    }  
}
```

EventBridge rule for tracking S3 Batch Operations job events

The following example shows how to create a rule in Amazon EventBridge to capture S3 Batch Operations events recorded by AWS CloudTrail to a target of your choice.

To do this, you create a rule by following all the steps in [Creating an EventBridge Rule That Triggers on an AWS API Call Using CloudTrail](#). You paste the following S3 Batch Operations custom event pattern policy where applicable, and choose the target service of your choice.

S3 Batch Operations custom event pattern policy

```
{  
    "source": [  
        "aws.s3"  
    ],  
    "detail-type": [  
        "AWS Service Event via CloudTrail"  
    ],  
    "detail": {  
        "eventSource": [  
            "s3.amazonaws.com"  
        ],  
        "eventName": [  
            "JobCreated",  
            "JobStatusChanged"  
        ]  
    }  
}
```

The following examples are two Batch Operations events that were sent to Amazon Simple Queue Service (Amazon SQS) from an EventBridge event rule. A Batch Operations job goes through many different states while processing (New, Preparing, Active, etc.), so you can expect to receive several messages for each job.

Example JobCreated sample event

```
{
    "version": "0",
    "id": "51dc8145-541c-5518-2349-56d7dffdf2d8",
    "detail-type": "AWS Service Event via CloudTrail",
    "source": "aws.s3",
    "account": "123456789012",
    "time": "2020-02-27T15:25:49Z",
    "region": "us-east-1",
    "resources": [],
    "detail": {
        "eventVersion": "1.05",
        "userIdentity": {
            "accountId": "11112223334444",
            "invokedBy": "s3.amazonaws.com"
        },
        "eventTime": "2020-02-27T15:25:49Z",
        "eventSource": "s3.amazonaws.com",
        "eventName": "JobCreated",
        "awsRegion": "us-east-1",
        "sourceIPAddress": "s3.amazonaws.com",
        "userAgent": "s3.amazonaws.com",
        "eventID": "7c38220f-f80b-4239-8b78-2ed867b7d3fa",
        "readOnly": false,
        "eventType": "AwsServiceEvent",
        "serviceEventDetails": {
            "jobId": "e849b567-5232-44be-9a0c-40988f14e80c",
            "jobArn": "arn:aws:s3:us-east-1:181572960644:job/e849b567-5232-44be-9a0c-40988f14e80c",
            "status": "New",
            "jobEventId": "f177ff24f1f097b69768e327038f30ac",
            "failureCodes": [],
            "statusChangeReason": []
        }
    }
}
```

Example JobStatusChanged job completion event

```
{
    "version": "0",
    "id": "c8791abf-2af8-c754-0435-fd869ce25233",
    "detail-type": "AWS Service Event via CloudTrail",
    "source": "aws.s3",
    "account": "123456789012",
    "time": "2020-02-27T15:26:42Z",
    "region": "us-east-1",
    "resources": [],
    "detail": {
        "eventVersion": "1.05",
        "userIdentity": {
            "accountId": "11112223334444",
            "invokedBy": "s3.amazonaws.com"
        },
        "eventTime": "2020-02-27T15:26:42Z",
        "eventSource": "s3.amazonaws.com",
```

```

        "eventName": "JobStatusChanged",
        "awsRegion": "us-east-1",
        "sourceIPAddress": "s3.amazonaws.com",
        "userAgent": "s3.amazonaws.com",
        "eventID": "0238c1f7-c2b0-440b-8dbd-1ed5e5833afb",
        "readOnly": false,
        "eventType": "AwsServiceEvent",
        "serviceEventDetails": {
            "jobId": "e849b567-5232-44be-9a0c-40988f14e80c",
            "jobArn": "arn:aws:s3:us-east-1:181572960644:job/e849b567-5232-44be-9a0c-40988f14e80c",
            "status": "Complete",
            "jobEventId": "51f5ac17dba408301d56cd1b2c8d1e9e",
            "failureCodes": [],
            "statusChangeReason": []
        }
    }
}

```

Examples: S3 Batch Operations completion reports

When you create an S3 Batch Operations job, you can request a completion report for all tasks or just for failed tasks. As long as at least one task has been invoked successfully, S3 Batch Operations generates a report for jobs that have completed, failed, or been canceled.

The completion report contains additional information for each task, including the object key name and version, status, error codes, and descriptions of any errors. The description of errors for each failed task can be used to diagnose issues that occur during job creation, such as permissions.

Example top-level manifest result file

The top-level `manifest.json` file contains the locations of each succeeded report and (if the job had any failures) the location of failed reports, as shown in the following example.

```
{
    "Format": "Report_CSV_20180820",
    "ReportCreationDate": "2019-04-05T17:48:39.725Z",
    "Results": [
        {
            "TaskExecutionStatus": "succeeded",
            "Bucket": "my-job-reports",
            "MD5Checksum": "83b1c4cbe93fc893f54053697e10fd6e",
            "Key": "job-f8fb9d89-a3aa-461d-bddc-ea6a1b131955/results/6217b0fab0de85c408b4be96aeaca9b195a7daa5.csv"
        },
        {
            "TaskExecutionStatus": "failed",
            "Bucket": "my-job-reports",
            "MD5Checksum": "22ee037f3515975f7719699e5c416eaa",
            "Key": "job-f8fb9d89-a3aa-461d-bddc-ea6a1b131955/results/b2ddad417e94331e9f37b44f1faf8c7ed5873f2e.csv"
        }
    ],
    "ReportSchema": "Bucket, Key, VersionId, TaskStatus, ErrorCode, HttpStatusCode, ResultMessage"
}
```

Example failed tasks reports

Failed tasks reports contain the following information for all *failed* tasks:

- Bucket
- Key
- VersionId
- TaskStatus
- ErrorCode
- HttpStatusCode
- ResultMessage

The following example report shows a case in which the AWS Lambda function timed out, causing failures to exceed the failure threshold. It was then marked as a `PermanentFailure`.

```
awsexamplebucket1,image_14975,,failed,200,PermanentFailure,"Lambda returned function error:
{""errorMessage":""}"2019-04-05T17:35:21.155Z 2845ca0d-38d9-4c4b-abcf-379dc749c452 Task
timed out after 3.00 seconds"}"
awsexamplebucket1,image_15897,,failed,200,PermanentFailure,"Lambda returned function error:
{""errorMessage":""}"2019-04-05T17:35:29.610Z 2d0a330b-de9b-425f-b511-29232fde5fe4 Task
timed out after 3.00 seconds"}"
awsexamplebucket1,image_14819,,failed,200,PermanentFailure,"Lambda returned function error:
{""errorMessage":""}"2019-04-05T17:35:22.362Z fcf5efde-74d4-4e6d-b37a-c7f18827f551 Task
timed out after 3.00 seconds"}"
awsexamplebucket1,image_15930,,failed,200,PermanentFailure,"Lambda returned function error:
{""errorMessage":""}"2019-04-05T17:35:29.809Z 3dd5b57c-4a4a-48aa-8a35-cbf027b7957e Task
timed out after 3.00 seconds"}"
awsexamplebucket1,image_17644,,failed,200,PermanentFailure,"Lambda returned function error:
{""errorMessage":""}"2019-04-05T17:35:46.025Z 10a764e4-2b26-4d8c-9056-1e1072b4723f Task
timed out after 3.00 seconds"}"
awsexamplebucket1,image_17398,,failed,200,PermanentFailure,"Lambda returned function error:
{""errorMessage":""}"2019-04-05T17:35:44.661Z 1e306352-4c54-4eba-aee8-4d02f8c0235c Task
timed out after 3.00 seconds"}"
```

Example succeeded tasks report

Succeeded tasks reports contain the following for the *completed* tasks:

- Bucket
- Key
- VersionId
- TaskStatus
- ErrorCode
- HttpStatusCode
- ResultMessage

In the following example, the Lambda function successfully copied the Amazon S3 object to another bucket. The returned Amazon S3 response is passed back to S3 Batch Operations and is then written into the final completion report.

```
awsexamplebucket1,image_17775,,succeeded,200,,"{u'CopySourceVersionId':
'xVR78haVKlRnurYofbTfYr3ufYbktF8h', u'CopyObjectResult': {u'LastModified':
datetime.datetime(2019, 4, 5, 17, 35, 39, tzinfo=tzlocal()), u'ETag':
'"fe66f4390c50f29798f040d7aae72784"'}, 'ResponseMetadata': {'HTTPStatusCode':
200, 'RetryAttempts': 0, 'HostId': 'nXNaClIMxEJzWNmeMNQV2KpjbaCJLn0OGoXWZpuVOFS/
iQYWxb3QtTvzX9SVfx2lA3oTKLwImKw=', 'RequestId': '3ED5852152014362', 'HTTPHeaders':
{'content-length': '234', 'x-amz-id-2': 'nXNaClIMxEJzWNmeMNQV2KpjbaCJLn0OGoXWZpuVOFS/
iQYWxb3QtTvzX9SVfx2lA3oTKLwImKw=', 'x-amz-copy-source-version-id':
```

```
'xVR78haVKlRnurYofbTfYr3ufYbktF8h', 'server': 'AmazonS3', 'x-amz-request-id': '3ED5852152014362', 'date': 'Fri, 05 Apr 2019 17:35:39 GMT', 'content-type': 'application/xml'}}}}"  
awsexamplebucket1,image_17763,,succeeded,200,,"{u'CopySourceVersionId': '6HjOUSim4Wj6BTcbxToXW44pSZ.40pwq', u'CopyObjectResult': {u'LastModified': datetime.datetime(2019, 4, 5, 17, 35, 39, tzinfo=tzlocal()), u'ETag': """fe66f4390c50f29798f040d7aae72784"""}, 'ResponseMetadata': {'HTTPStatusCode': 200, 'RetryAttempts': 0, 'HostId': 'GICZNYr8LHd/Thyk6beTRP96IGZk2sYxujLe13TuuLpq6U2RD3we0Yoluuidm1PRvkMwnEW1aFc='}, 'RequestId': '1BC9F5B1B95D7000', 'HTTPHeaders': {'content-length': '234', 'x-amz-id-2': 'GICZNYr8LHd/Thyk6beTRP96IGZk2sYxujLe13TuuLpq6U2RD3we0Yoluuidm1PRvkMwnEW1aFc='}, 'x-amz-copy-source-version-id': '6HjOUSim4Wj6BTcbxToXW44pSZ.40pwq', 'server': 'AmazonS3', 'x-amz-request-id': '1BC9F5B1B95D7000', 'date': 'Fri, 05 Apr 2019 17:35:39 GMT', 'content-type': 'application/xml'}}}}"  
awsexamplebucket1,image_17860,,succeeded,200,,"{u'CopySourceVersionId': 'm.MDD0g_QsUnYZ8TBzVFrp.TmjN8PJyX', u'CopyObjectResult': {u'LastModified': datetime.datetime(2019, 4, 5, 17, 35, 40, tzinfo=tzlocal()), u'ETag': """fe66f4390c50f29798f040d7aae72784"""}, 'ResponseMetadata': {'HTTPStatusCode': 200, 'RetryAttempts': 0, 'HostId': 'F9ooZOgpE5g9sNgBZxjdipHqB4+0DNWgj3qbsir+sKai4fv7rQEcf2fBN1VeeFc2WH45a9ygb2g='}, 'RequestId': '8D9CA56A56813DF3', 'HTTPHeaders': {'content-length': '234', 'x-amz-id-2': 'F9ooZOgpE5g9sNgBZxjdipHqB4+0DNWgj3qbsir+sKai4fv7rQEcf2fBN1VeeFc2WH45a9ygb2g='}, 'x-amz-copy-source-version-id': 'm.MDD0g_QsUnYZ8TBzVFrp.TmjN8PJyX', 'server': 'AmazonS3', 'x-amz-request-id': '8D9CA56A56813DF3', 'date': 'Fri, 05 Apr 2019 17:35:40 GMT', 'content-type': 'application/xml'}}}"
```

Controlling access and labeling jobs using tags

You can label and control access to your S3 Batch Operations jobs by adding *tags*. Tags can be used to identify who is responsible for a Batch Operations job. The presence of job tags can grant or limit a user's ability to cancel a job, activate a job in the confirmation state, or change a job's priority level. You can create jobs with tags attached to them, and you can add tags to jobs after they are created. Each tag is a key-value pair that can be included when you create the job or updated later.

Warning

Job tags should not contain any confidential information or personal data.

Consider the following tagging example: Suppose that you want your Finance department to create a Batch Operations job. You could write an AWS Identity and Access Management (IAM) policy that allows a user to invoke `CreateJob`, provided that the job is created with the `Department` tag assigned the value `Finance`. Furthermore, you could attach that policy to all users who are members of the Finance department.

Continuing with this example, you could write a policy that allows a user to update the priority of any job that has the desired tags, or cancel any job that has those tags. For more information, see [the section called "Controlling permissions" \(p. 937\)](#).

You can add tags to new S3 Batch Operations jobs when you create them, or you can add them to existing jobs.

Note the following tag restrictions:

- You can associate up to 50 tags with a job as long as they have unique tag keys.
- A tag key can be up to 128 Unicode characters in length, and tag values can be up to 256 Unicode characters in length.
- The key and values are case sensitive.

For more information about tag restrictions, see [User-Defined Tag Restrictions](#) in the *AWS Billing and Cost Management User Guide*.

API operations related to S3 Batch Operations job tagging

Amazon S3 supports the following API operations that are specific to S3 Batch Operations job tagging:

- [GetJobTagging](#) — Returns the tag set associated with a Batch Operations job.
- [PutJobTagging](#) — Replaces the tag set associated with a job. There are two distinct scenarios for S3 Batch Operations job tag management using this API action:
 - Job has no tags — You can add a set of tags to a job (the job has no prior tags).
 - Job has a set of existing tags — To modify the existing tag set, you can either replace the existing tag set entirely, or make changes within the existing tag set by retrieving the existing tag set using [GetJobTagging](#), modify that tag set, and use this API action to replace the tag set with the one you have modified.

Note

If you send this request with an empty tag set, S3 Batch Operations deletes the existing tag set on the object. If you use this method, you are charged for a Tier 1 Request (PUT). For more information, see [Amazon S3 pricing](#).

To delete existing tags for your Batch Operations job, the `DeleteJobTagging` action is preferred because it achieves the same result without incurring charges.

- [DeleteJobTagging](#) — Deletes the tag set associated with a Batch Operations job.

Creating a Batch Operations job with job tags used for labeling

You can label and control access to your S3 Batch Operations jobs by adding *tags*. Tags can be used to identify who is responsible for a Batch Operations job. You can create jobs with tags attached to them, and you can add tags to jobs after they are created. For more information, see the section called "Using tags" (p. 932).

Using the AWS CLI

The following AWS CLI example creates an S3 Batch Operations `S3PutObjectCopy` job using job tags as labels for the job.

1. Select the action or `OPERATION` that you want the Batch Operations job to perform, and choose your `TargetResource`.

```
read -d '' OPERATION <<EOF
{
    "S3PutObjectCopy": {
        "TargetResource": "arn:aws:s3:::destination-bucket"
    }
}
EOF
```

2. Identify the job `TAGS` that you want for the job. In this case, you apply two tags, `department` and `FiscalYear`, with the values `Marketing` and `2020` respectively.

```
read -d '' TAGS <<EOF
[
    {
        "Key": "department",
        "Value": "Marketing"
    },
    {
        "Key": "FiscalYear",
        "Value": "2020"
    }
]
EOF
```

```
]  
EOF
```

3. Specify the **MANIFEST** for the Batch Operations job.

```
read -d '' MANIFEST <<EOF
{
  "Spec": {
    "Format": "EXAMPLE_S3BatchOperations_CSV_20180820",
    "Fields": [
      "Bucket",
      "Key"
    ],
    "Location": {
      "ObjectArn": "arn:aws:s3:::example-bucket/example_manifest.csv",
      "ETag": "example-5dc7a8bfb90808fc5d546218"
    }
  }
}
EOF
```

4. Configure the **REPORT** for the Batch Operations job.

```
read -d '' REPORT <<EOF
{
  "Bucket": "arn:aws:s3:::example-report-bucket",
  "Format": "Example_Report_CSV_20180820",
  "Enabled": true,
  "Prefix": "reports/copy-with-replace-metadata",
  "ReportScope": "AllTasks"
}
EOF
```

5. Run the **create-job** action to create your Batch Operations job with inputs set in the preceding steps.

```
aws \
  s3control create-job \
  --account-id 123456789012 \
  --manifest "${MANIFEST//$/\n}" \
  --operation "${OPERATION//$/\n/}" \
  --report "${REPORT//$/\n}" \
  --priority 10 \
  --role-arn arn:aws:iam::123456789012:role/batch-operations-role \
  --tags "${TAGS//$/\n/}" \
  --client-request-token "$(uuidgen)" \
  --region us-west-2 \
  --description "Copy with Replace Metadata";
```

Using the AWS SDK for Java

Example

The following example creates an S3 Batch Operations job with tags using the AWS SDK for Java.

```
public String createJob(final AWSS3ControlClient awss3ControlClient) {
    final String manifestObjectArn = "arn:aws:s3:::example-manifest-bucket/
manifests/10_manifest.csv";
    final String manifestObjectVersionId = "example-5dc7a8bfb90808fc5d546218";

    final JobManifestLocation manifestLocation = new JobManifestLocation()
```

```

        .withObjectArn(manifestObjectArn)
        .withETag(manifestObjectVersionId);

    final JobManifestSpec manifestSpec =
        new
    JobManifestSpec().withFormat(JobManifestFormat.S3InventoryReport_CSV_20161130);

    final JobManifest manifestToPublicApi = new JobManifest()
        .withLocation(manifestLocation)
        .withSpec(manifestSpec);

    final String jobReportBucketArn = "arn:aws:s3:::example-report-bucket";
    final String jobReportPrefix = "example-job-reports";

    final JobReport jobReport = new JobReport()
        .withEnabled(true)
        .withReportScope(JobReportScope.AllTasks)
        .withBucket(jobReportBucketArn)
        .withPrefix(jobReportPrefix)
        .withFormat(JobReportFormat.Report_CSV_20180820);

    final String lambdaFunctionArn = "arn:aws:lambda:us-west-2:123456789012:function:example-function";

    final JobOperation jobOperation = new JobOperation()
        .withLambdaInvoke(new
    LambdaInvokeOperation().withFunctionArn(lambdaFunctionArn));

    final S3Tag departmentTag = new S3Tag().withKey("department").withValue("Marketing");
    final S3Tag fiscalYearTag = new S3Tag().withKey("FiscalYear").withValue("2020");

    final String roleArn = "arn:aws:iam::123456789012:role/example-batch-operations-role";
    final Boolean requiresConfirmation = true;
    final int priority = 10;

    final CreateJobRequest request = new CreateJobRequest()
        .withAccountId("123456789012")
        .withDescription("Test lambda job")
        .withManifest(manifestToPublicApi)
        .withOperation(jobOperation)
        .withPriority(priority)
        .withRoleArn(roleArn)
        .withReport(jobReport)
        .withTags(departmentTag, fiscalYearTag)
        .withConfirmationRequired(requiresConfirmation);

    final CreateJobResult result = awss3ControlClient.createJob(request);

    return result.getJobId();
}

```

Deleting the tags from an S3 Batch Operations job

You can use these examples to delete the tags from a Batch Operations job.

Using the AWS CLI

The following example deletes the tags from a Batch Operations job using the AWS CLI.

```

aws \
    s3control delete-job-tagging \
    --account-id 123456789012 \
    --job-id Example-e25a-4ed2-8bee-7f8ed7fc2f1c \

```

```
--region us-east-1;
```

Delete the job tags of a Batch Operations job

Example

The following example deletes the tags of an S3 Batch Operations job using the AWS SDK for Java.

```
public void deleteJobTagging(final AWSS3ControlClient awss3ControlClient,
                             final String jobId) {
    final DeleteJobTaggingRequest deleteJobTaggingRequest = new DeleteJobTaggingRequest()
        .withJobId(jobId);

    final DeleteJobTaggingResult deleteJobTaggingResult =
        awss3ControlClient.deleteJobTagging(deleteJobTaggingRequest);
}
```

Putting job tags for an existing S3 Batch Operations job

You can use [PutJobTagging](#) to add job tags to your existing S3 Batch Operations jobs. For more information, see the following examples.

Using the AWS CLI

The following is an example of using `s3control put-job-tagging` to add job tags to your S3 Batch Operations job using the AWS CLI.

Note

If you send this request with an empty tag set, S3 Batch Operations deletes the existing tag set on the object. Also, if you use this method, you are charged for a Tier 1 Request (PUT). For more information, see [Amazon S3 pricing](#).

To delete existing tags for your Batch Operations job, the `DeleteJobTagging` action is preferred because it achieves the same result without incurring charges.

1. Identify the job TAGS that you want for the job. In this case, you apply two tags, `department` and `FiscalYear`, with the values `Marketing` and `2020` respectively.

```
read -d '' TAGS <<EOF
[
  {
    "Key": "department",
    "Value": "Marketing"
  },
  {
    "Key": "FiscalYear",
    "Value": "2020"
  }
]
EOF
```

2. Run the `put-job-tagging` action with the required parameters.

```
aws \
  s3control put-job-tagging \
  --account-id 123456789012 \
  --tags "${TAGS//$/\n}" \
  --job-id Example-e25a-4ed2-8bee-7f8ed7fc2f1c \
  --region us-east-1;
```

Using the AWS SDK for Java

Example

The following example puts the tags of an S3 Batch Operations job using the AWS SDK for Java.

```
public void putJobTagging(final AWSS3ControlClient awss3ControlClient,
                           final String jobId) {
    final S3Tag departmentTag = new S3Tag().withKey("department").withValue("Marketing");
    final S3Tag fiscalYearTag = new S3Tag().withKey("FiscalYear").withValue("2020");

    final PutJobTaggingRequest putJobTaggingRequest = new PutJobTaggingRequest()
        .withJobId(jobId)
        .withTags(departmentTag, fiscalYearTag);

    final PutJobTaggingResult putJobTaggingResult =
        awss3ControlClient.putJobTagging(putJobTaggingRequest);
}
```

Getting the tags of a S3 Batch Operations job

You can use `GetJobTagging` to return the tags of an S3 Batch Operations job. For more information, see the following examples.

Using the AWS CLI

The following example gets the tags of a Batch Operations job using the AWS CLI.

```
aws \
  s3control get-job-tagging \
  --account-id 123456789012 \
  --job-id Example-e25a-4ed2-8bee-7f8ed7fc2f1c \
  --region us-east-1;
```

Using the AWS SDK for Java

Example

The following example gets the tags of an S3 Batch Operations job using the AWS SDK for Java.

```
public List<S3Tag> getJobTagging(final AWSS3ControlClient awss3ControlClient,
                                   final String jobId) {
    final GetJobTaggingRequest getJobTaggingRequest = new GetJobTaggingRequest()
        .withJobId(jobId);

    final GetJobTaggingResult getJobTaggingResult =
        awss3ControlClient.getJobTagging(getJobTaggingRequest);

    final List<S3Tag> tags = getJobTaggingResult.getTags();

    return tags;
}
```

Controlling permissions for S3 Batch Operations using job tags

To help you manage your S3 Batch Operations jobs, you can add *job tags*. With job tags, you can control access to your Batch Operations jobs and enforce that tags be applied when any job is created.

You can apply up to 50 job tags to each Batch Operations job. This allows you to set very granular policies restricting the set of users that can edit the job. Job tags can grant or limit a user's ability to

cancel a job, activate a job in the confirmation state, or change a job's priority level. In addition, you can enforce that tags be applied to all new jobs, and specify the allowed key-value pairs for the tags. You can express all of these conditions using the same [IAM policy language](#). For more information, see [Actions, resources, and condition keys for Amazon S3 \(p. 431\)](#).

The following example shows how you can use S3 Batch Operations job tags to grant users permission to create and edit only the jobs that are run within a specific *department* (for example, the *Finance* or *Compliance* department). You can also assign jobs based on the stage of *development* that they are related to, such as *QA* or *Production*.

In this example, you use S3 Batch Operations job tags in AWS Identity and Access Management (IAM) policies to grant users permission to create and edit only the jobs being run within their department. You assign jobs based on the stage of development that they are related to, such as *QA* or *Production*.

This example uses the following departments, with each using Batch Operations in different ways:

- Finance
- Compliance
- Business Intelligence
- Engineering

Topics

- [Controlling access by assigning tags to users and resources \(p. 938\)](#)
- [Tagging Batch Operations jobs by stage and enforcing limits on job priority \(p. 939\)](#)

Controlling access by assigning tags to users and resources

In this scenario, the administrators are using [attribute-based access control \(ABAC\)](#). ABAC is an IAM authorization strategy that defines permissions by attaching tags to both IAM users and AWS resources.

Users and jobs are assigned one of the following department tags:

Key : Value

- department : Finance
- department : Compliance
- department : BusinessIntelligence
- department : Engineering

Note

Job tag keys and values are case sensitive.

Using the ABAC access control strategy, you grant a user in the Finance department permission to create and manage S3 Batch Operations jobs within their department by associating the tag `department=Finance` with their IAM user.

Furthermore, you can attach a managed policy to the IAM user that allows any user in their company to create or modify S3 Batch Operations jobs within their respective departments.

The policy in this example includes three policy statements:

- The first statement in the policy allows the user to create a Batch Operations job provided that the job creation request includes a job tag that matches their respective department. This is expressed using the `"${aws:PrincipalTag/department}"` syntax, which is replaced by the IAM user's department

tag at policy evaluation time. The condition is satisfied when the value provided for the department tag in the request ("aws:RequestTag/department") matches the user's department.

- The second statement in the policy allows users to change the priority of jobs or update a job's status provided that the job the user is updating matches the user's department.
- The third statement allows a user to update a Batch Operations job's tags at any time via a PutJobTagging request as long as (1) their department tag is preserved and (2) the job they're updating is within their department.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": "s3:CreateJob",
            "Resource": "*",
            "Condition": {
                "StringEquals": {
                    "aws:RequestTag/department": "${aws:PrincipalTag/department}"
                }
            }
        },
        {
            "Effect": "Allow",
            "Action": [
                "s3:UpdateJobPriority",
                "s3:UpdateJobStatus"
            ],
            "Resource": "*",
            "Condition": {
                "StringEquals": {
                    "aws:ResourceTag/department": "${aws:PrincipalTag/department}"
                }
            }
        },
        {
            "Effect": "Allow",
            "Action": "s3:PutJobTagging",
            "Resource": "*",
            "Condition": {
                "StringEquals": {
                    "aws:RequestTag/department": "${aws:PrincipalTag/department}",
                    "aws:ResourceTag/department": "${aws:PrincipalTag/department}"
                }
            }
        }
    ]
}
```

Tagging Batch Operations jobs by stage and enforcing limits on job priority

All S3 Batch Operations jobs have a numeric priority, which Amazon S3 uses to decide in what order to run the jobs. For this example, you restrict the maximum priority that most users can assign to jobs, with higher priority ranges reserved for a limited set of privileged users, as follows:

- QA stage priority range (low): 1-100
- Production stage priority range (high): 1-300

To do this, introduce a new tag set representing the *stage* of the job:

Key : Value

- stage : QA
- stage : Production

Creating and updating low-priority jobs within a department

This policy introduces two new restrictions on S3 Batch Operations job creation and update, in addition to the department-based restriction:

- It allows users to create or update jobs in their department with a new condition that requires the job to include the tag `stage=QA`.
- It allows users to create or update a job's priority up to a new maximum priority of 100.

```
{  
    "Version": "2012-10-17",  
    "Statement": [  
        {  
            "Effect": "Allow",  
            "Action": "s3:CreateJob",  
            "Resource": "*",  
            "Condition": {  
                "StringEquals": {  
                    "aws:RequestTag/department": "${aws:PrincipalTag/department}",  
                    "aws:RequestTag/stage": "QA"  
                },  
                "NumericLessThanEquals": {  
                    "s3:RequestJobPriority": 100  
                }  
            }  
        },  
        {  
            "Effect": "Allow",  
            "Action": [  
                "s3:UpdateJobStatus"  
            ],  
            "Resource": "*",  
            "Condition": {  
                "StringEquals": {  
                    "aws:ResourceTag/department": "${aws:PrincipalTag/department}"  
                }  
            }  
        },  
        {  
            "Effect": "Allow",  
            "Action": "s3:UpdateJobPriority",  
            "Resource": "*",  
            "Condition": {  
                "StringEquals": {  
                    "aws:ResourceTag/department": "${aws:PrincipalTag/department}",  
                    "aws:ResourceTag/stage": "QA"  
                },  
                "NumericLessThanEquals": {  
                    "s3:RequestJobPriority": 100  
                }  
            }  
        },  
        {
```

```
        "Effect": "Allow",
        "Action": "s3:PutJobTagging",
        "Resource": "*",
        "Condition": {
            "StringEquals": {
                "aws:RequestTag/department" : "${aws:PrincipalTag/department}",
                "aws:ResourceTag/department": "${aws:PrincipalTag/department}",
                "aws:RequestTag/stage": "QA",
                "aws:ResourceTag/stage": "QA"
            }
        }
    },
{
    "Effect": "Allow",
    "Action": "s3:GetJobTagging",
    "Resource": "*"
}
]
```

Creating and updating high-priority jobs within a department

A small number of users might require the ability to create high priority jobs in either *QA* or *Production*. To support this need, you create a managed policy that's adapted from the low-priority policy in the previous section.

This policy does the following:

- Allows users to create or update jobs in their department with either the tag `stage=QA` or `stage=Production`.
- Allows users to create or update a job's priority up to a maximum of 300.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": "s3:CreateJob",
            "Resource": "*",
            "Condition": {
                "ForAnyValue:StringEquals": {
                    "aws:RequestTag/stage": [
                        "QA",
                        "Production"
                    ]
                },
                "StringEquals": {
                    "aws:RequestTag/department": "${aws:PrincipalTag/department}"
                },
                "NumericLessThanEquals": {
                    "s3:RequestJobPriority": 300
                }
            }
        },
        {
            "Effect": "Allow",
            "Action": [
                "s3:UpdateJobStatus"
            ],
            "Resource": "*",
        }
    ]
}
```

```
        "Condition": {
            "StringEquals": {
                "aws:ResourceTag/department": "${aws:PrincipalTag/department}"
            }
        },
        {
            "Effect": "Allow",
            "Action": "s3:UpdateJobPriority",
            "Resource": "*",
            "Condition": {
                "ForAnyValue:StringEquals": [
                    "aws:ResourceTag/stage": [
                        "QA",
                        "Production"
                    ]
                ],
                "StringEquals": {
                    "aws:ResourceTag/department": "${aws:PrincipalTag/department}"
                },
                "NumericLessThanEquals": {
                    "s3:RequestJobPriority": 300
                }
            }
        },
        {
            "Effect": "Allow",
            "Action": "s3:PutJobTagging",
            "Resource": "*",
            "Condition": {
                "StringEquals": {
                    "aws:RequestTag/department": "${aws:PrincipalTag/department}",
                    "aws:ResourceTag/department": "${aws:PrincipalTag/department}"
                },
                "ForAnyValue:StringEquals": [
                    "aws:RequestTag/stage": [
                        "QA",
                        "Production"
                    ],
                    "aws:ResourceTag/stage": [
                        "QA",
                        "Production"
                    ]
                ]
            }
        }
    ]
}
```

Managing S3 Object Lock using S3 Batch Operations

With S3 Object Lock, you can place a legal hold on an object version. Like setting a retention period, a legal hold prevents an object version from being overwritten or deleted. However, a legal hold doesn't have an associated retention period and remains in effect until removed. For more information, see [S3 Object Lock legal hold \(p. 919\)](#).

For information about using S3 Batch Operations with Object Lock to add legal holds to *many* Amazon S3 objects at once, see the following sections.

Topics

- [Enabling S3 Object Lock using S3 Batch Operations \(p. 943\)](#)
- [Setting Object Lock retention using Batch Operations \(p. 946\)](#)
- [Using S3 Batch Operations with S3 Object Lock retention compliance mode \(p. 947\)](#)
- [Use S3 Batch Operations with S3 Object Lock retention governance mode \(p. 951\)](#)
- [Using S3 Batch Operations to turn off S3 Object Lock legal hold \(p. 955\)](#)

Enabling S3 Object Lock using S3 Batch Operations

You can use S3 Batch Operations with S3 Object Lock to manage retention or enable a legal hold for many Amazon S3 objects at once. You specify the list of target objects in your manifest and submit it to Batch Operations for completion. For more information, see [the section called "Object Lock retention" \(p. 918\)](#) and [the section called "Object Lock legal hold" \(p. 919\)](#).

The following examples show how to create an IAM role with S3 Batch Operations permissions and update the role permissions to create jobs that enable Object Lock. In the examples, replace any variable values with those that suit your needs. You must also have a CSV manifest identifying the objects for your S3 Batch Operations job. For more information, see [the section called "Specifying a manifest" \(p. 890\)](#).

Using the AWS CLI

1. Create an IAM role and assign S3 Batch Operations permissions to run.

This step is required for all S3 Batch Operations jobs.

```
export AWS_PROFILE='aws-user'

read -d '' bops_trust_policy <<EOF
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Principal": {
                "Service": [
                    "batchoperations.s3.amazonaws.com"
                ]
            },
            "Action": "sts:AssumeRole"
        }
    ]
}
EOF
aws iam create-role --role-name bops-objectlock --assume-role-policy-document
"${bops_trust_policy}"
```

2. Set up S3 Batch Operations with S3 Object Lock to run.

In this step, you allow the role to do the following:

- a. Run Object Lock on the S3 bucket that contains the target objects that you want Batch Operations to run on.
- b. Read the S3 bucket where the manifest CSV file and the objects are located.
- c. Write the results of the S3 Batch Operations job to the reporting bucket.

```
read -d '' bops_permissions <<EOF
{
```

```

"Version": "2012-10-17",
"Statement": [
    {
        "Effect": "Allow",
        "Action": "s3:GetBucketObjectLockConfiguration",
        "Resource": [
            "arn:aws:s3:::{ManifestBucket}"
        ]
    },
    {
        "Effect": "Allow",
        "Action": [
            "s3:GetObject",
            "s3:GetObjectVersion",
            "s3:GetBucketLocation"
        ],
        "Resource": [
            "arn:aws:s3:::{ManifestBucket}/*"
        ]
    },
    {
        "Effect": "Allow",
        "Action": [
            "s3:PutObject",
            "s3:GetBucketLocation"
        ],
        "Resource": [
            "arn:aws:s3:::{ReportBucket}/*"
        ]
    }
]
}
EOF

aws iam put-role-policy --role-name bops-objectlock --policy-name object-lock-permissions --policy-document "${bops_permissions}"

```

Using the AWS SDK for Java

The following examples show how to create an IAM role with S3 Batch Operations permissions, and update the role permissions to create jobs that enable object lock using the AWS SDK for Java. In the code, replace any variable values with those that suit your needs. You must also have a CSV manifest identifying the objects for your S3 Batch Operations job. For more information, see [the section called "Specifying a manifest" \(p. 890\)](#).

You perform the following steps:

1. Create an IAM role and assign S3 Batch Operations permissions to run. This step is required for all S3 Batch Operations jobs.
2. Set up S3 Batch Operations with S3 Object Lock to run.

You allow the role to do the following:

1. Run Object Lock on the S3 bucket that contains the target objects that you want Batch Operations to run on.
2. Read the S3 bucket where the manifest CSV file and the objects are located.
3. Write the results of the S3 Batch Operations job to the reporting bucket.

```

public void createObjectLockRole() {
    final String roleName = "bops-object-lock";

```

```

final String trustPolicy = "{" +
    "  \"Version\": \"2012-10-17\", " +
    "  \"Statement\": [ " +
    "    { " +
    "      \"Effect\": \"Allow\", " +
    "      \"Principal\": { " +
    "        \"Service\": [ " +
    "          \"batchoperations.s3.amazonaws.com\" +
    "        ]" +
    "      }, " +
    "      \"Action\": \"sts:AssumeRole\" " +
    "    } " +
    "  ]" +
"}";

final String bopsPermissions = "{" +
    "  \"Version\": \"2012-10-17\", " +
    "  \"Statement\": [ " +
    "    { " +
    "      \"Effect\": \"Allow\", " +
    "      \"Action\": \"s3:GetBucketObjectLockConfiguration\", " +
    "      \"Resource\": [ " +
    "        \"arn:aws:s3:::ManifestBucket\" +
    "      ]" +
    "    }, " +
    "    { " +
    "      \"Effect\": \"Allow\", " +
    "      \"Action\": [ " +
    "        \"s3:GetObject\", " +
    "        \"s3:GetObjectVersion\", " +
    "        \"s3:GetBucketLocation\" +
    "      ], " +
    "      \"Resource\": [ " +
    "        \"arn:aws:s3:::ManifestBucket/*\" +
    "      ]" +
    "    }, " +
    "    { " +
    "      \"Effect\": \"Allow\", " +
    "      \"Action\": [ " +
    "        \"s3:PutObject\", " +
    "        \"s3:GetBucketLocation\" +
    "      ], " +
    "      \"Resource\": [ " +
    "        \"arn:aws:s3:::ReportBucket/*\" +
    "      ]" +
    "    }" +
    "  ]" +
"}";

final AmazonIdentityManagement iam =
    AmazonIdentityManagementClientBuilder.defaultClient();

final CreateRoleRequest createRoleRequest = new CreateRoleRequest()
    .withAssumeRolePolicyDocument(bopsPermissions)
    .withRoleName(roleName);

final CreateRoleResult createRoleResult = iam.createRole(createRoleRequest);

final PutRolePolicyRequest putRolePolicyRequest = new PutRolePolicyRequest()
    .withPolicyDocument(bopsPermissions)
    .withPolicyName("bops-permissions")
    .withRoleName(roleName);

final PutRolePolicyResult putRolePolicyResult =
    iam.putRolePolicy(putRolePolicyRequest);

```

}

Setting Object Lock retention using Batch Operations

The following example allows the rule to set S3 Object Lock retention for your objects in the manifest bucket.

You update the role to include `s3:PutObjectRetention` permissions so that you can run Object Lock retention on the objects in your bucket.

Using the AWS CLI

```
export AWS_PROFILE='aws-user'

read -d '' retention_permissions <<EOF
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "s3:PutObjectRetention"
            ],
            "Resource": [
                "arn:aws:s3:::{ManifestBucket}/*"
            ]
        }
    ]
}
EOF

aws iam put-role-policy --role-name bops-objectlock --policy-name retention-permissions --policy-document "${retention_permissions}"
```

Using the AWS SDK for Java

```
public void allowPutObjectRetention() {
    final String roleName = "bops-object-lock";

    final String retentionPermissions = "{" +
        "    \"Version\": \"2012-10-17\", " +
        "    \"Statement\": [ " +
        "        { " +
        "            \"Effect\": \"Allow\", " +
        "            \"Action\": [ " +
        "                \"s3:PutObjectRetention\" " +
        "            ], " +
        "            \"Resource\": [ " +
        "                \"arn:aws:s3:::ManifestBucket*\" " +
        "            ] " +
        "        } " +
        "    ] " +
    "}";

    final AmazonIdentityManagement iam =
        AmazonIdentityManagementClientBuilder.defaultClient();

    final PutRolePolicyRequest putRolePolicyRequest = new PutRolePolicyRequest()
        .withPolicyDocument(retentionPermissions)
        .withPolicyName("retention-permissions")
        .withRoleName(roleName);
```

```
    final PutRolePolicyResult putRolePolicyResult =
    iam.putRolePolicy(putRolePolicyRequest);
}
```

Using S3 Batch Operations with S3 Object Lock retention compliance mode

The following example builds on the previous examples of creating a trust policy and setting S3 Batch Operations and S3 Object Lock configuration permissions on your objects. This example sets the retention mode to COMPLIANCE and the retain until date to January 1, 2020. It creates a job that targets objects in the manifest bucket and reports the results in the reports bucket that you identified.

Using the AWS CLI

Example Set mention compliance across multiple objects

```
export AWS_PROFILE='aws-user'
export AWS_DEFAULT_REGION='us-west-2'
export ACCOUNT_ID=123456789012
export ROLE_ARN='arn:aws:iam::123456789012:role/bops-objectlock'

read -d '' OPERATION <<EOF
{
    "S3PutObjectRetention": {
        "Retention": {
            "RetainUntilDate": "2025-01-01T00:00:00",
            "Mode": "COMPLIANCE"
        }
    }
}
EOF

read -d '' MANIFEST <<EOF
{
    "Spec": {
        "Format": "S3BatchOperations_CSV_20180820",
        "Fields": [
            "Bucket",
            "Key"
        ]
    },
    "Location": {
        "ObjectArn": "arn:aws:s3:::ManifestBucket/compliance-objects-manifest.csv",
        "ETag": "Your-manifest-ETag"
    }
}
EOF

read -d '' REPORT <<EOF
{
    "Bucket": "arn:aws:s3:::ReportBucket",
    "Format": "Report_CSV_20180820",
    "Enabled": true,
    "Prefix": "reports/compliance-objects-bops",
    "ReportScope": "AllTasks"
}
EOF

aws \
    s3control create-job \
    --account-id "${ACCOUNT_ID}" \
    --manifest "${MANIFEST}://${'\n'}" \
```

```
--operation "${OPERATION//$/\n}" \
--report "${REPORT//$/\n}" \
--priority 10 \
--role-arn "${ROLE_ARN}" \
--client-request-token "$(uuidgen)" \
--region "${AWS_DEFAULT_REGION}" \
--description "Set compliance retain-until to 1 Jul 2030";
```

Example Extend the COMPLIANCE mode's retain until date to January 15, 2020

The following example extends the COMPLIANCE mode's retain until date to January 15, 2025.

```
export AWS_PROFILE='aws-user'
export AWS_DEFAULT_REGION='us-west-2'
export ACCOUNT_ID=123456789012
export ROLE_ARN='arn:aws:iam::123456789012:role/bops-objectlock'

read -d '' OPERATION <<EOF
{
    "S3PutObjectRetention": {
        "Retention": {
            "RetainUntilDate": "2025-01-15T00:00:00",
            "Mode": "COMPLIANCE"
        }
    }
}
EOF

read -d '' MANIFEST <<EOF
{
    "Spec": {
        "Format": "S3BatchOperations_CSV_20180820",
        "Fields": [
            "Bucket",
            "Key"
        ],
        "Location": {
            "ObjectArn": "arn:aws:s3:::ManifestBucket/compliance-objects-manifest.csv",
            "ETag": "Your-manifest-ETag"
        }
    }
}
EOF

read -d '' REPORT <<EOF
{
    "Bucket": "arn:aws:s3:::ReportBucket",
    "Format": "Report_CSV_20180820",
    "Enabled": true,
    "Prefix": "reports/compliance-objects-bops",
    "ReportScope": "AllTasks"
}
EOF

aws \
    s3control create-job \
    --account-id "${ACCOUNT_ID}" \
    --manifest "${MANIFEST//$/\n}" \
    --operation "${OPERATION//$/\n}" \
    --report "${REPORT//$/\n}" \
    --priority 10 \
    --role-arn "${ROLE_ARN}" \
    --client-request-token "$(uuidgen)" \
    --region "${AWS_DEFAULT_REGION}" \
```

```
--description "Extend compliance retention to 15 Jan 2020";
```

Using the AWS SDK for Java

Example Set the retention mode to COMPLIANCE and the retain until date to January 1, 2020.

```
public String createComplianceRetentionJob(final AWSS3ControlClient awss3ControlClient)
    throws ParseException {
    final String manifestObjectArn = "arn:aws:s3:::ManifestBucket/compliance-objects-
manifest.csv";
    final String manifestObjectVersionId = "your-object-version-Id";

    final JobManifestLocation manifestLocation = new JobManifestLocation()
        .withObjectArn(manifestObjectArn)
        .withETag(manifestObjectVersionId);

    final JobManifestSpec manifestSpec =
        new JobManifestSpec()
            .withFormat(JobManifestFormat.S3BatchOperations_CSV_20180820)
            .withFields("Bucket", "Key");

    final JobManifest manifestToPublicApi = new JobManifest()
        .withLocation(manifestLocation)
        .withSpec(manifestSpec);

    final String jobReportBucketArn = "arn:aws:s3:::ReportBucket";
    final String jobReportPrefix = "reports/compliance-objects-bops";

    final JobReport jobReport = new JobReport()
        .withEnabled(true)
        .withReportScope(JobReportScope.AllTasks)
        .withBucket(jobReportBucketArn)
        .withPrefix(jobReportPrefix)
        .withFormat(JobReportFormat.Report_CSV_20180820);

    final SimpleDateFormat format = new SimpleDateFormat("dd/MM/yyyy");
    final Date janFirst = format.parse("01/01/2020");

    final JobOperation jobOperation = new JobOperation()
        .withS3PutObjectRetention(new S3SetObjectRetentionOperation()
            .withRetention(new S3Retention()
                .withMode(S3ObjectLockRetentionMode.COMPLIANCE)
                .withRetainUntilDate(janFirst)));

    final String roleArn = "arn:aws:iam::123456789012:role/bops-object-lock";
    final Boolean requiresConfirmation = true;
    final int priority = 10;

    final CreateJobRequest request = new CreateJobRequest()
        .withAccountId("123456789012")
        .withDescription("Set compliance retain-until to 1 Jan 2020")
        .withManifest(manifestToPublicApi)
        .withOperation(jobOperation)
        .withPriority(priority)
        .withRoleArn(roleArn)
        .withReport(jobReport)
        .withConfirmationRequired(requiresConfirmation);

    final CreateJobResult result = awss3ControlClient.createJob(request);

    return result.getJobId();
}
```

Example Extending the COMPLIANCE mode's retain until date

The following example extends the COMPLIANCE mode's retain until date to January 15, 2020.

```
public String createExtendComplianceRetentionJob(final AWSS3ControlClient awss3ControlClient) throws ParseException {
    final String manifestObjectArn = "arn:aws:s3:::ManifestBucket/compliance-objects-manifest.csv";
    final String manifestObjectVersionId = "15ad5ba069e6bbc465c77bf83d541385";

    final JobManifestLocation manifestLocation = new JobManifestLocation()
        .withObjectArn(manifestObjectArn)
        .withETag(manifestObjectVersionId);

    final JobManifestSpec manifestSpec =
        new JobManifestSpec()
            .withFormat(JobManifestFormat.S3BatchOperations_CSV_20180820)
            .withFields("Bucket", "Key");

    final JobManifest manifestToPublicApi = new JobManifest()
        .withLocation(manifestLocation)
        .withSpec(manifestSpec);

    final String jobReportBucketArn = "arn:aws:s3:::ReportBucket";
    final String jobReportPrefix = "reports/compliance-objects-bops";

    final JobReport jobReport = new JobReport()
        .withEnabled(true)
        .withReportScope(JobReportScope.AllTasks)
        .withBucket(jobReportBucketArn)
        .withPrefix(jobReportPrefix)
        .withFormat(JobReportFormat.Report_CSV_20180820);

    final SimpleDateFormat format = new SimpleDateFormat("dd/MM/yyyy");
    final Date jan15th = format.parse("15/01/2020");

    final JobOperation jobOperation = new JobOperation()
        .withS3PutObjectRetention(new S3SetObjectRetentionOperation()
            .withRetention(new S3Retention()
                .withMode(S3ObjectLockRetentionMode.COMPLIANCE)
                .withRetainUntilDate(jan15th)));
}

final String roleArn = "arn:aws:iam::123456789012:role/bops-object-lock";
final Boolean requiresConfirmation = true;
final int priority = 10;

final CreateJobRequest request = new CreateJobRequest()
    .withAccountId("123456789012")
    .withDescription("Extend compliance retention to 15 Jan 2020")
    .withManifest(manifestToPublicApi)
    .withOperation(jobOperation)
    .withPriority(priority)
    .withRoleArn(roleArn)
    .withReport(jobReport)
    .withConfirmationRequired(requiresConfirmation);

final CreateJobResult result = awss3ControlClient.createJob(request);

return result.getJobId();
}
```

Use S3 Batch Operations with S3 Object Lock retention governance mode

The following example builds on the previous example of creating a trust policy, and setting S3 Batch Operations and S3 Object Lock configuration permissions. It shows how to apply S3 Object Lock retention governance with the `retain until` date of January 30, 2025, across multiple objects. It creates a Batch Operations job that uses the manifest bucket and reports the results in the reports bucket.

Using the AWS CLI

Example Apply S3 Object Lock retention governance across multiple objects with the retain until date of January 30, 2020

```
export AWS_PROFILE='aws-user'
export AWS_DEFAULT_REGION='us-west-2'
export ACCOUNT_ID=123456789012
export ROLE_ARN='arn:aws:iam::123456789012:role/bops-objectlock'

read -d '' OPERATION <<EOF
{
    "S3PutObjectRetention": {
        "Retention": {
            "RetainUntilDate": "2025-01-30T00:00:00",
            "Mode": "GOVERNANCE"
        }
    }
}
EOF

read -d '' MANIFEST <<EOF
{
    "Spec": {
        "Format": "S3BatchOperations_CSV_20180820",
        "Fields": [
            "Bucket",
            "Key"
        ],
        "Location": {
            "ObjectArn": "arn:aws:s3:::ManifestBucket/governance-objects-manifest.csv",
            "ETag": "Your-manifest-ETag"
        }
    }
}
EOF

read -d '' REPORT <<EOF
{
    "Bucket": "arn:aws:s3:::ReportBucketT",
    "Format": "Report_CSV_20180820",
    "Enabled": true,
    "Prefix": "reports/governance-objects",
    "ReportScope": "AllTasks"
}
EOF

aws \
    s3control create-job \
    --account-id "${ACCOUNT_ID}" \
    --manifest "${MANIFEST//$/\n}" \
    --operation "${OPERATION//$/\n/}" \
    --report "${REPORT//$/\n}" \
    --priority 10 \
```

```
--role-arn "${ROLE_ARN}" \
--client-request-token "$(uuidgen)" \
--region "${AWS_DEFAULT_REGION}" \
--description "Put governance retention";
```

Example Bypass retention governance across multiple objects

The following example builds on the previous example of creating a trust policy, and setting S3 Batch Operations and S3 Object Lock configuration permissions. It shows how to bypass retention governance across multiple objects and creates a Batch Operations job that uses the manifest bucket and reports the results in the reports bucket.

```
export AWS_PROFILE='aws-user'

read -d '' bypass_governance_permissions <<EOF
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "s3:BypassGovernanceRetention"
            ],
            "Resource": [
                "arn:aws:s3:::${ManifestBucket}/*"
            ]
        }
    ]
}
EOF

aws iam put-role-policy --role-name bops-objectlock --policy-name bypass-governance-
permissions --policy-document "${bypass_governance_permissions}"

export AWS_PROFILE='aws-user'
export AWS_DEFAULT_REGION='us-west-2'
export ACCOUNT_ID=123456789012
export ROLE_ARN='arn:aws:iam::123456789012:role/bops-objectlock'

read -d '' OPERATION <<EOF
{
    "S3PutObjectRetention": {
        "BypassGovernanceRetention": true,
        "Retention": {}
    }
}
EOF

read -d '' MANIFEST <<EOF
{
    "Spec": {
        "Format": "S3BatchOperations_CSV_20180820",
        "Fields": [
            "Bucket",
            "Key"
        ],
        "Location": {
            "ObjectArn": "arn:aws:s3:::${ManifestBucket}/governance-objects-manifest.csv",
            "ETag": "Your-manifest-ETag"
        }
    }
}
EOF
```

```

read -d '' REPORT <<EOF
{
    "Bucket": "arn:aws:s3:::$REPORT_BUCKET",
    "Format": "Report_CSV_20180820",
    "Enabled": true,
    "Prefix": "reports/bops-governance",
    "ReportScope": "AllTasks"
}
EOF

aws \
    s3control create-job \
    --account-id "${ACCOUNT_ID}" \
    --manifest "${MANIFEST//$/\n}" \
    --operation "${OPERATION//$/\n/}" \
    --report "${REPORT//$/\n}" \
    --priority 10 \
    --role-arn "${ROLE_ARN}" \
    --client-request-token "$(uuidgen)" \
    --region "${AWS_DEFAULT_REGION}" \
    --description "Remove governance retention";

```

Using the AWS SDK for Java

The following example builds on the previous example of creating a trust policy, and setting S3 Batch Operations and S3 Object Lock configuration permissions. It shows how to apply S3 Object Lock retention governance with the `retain` until date set to January 30, 2020 across multiple objects. It creates a Batch Operations job that uses the manifest bucket and reports the results in the reports bucket.

Example Apply S3 Object Lock retention governance across multiple objects with the retain until date of January 30, 2020

```

public String createGovernanceRetentionJob(final AWSS3ControlClient awss3ControlClient)
    throws ParseException {
    final String manifestObjectArn = "arn:aws:s3:::$ManifestBucket/governance-objects-
manifest.csv";
    final String manifestObjectVersionId = "15ad5ba069e6bbc465c77bf83d541385";

    final JobManifestLocation manifestLocation = new JobManifestLocation()
        .withObjectArn(manifestObjectArn)
        .withETag(manifestObjectVersionId);

    final JobManifestSpec manifestSpec =
        new JobManifestSpec()
            .withFormat(JobManifestFormat.S3BatchOperations_CSV_20180820)
            .withFields("Bucket", "Key");

    final JobManifest manifestToPublicApi = new JobManifest()
        .withLocation(manifestLocation)
        .withSpec(manifestSpec);

    final String jobReportBucketArn = "arn:aws:s3:::$ReportBucket";
    final String jobReportPrefix = "reports/governance-objects";

    final JobReport jobReport = new JobReport()
        .withEnabled(true)
        .withReportScope(JobReportScope.AllTasks)
        .withBucket(jobReportBucketArn)
        .withPrefix(jobReportPrefix)
        .withFormat(JobReportFormat.Report_CSV_20180820);

    final SimpleDateFormat format = new SimpleDateFormat("dd/MM/yyyy");

```

```

final Date jan30th = format.parse("30/01/2020");

final JobOperation jobOperation = new JobOperation()
    .withS3PutObjectRetention(new S3SetObjectRetentionOperation()
        .withRetention(new S3Retention()
            .withMode(S3ObjectLockRetentionMode.GOVERNANCE)
            .withRetainUntilDate(jan30th)));

final String roleArn = "arn:aws:iam::123456789012:role/bops-object-lock";
final Boolean requiresConfirmation = true;
final int priority = 10;

final CreateJobRequest request = new CreateJobRequest()
    .withAccountId("123456789012")
    .withDescription("Put governance retention")
    .withManifest(manifestToPublicApi)
    .withOperation(jobOperation)
    .withPriority(priority)
    .withRoleArn(roleArn)
    .withReport(jobReport)
    .withConfirmationRequired(requiresConfirmation);

final CreateJobResult result = awss3ControlClient.createJob(request);

return result.getJobId();
}

```

Example Bypass retention governance across multiple objects

The following example builds on the previous example of creating a trust policy, and setting S3 Batch Operations and S3 Object Lock configuration permissions. It shows how to bypass retention governance across multiple objects and creates a Batch Operations job that uses the manifest bucket and reports the results in the reports bucket.

```

public void allowBypassGovernance() {
    final String roleName = "bops-object-lock";

    final String bypassGovernancePermissions = "{" +
        "    \"Version\": \"2012-10-17\", " +
        "    \"Statement\": [ " +
        "        { " +
        "            \"Effect\": \"Allow\", " +
        "            \"Action\": [ " +
        "                \"s3:BypassGovernanceRetention\" " +
        "            ], " +
        "            \"Resource\": [ " +
        "                \"arn:aws:s3:::ManifestBucket/*\" " +
        "            ] " +
        "        } " +
        "    ] " +
    "}";
}

final AmazonIdentityManagement iam =
    AmazonIdentityManagementClientBuilder.defaultClient();

final PutRolePolicyRequest putRolePolicyRequest = new PutRolePolicyRequest()
    .withPolicyDocument(bypassGovernancePermissions)
    .withPolicyName("bypass-governance-permissions")
    .withRoleName(roleName);

final PutRolePolicyResult putRolePolicyResult =
    iam.putRolePolicy(putRolePolicyRequest);
}

```

```

public String createRemoveGovernanceRetentionJob(final AWSS3ControlClient
awss3ControlClient) {
    final String manifestObjectArn = "arn:aws:s3:::ManifestBucket/governance-objects-manifest.csv";
    final String manifestObjectVersionId = "15ad5ba069e6bbc465c77bf83d541385";

    final JobManifestLocation manifestLocation = new JobManifestLocation()
        .withObjectArn(manifestObjectArn)
        .withETag(manifestObjectVersionId);

    final JobManifestSpec manifestSpec =
        new JobManifestSpec()
            .withFormat(JobManifestFormat.S3BatchOperations_CSV_20180820)
            .withFields("Bucket", "Key");

    final JobManifest manifestToPublicApi = new JobManifest()
        .withLocation(manifestLocation)
        .withSpec(manifestSpec);

    final String jobReportBucketArn = "arn:aws:s3:::ReportBucket";
    final String jobReportPrefix = "reports/bops-governance";

    final JobReport jobReport = new JobReport()
        .withEnabled(true)
        .withReportScope(JobReportScope.AllTasks)
        .withBucket(jobReportBucketArn)
        .withPrefix(jobReportPrefix)
        .withFormat(JobReportFormat.Report_CSV_20180820);

    final JobOperation jobOperation = new JobOperation()
        .withS3PutObjectRetention(new S3SetObjectRetentionOperation()
            .withRetention(new S3Retention()));

    final String roleArn = "arn:aws:iam::123456789012:role/bops-object-lock";
    final Boolean requiresConfirmation = true;
    final int priority = 10;

    final CreateJobRequest request = new CreateJobRequest()
        .withAccountId("123456789012")
        .withDescription("Remove governance retention")
        .withManifest(manifestToPublicApi)
        .withOperation(jobOperation)
        .withPriority(priority)
        .withRoleArn(roleArn)
        .withReport(jobReport)
        .withConfirmationRequired(requiresConfirmation);

    final CreateJobResult result = awss3ControlClient.createJob(request);

    return result.getJobId();
}

```

Using S3 Batch Operations to turn off S3 Object Lock legal hold

The following example builds on the previous examples of creating a trust policy, and setting S3 Batch Operations and S3 Object Lock configuration permissions. It shows how to disable Object Lock legal hold on objects using Batch Operations.

The example first updates the role to grant `s3:PutObjectLegalHold` permissions, creates a Batch Operations job that turns off (removes) legal hold from the objects identified in the manifest, and then reports on it.

Using the AWS CLI

Example Updates the role to grant s3:PutObjectLegalHold permissions

```
export AWS_PROFILE='aws-user'

read -d '' legal_hold_permissions <<EOF
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "s3:PutObjectLegalHold"
            ],
            "Resource": [
                "arn:aws:s3:::ManifestBucket/*"
            ]
        }
    ]
}

EOF

aws iam put-role-policy --role-name bops-objectlock --policy-name legal-hold-permissions --policy-document "${legal_hold_permissions}"
```

Example Turn off legal hold

The following example turns off legal hold.

```
export AWS_PROFILE='aws-user'
export AWS_DEFAULT_REGION='us-west-2'
export ACCOUNT_ID=123456789012
export ROLE_ARN='arn:aws:iam::123456789012:role/bops-objectlock'

read -d '' OPERATION <<EOF
{
    "S3PutObjectLegalHold": {
        "LegalHold": {
            "Status": "OFF"
        }
    }
}
EOF

read -d '' MANIFEST <<EOF
{
    "Spec": {
        "Format": "S3BatchOperations_CSV_20180820",
        "Fields": [
            "Bucket",
            "Key"
        ]
    },
    "Location": {
        "ObjectArn": "arn:aws:s3:::ManifestBucket/legalhold-object-manifest.csv",
        "ETag": "Your-manifest-ETag"
    }
}
EOF

read -d '' REPORT <<EOF
{
```

```

    "Bucket": "arn:aws:s3:::ReportBucket",
    "Format": "Report_CSV_20180820",
    "Enabled": true,
    "Prefix": "reports/legalhold-objects-bops",
    "ReportScope": "AllTasks"
}
EOF

aws \
    s3control create-job \
    --account-id "${ACCOUNT_ID}" \
    --manifest "${MANIFEST//$/\n}" \
    --operation "${OPERATION//$/\n/}" \
    --report "${REPORT//$/\n}" \
    --priority 10 \
    --role-arn "${ROLE_ARN}" \
    --client-request-token "$(uuidgen)" \
    --region "${AWS_DEFAULT_REGION}" \
    --description "Turn off legal hold";

```

Using the AWS SDK for Java

Example Updates the role to grant s3:PutObjectLegalHold permissions

```

public void allowPutObjectLegalHold() {
    final String roleName = "bops-object-lock";

    final String legalHoldPermissions = "{" +
        "    \"Version\": \"2012-10-17\", " +
        "    \"Statement\": [ " +
        "        { " +
        "            \"Effect\": \"Allow\", " +
        "            \"Action\": [ " +
        "                \"s3:PutObjectLegalHold\" " +
        "            ], " +
        "            \"Resource\": [ " +
        "                \"arn:aws:s3:::ManifestBucket/*\" " +
        "            ] " +
        "        } " +
        "    ] " +
    "}";
}

final AmazonIdentityManagement iam =
    AmazonIdentityManagementClientBuilder.defaultClient();

final PutRolePolicyRequest putRolePolicyRequest = new PutRolePolicyRequest()
    .withPolicyDocument(legalHoldPermissions)
    .withPolicyName("legal-hold-permissions")
    .withRoleName(roleName);

final PutRolePolicyResult putRolePolicyResult =
iam.putRolePolicy(putRolePolicyRequest);
}

```

Example Turn off legal hold

Use the example below if you want to turn off legal hold.

```

public String createLegalHoldOffJob(final AWSS3ControlClient awss3ControlClient) {
    final String manifestObjectArn = "arn:aws:s3:::ManifestBucket/legalhold-object-manifest.csv";
    final String manifestObjectVersionId = "15ad5ba069e6bbc465c77bf83d541385";
}

```

```
final JobManifestLocation manifestLocation = new JobManifestLocation()
    .withObjectArn(manifestObjectArn)
    .withETag(manifestObjectVersionId);

final JobManifestSpec manifestSpec =
    new JobManifestSpec()
        .withFormat(JobManifestFormat.S3BatchOperations_CSV_20180820)
        .withFields("Bucket", "Key");

final JobManifest manifestToPublicApi = new JobManifest()
    .withLocation(manifestLocation)
    .withSpec(manifestSpec);

final String jobReportBucketArn = "arn:aws:s3:::ReportBucket";
final String jobReportPrefix = "reports/legalhold-objects-bops";

final JobReport jobReport = new JobReport()
    .withEnabled(true)
    .withReportScope(JobReportScope.AllTasks)
    .withBucket(jobReportBucketArn)
    .withPrefix(jobReportPrefix)
    .withFormat(JobReportFormat.Report_CSV_20180820);

final JobOperation jobOperation = new JobOperation()
    .withS3PutObjectLegalHold(new S3SetObjectLegalHoldOperation()
        .withLegalHold(new S3ObjectLockLegalHold()
            .withStatus(S3ObjectLockLegalHoldStatus.OFF)));
}

final String roleArn = "arn:aws:iam::123456789012:role/bops-object-lock";
final Boolean requiresConfirmation = true;
final int priority = 10;

final CreateJobRequest request = new CreateJobRequest()
    .withAccountId("123456789012")
    .withDescription("Turn off legal hold")
    .withManifest(manifestToPublicApi)
    .withOperation(jobOperation)
    .withPriority(priority)
    .withRoleArn(roleArn)
    .withReport(jobReport)
    .withConfirmationRequired(requiresConfirmation);

final CreateJobResult result = awss3ControlClient.createJob(request);

return result.getJobId();
}
```

S3 Batch Operations tutorial

The following tutorial presents complete end-to-end procedures for some Batch Operations tasks.

- [Tutorial: Batch-transcoding videos with S3 Batch Operations, AWS Lambda, and AWS Elemental MediaConvert \(p. 64\)](#)

Monitoring Amazon S3

Monitoring is an important part of maintaining the reliability, availability, and performance of Amazon S3 and your AWS solutions. We recommend collecting monitoring data from all of the parts of your AWS solution so that you can more easily debug a multipoint failure if one occurs. Before you start monitoring Amazon S3, create a monitoring plan that includes answers to the following questions:

- What are your monitoring goals?
- What resources will you monitor?
- How often will you monitor these resources?
- What monitoring tools will you use?
- Who will perform the monitoring tasks?
- Who should be notified when something goes wrong?

For more information about logging and monitoring in Amazon S3, see the following topics.

Topics

- [Monitoring tools \(p. 959\)](#)
- [Logging options for Amazon S3 \(p. 960\)](#)
- [Logging Amazon S3 API calls using AWS CloudTrail \(p. 962\)](#)
- [Logging requests using server access logging \(p. 978\)](#)
- [Monitoring metrics with Amazon CloudWatch \(p. 1002\)](#)
- [Amazon S3 Event Notifications \(p. 1017\)](#)

Monitoring tools

AWS provides various tools that you can use to monitor Amazon S3. You can configure some of these tools to do the monitoring for you, while some of the tools require manual intervention. We recommend that you automate monitoring tasks as much as possible.

Automated monitoring tools

You can use the following automated monitoring tools to watch Amazon S3 and report when something is wrong:

- **Amazon CloudWatch Alarms** – Watch a single metric over a time period that you specify, and perform one or more actions based on the value of the metric relative to a given threshold over a number of time periods. The action is a notification sent to an Amazon Simple Notification Service (Amazon SNS) topic or Amazon EC2 Auto Scaling policy. CloudWatch alarms do not invoke actions simply because they are in a particular state. The state must have changed and been maintained for a specified number of periods. For more information, see [Monitoring metrics with Amazon CloudWatch \(p. 1002\)](#).
- **AWS CloudTrail Log Monitoring** – Share log files between accounts, monitor CloudTrail log files in real time by sending them to CloudWatch Logs, write log processing applications in Java, and validate that your log files have not changed after delivery by CloudTrail. For more information, see [Logging Amazon S3 API calls using AWS CloudTrail \(p. 962\)](#).

Manual monitoring tools

Another important part of monitoring Amazon S3 involves manually monitoring those items that the CloudWatch alarms don't cover. The Amazon S3, CloudWatch, Trusted Advisor, and other AWS Management Console dashboards provide an at-a-glance view of the state of your AWS environment. You might want to enable *server access logging*, which tracks requests for access to your bucket. Each access log record provides details about a single access request, such as the requester, bucket name, request time, request action, response status, and error code, if any. For more information, see [Logging requests using server access logging \(p. 978\)](#).

- The Amazon S3 dashboard shows the following:
 - Your buckets and the objects and properties they contain
- The CloudWatch home page shows the following:
 - Current alarms and status
 - Graphs of alarms and resources
 - Service health status

In addition, you can use CloudWatch to do the following:

- Create [customized dashboards](#) to monitor the services you care about.
- Graph metric data to troubleshoot issues and discover trends.
- Search and browse all your AWS resource metrics.
- Create and edit alarms to be notified of problems.
- AWS Trusted Advisor can help you monitor your AWS resources to improve performance, reliability, security, and cost effectiveness. Four Trusted Advisor checks are available to all users; more than 50 checks are available to users with a Business or Enterprise support plan. For more information, see [AWS Trusted Advisor](#).

Trusted Advisor has these checks that relate to Amazon S3:

- Checks of the logging configuration of Amazon S3 buckets.
- Security checks for Amazon S3 buckets that have open access permissions.
- Fault tolerance checks for Amazon S3 buckets that do not have versioning enabled, or have versioning suspended.

Logging options for Amazon S3

You can record the actions that are taken by users, roles, or AWS services on Amazon S3 resources and maintain log records for auditing and compliance purposes. To do this, you can use server access logging, AWS CloudTrail logging, or a combination of both. We recommend that you use AWS CloudTrail for logging bucket and object-level actions for your Amazon S3 resources. For more information about each option, see the following sections:

- [Logging requests using server access logging \(p. 978\)](#)
- [Logging Amazon S3 API calls using AWS CloudTrail \(p. 962\)](#)

The following table lists the key properties of AWS CloudTrail logs and Amazon S3 server access logs. Review the table and notes to ensure that AWS CloudTrail meets your security requirements.

Log properties	AWS CloudTrail	Amazon S3 server logs
Can be forwarded to other systems (CloudWatch Logs, CloudWatch Events)	Yes	
Deliver logs to more than one destination (for example, send the same logs to two different buckets)	Yes	
Turn on logs for a subset of objects (prefix)	Yes	
Cross-account log delivery (target and source bucket owned by different accounts)	Yes	
Integrity validation of log file using digital signature/hashing	Yes	
Default/choice of encryption for log files	Yes	
Object operations (using Amazon S3 APIs)	Yes	Yes
Bucket operations (using Amazon S3 APIs)	Yes	Yes
Searchable UI for logs	Yes	
Fields for Object Lock parameters, Amazon S3 Select properties for log records	Yes	
Fields for Object Size, Total Time, Turn-Around Time, and HTTP Referer for log records		Yes
Lifecycle transitions, expirations, restores		Yes
Logging of keys in a batch delete operation		Yes
Authentication failures ¹		Yes
Accounts where logs get delivered	Bucket owner ² , and requester	Bucket owner only
Performance and Cost	AWS CloudTrail	Amazon S3 Server Logs
Price	Management events (first delivery) are free; data events incur a fee, in addition to storage of logs	No additional cost in addition to storage of logs
Speed of log delivery	Data events every 5 mins; management events every 15 mins	Within a few hours

Log properties	AWS CloudTrail	Amazon S3 server logs
Log format	JSON	Log file with space-separated, newline-delimited records

Notes:

1. CloudTrail does not deliver logs for requests that fail authentication (in which the provided credentials are not valid). However, it does include logs for requests in which authorization fails (`AccessDenied`) and requests that are made by anonymous users.
2. The S3 bucket owner receives CloudTrail logs only if the account also owns or has full access to the object in the request. For more information, see [Object-level actions in cross-account scenarios \(p. 966\)](#).

Logging Amazon S3 API calls using AWS CloudTrail

Amazon S3 is integrated with AWS CloudTrail, a service that provides a record of actions taken by a user, role, or an AWS service in Amazon S3. CloudTrail captures a subset of API calls for Amazon S3 as events, including calls from the Amazon S3 console and code calls to the Amazon S3 APIs.

If you create a trail, you can enable continuous delivery of CloudTrail events to an Amazon S3 bucket, including events for Amazon S3. If you don't configure a trail, you can still view the most recent events in the CloudTrail console in **Event history**. Using the information collected by CloudTrail, you can determine the request that was made to Amazon S3, the IP address from which the request was made, who made the request, when it was made, and additional details.

To learn more about CloudTrail, including how to configure and enable it, see the [AWS CloudTrail User Guide](#).

Using CloudTrail logs with Amazon S3 server access logs and CloudWatch Logs

AWS CloudTrail logs provide a record of actions taken by a user, role, or an AWS service in Amazon S3, while Amazon S3 server access logs provide detailed records for the requests that are made to an S3 bucket. For more information about how the different logs work, and their properties, performance, and costs, see [the section called "Logging options" \(p. 960\)](#).

You can use AWS CloudTrail logs together with server access logs for Amazon S3. CloudTrail logs provide you with detailed API tracking for Amazon S3 bucket-level and object-level operations. Server access logs for Amazon S3 provide you visibility into object-level operations on your data in Amazon S3. For more information about server access logs, see [Logging requests using server access logging \(p. 978\)](#).

You can also use CloudTrail logs together with CloudWatch for Amazon S3. CloudTrail integration with CloudWatch Logs delivers S3 bucket-level API activity captured by CloudTrail to a CloudWatch log stream in the CloudWatch log group that you specify. You can create CloudWatch alarms for monitoring specific API activity and receive email notifications when the specific API activity occurs. For more information about CloudWatch alarms for monitoring specific API activity, see the [AWS CloudTrail User Guide](#). For more information about using CloudWatch with Amazon S3, see [Monitoring metrics with Amazon CloudWatch \(p. 1002\)](#).

CloudTrail tracking with Amazon S3 SOAP API calls

CloudTrail tracks Amazon S3 SOAP API calls. Amazon S3 SOAP support over HTTP is deprecated, but it is still available over HTTPS. For more information about Amazon S3 SOAP support, see [Appendix a: Using the SOAP API \(p. 1204\)](#).

Important

Newer Amazon S3 features are not supported for SOAP. We recommend that you use either the REST API or the AWS SDKs.

Amazon S3 SOAP actions tracked by CloudTrail logging

SOAP API name	API event name used in CloudTrail log
ListAllMyBuckets	ListBuckets
CreateBucket	CreateBucket
DeleteBucket	DeleteBucket
GetBucketAccessControlPolicy	GetBucketAcl
SetBucketAccessControlPolicy	PutBucketAcl
GetBucketLoggingStatus	GetBucketLogging
SetBucketLoggingStatus	PutBucketLogging

For more information about CloudTrail and Amazon S3, see the following topics:

Topics

- [Amazon S3 CloudTrail events \(p. 963\)](#)
- [CloudTrail log file entries for Amazon S3 and S3 on Outposts \(p. 967\)](#)
- [Enabling CloudTrail event logging for S3 buckets and objects \(p. 971\)](#)
- [Identifying Amazon S3 requests using CloudTrail \(p. 972\)](#)

Amazon S3 CloudTrail events

CloudTrail is enabled on your AWS account when you create the account. When supported event activity occurs in Amazon S3, that activity is recorded in a CloudTrail event along with other AWS service events in **Event history**. You can view, search, and download recent events in your AWS account. For more information, see [Viewing Events with CloudTrail Event History](#).

For an ongoing record of events in your AWS account, including events for Amazon S3, create a trail. A trail enables CloudTrail to deliver log files to an Amazon S3 bucket. By default, when you create a trail in the console, the trail applies to all Regions. The trail logs events from all Regions in the AWS partition and delivers the log files to the Amazon S3 bucket that you specify. Additionally, you can configure other AWS services to further analyze and act upon the event data collected in CloudTrail logs. For more information, see the following:

- [Creating a trail for your AWS account](#)
- [AWS Service Integrations with CloudTrail Logs](#)
- [Configuring Amazon SNS Notifications for CloudTrail](#)
- [Receiving CloudTrail Log Files from Multiple Regions](#)
- [Receiving CloudTrail Log Files from Multiple Accounts](#)

Every event or log entry contains information about who generated the request. The identity information helps you determine the following:

- Whether the request was made with root or IAM user credentials
- Whether the request was made with temporary security credentials for a role or federated user
- Whether the request was made by another AWS service.

For more information, see the [CloudTrail userIdentity Element](#).

You can store your log files in your bucket for as long as you want, but you can also define Amazon S3 Lifecycle rules to archive or delete log files automatically. By default, your log files are encrypted by using Amazon S3 server-side encryption (SSE).

How CloudTrail captures requests made to Amazon S3

By default, CloudTrail logs S3 bucket-level API calls that were made in the last 90 days, but not log requests made to objects. Bucket-level calls include events like `CreateBucket`, `DeleteBucket`, `PutBucketLifecycle`, `PutBucketPolicy`, etc. You can see bucket-level events on the CloudTrail console. However, you can't view data events (Amazon S3 object-level calls) there—you must parse or query CloudTrail logs for them.

Amazon S3 account-level actions tracked by CloudTrail logging

CloudTrail logs account-level actions. Amazon S3 records are written together with other AWS service records in a log file. CloudTrail determines when to create and write to a new file based on a time period and file size.

The tables in this section list the Amazon S3 account-level actions that are supported for logging by CloudTrail.

Amazon S3 account-level API actions tracked by CloudTrail logging appear as the following event names:

- [DeleteAccountPublicAccessBlock](#)
- [GetAccountPublicAccessBlock](#)
- [PutAccountPublicAccessBlock](#)

Amazon S3 bucket-level actions tracked by CloudTrail logging

By default, CloudTrail logs bucket-level actions. Amazon S3 records are written together with other AWS service records in a log file. CloudTrail determines when to create and write to a new file based on a time period and file size.

The tables in this section list the Amazon S3 bucket-level actions that are supported for logging by CloudTrail.

Amazon S3 bucket-level API actions tracked by CloudTrail logging appear as the following event names:

- [CreateBucket](#)
- [DeleteBucket](#)
- [DeleteBucketCors](#)
- [DeleteBucketEncryption](#)
- [DeleteBucketLifecycle](#)
- [DeleteBucketPolicy](#)
- [DeleteBucketReplication](#)
- [DeleteBucketTagging](#)

- [DeleteBucketPublicAccessBlock](#)
- [GetBucketAcl](#)
- [GetBucketCors](#)
- [GetBucketEncryption](#)
- [GetBucketLifecycle](#)
- [GetBucketLocation](#)
- [GetBucketLogging](#)
- [GetBucketNotification](#)
- [GetObjectLockConfiguration](#)
- [GetBucketPolicy](#)
- [GetBucketReplication](#)
- [GetBucketRequestPayment](#)
- [GetBucketTagging](#)
- [GetBucketVersioning](#)
- [GetBucketWebsite](#)
- [GetBucketPublicAccessBlock](#)
- [ListBuckets](#)
- [PutBucketAcl](#)
- [PutBucketCors](#)
- [PutBucketEncryption](#)
- [PutBucketLifecycle](#)
- [PutBucketLogging](#)
- [PutBucketNotification](#)
- [PutBucketPolicy](#)
- [PutBucketReplication](#)
- [PutBucketRequestPayment](#)
- [PutBucketTagging](#)
- [PutBucketVersioning](#)
- [PutBucketWebsite](#)
- [PutBucketPublicAccessBlock](#)

In addition to these API operations, you can also use the [OPTIONS object](#) object-level action. This action is treated like a bucket-level action in CloudTrail logging because the action checks the CORS configuration of a bucket.

Amazon S3 object-level actions tracked by AWS CloudTrail logging

You can also get CloudTrail logs for object-level Amazon S3 actions. To do this, enable data events for your S3 bucket or all buckets in your account. When an object-level action occurs in your account, CloudTrail evaluates your trail settings. If the event matches the object that you specified in a trail, the event is logged. For more information, see [Enabling CloudTrail event logging for S3 buckets and objects \(p. 971\)](#) and [Logging Data Events for Trails](#) in the *AWS CloudTrail User Guide*.

The following object-level API actions are logged as CloudTrail events:

- [AbortMultipartUpload](#)
- [CompleteMultipartUpload](#)

- [DeleteObjects](#)
- [DeleteObject](#)
- [GetObject](#)
- [GetObjectAcl](#)
- [GetObjectAttributes](#)
- [GetObjectTagging](#)
- [HeadObject](#)
- [CreateMultipartUpload](#)
- [ListParts](#)
- [PostObject](#)
- [RestoreObject](#)
- [PutObject](#)
- [PutObjectAcl](#)
- [PutObjectTagging](#)
- [CopyObject](#)
- [UploadPart](#)
- [UploadPartCopy](#)

In addition to these operations, you can use the following bucket-level operations to get CloudTrail logs as object-level Amazon S3 actions under certain conditions:

- [GET Bucket Object \(List Objects\)](#) Version 2 – Select a prefix specified in the trail.
- [GET Bucket Object Versions \(List Object Versions\)](#) – Select a prefix specified in the trail.
- [HEAD Bucket](#) – Specify a bucket and an empty prefix.
- [Delete Multiple Objects](#) – Specify a bucket and an empty prefix.

Note

CloudTrail does not log key names for the keys that are deleted using the Delete Multiple Objects operation.

Object-level actions in cross-account scenarios

The following are special use cases involving the object-level API calls in cross-account scenarios and how CloudTrail logs are reported. CloudTrail always delivers logs to the requester (who made the API call). When setting up cross-account access, consider the examples in this section.

Note

The examples assume that CloudTrail logs are appropriately configured.

Example 1: CloudTrail delivers access logs to the bucket owner

CloudTrail delivers access logs to the bucket owner only if the bucket owner has permissions for the same object API. Consider the following cross-account scenario:

- Account-A owns the bucket.
- Account-B (the requester) tries to access an object in that bucket.
- Account-C owns the object. May be the same account as account-A.

Note

CloudTrail always delivers object-level API access logs to the requester (account-B). In addition, CloudTrail also delivers the same logs to the bucket owner (account-A) only if the bucket owner

owns (account-C) or has permissions for those same API actions on that object. Otherwise, the bucket owner must get permissions, through the object's ACL to get object-level API access logs.

Example 2: CloudTrail does not proliferate email addresses used in setting object ACLs

Consider the following cross-account scenario:

- Account-A owns the bucket.
- Account-B (the requester) sends a request to set an object ACL grant using an email address. For more information about ACLs, see [Access control list \(ACL\) overview \(p. 554\)](#).

The request gets the logs along with the email information. However, the bucket owner—if they are eligible to receive logs, as in example 1—gets the CloudTrail log reporting the event. However, the bucket owner doesn't get the ACL configuration information, specifically the grantee email and the grant. The only information that the log tells the bucket owner is that an ACL API call was made by Account-B.

CloudTrail log file entries for Amazon S3 and S3 on Outposts

A trail is a configuration that enables delivery of events as log files to an Amazon S3 bucket that you specify. CloudTrail log files contain one or more log entries. An event represents a single request from any source. It includes information about the requested action, the date and time of the action, request parameters, and so on. CloudTrail log files are not an ordered stack trace of the public API calls, so they do not appear in any specific order.

For more information, see the following examples.

Topics

- [Example: CloudTrail log file entry for Amazon S3 \(p. 967\)](#)
- [Example: Amazon S3 on Outposts log file entries \(p. 969\)](#)

Example: CloudTrail log file entry for Amazon S3

The following example shows a CloudTrail log entry that demonstrates the [GET Service](#), [PUT Bucket acl](#), and [GET Bucket versioning](#) actions.

```
{  
    "Records": [  
        {  
            "eventVersion": "1.03",  
            "userIdentity": {  
                "type": "IAMUser",  
                "principalId": "111122223333",  
                "arn": "arn:aws:iam::111122223333:user/myUserName",  
                "accountId": "111122223333",  
                "accessKeyId": "AKIAIOSFODNN7EXAMPLE",  
                "userName": "myUserName"  
            },  
            "eventTime": "2019-02-01T03:18:19Z",  
            "eventSource": "s3.amazonaws.com",  
            "eventName": "ListBuckets",  
            "awsRegion": "us-west-2",  
            "sourceIPAddress": "127.0.0.1",  
            "userAgent": "[ ]",  
            "requestParameters": {  
                "host": [
```

```

        "s3.us-west-2.amazonaws.com"
    ],
},
"responseElements": null,
"additionalEventData": {
    "SignatureVersion": "SigV2",
    "AuthenticationMethod": "QueryString"
},
"requestID": "47B8E8D397DCE7A6",
"eventID": "cdc4b7ed-e171-4cef-975a-ad829d4123e8",
"eventType": "AwsApiCall",
"recipientAccountId": "111122223333"
},
{
    "eventVersion": "1.03",
    "userIdentity": {
        "type": "IAMUser",
        "principalId": "111122223333",
        "arn": "arn:aws:iam::111122223333:user/myUserName",
        "accountId": "111122223333",
        "accessKeyId": "AKIAIOSFODNN7EXAMPLE",
        "userName": "myUserName"
    },
    "eventTime": "2019-02-01T03:22:33Z",
    "eventSource": "s3.amazonaws.com",
    "eventName": "PutBucketAcl",
    "awsRegion": "us-west-2",
    "sourceIPAddress": "",
    "userAgent": "[ ]",
    "requestParameters": {
        "bucketName": "",
        "AccessControlPolicy": {
            "AccessControlList": {
                "Grant": {
                    "Grantee": {
                        "xsi:type": "CanonicalUser",
                        "xmlns:xsi": "http://www.w3.org/2001/XMLSchema-instance",
                        "ID": "d25639fbe9c19cd30a4c0f43fbf00e2d3f96400a9aa8dabfbbebe1906Example"
                    },
                    "Permission": "FULL_CONTROL"
                }
            },
            "xmlns": "http://s3.amazonaws.com/doc/2006-03-01/",
            "Owner": {
                "ID": "d25639fbe9c19cd30a4c0f43fbf00e2d3f96400a9aa8dabfbbebe1906Example"
            }
        }
    },
    "host": [
        "s3.us-west-2.amazonaws.com"
    ],
    "acl": [
        ""
    ]
},
"responseElements": null,
"additionalEventData": {
    "SignatureVersion": "SigV4",
    "CipherSuite": "ECDHE-RSA-AES128-SHA",
    "AuthenticationMethod": "AuthHeader"
},
"requestID": "BD8798EACDD16751",
"eventID": "607b9532-1423-41c7-b048-ec2641693c47",
"eventType": "AwsApiCall",
"recipientAccountId": "111122223333"
}
,
```

```
{
    "eventVersion": "1.03",
    "userIdentity": {
        "type": "IAMUser",
        "principalId": "111122223333",
        "arn": "arn:aws:iam::111122223333:user/myUserName",
        "accountId": "111122223333",
        "accessKeyId": "AKIAIOSFODNN7EXAMPLE",
        "userName": "myUserName"
    },
    "eventTime": "2019-02-01T03:26:37Z",
    "eventSource": "s3.amazonaws.com",
    "eventName": "GetBucketVersioning",
    "awsRegion": "us-west-2",
    "sourceIPAddress": "",
    "userAgent": "[ ]",
    "requestParameters": {
        "host": [
            "s3.us-west-2.amazonaws.com"
        ],
        "bucketName": "DOC-EXAMPLE-BUCKET1",
        "versioning": [
            ""
        ]
    },
    "responseElements": null,
    "additionalEventData": {
        "SignatureVersion": "SigV4",
        "CipherSuite": "ECDHE-RSA-AES128-SHA",
        "AuthenticationMethod": "AuthHeader",
    },
    "requestID": "07D681279BD94AED",
    "eventID": "f2b287f3-0df1-4961-a2f4-c4bdfed47657",
    "eventType": "AwsApiCall",
    "recipientAccountId": "111122223333"
}
]
```

Example: Amazon S3 on Outposts log file entries

Amazon S3 on Outposts management events are available via AWS CloudTrail. For more information, see [Logging Amazon S3 API calls using AWS CloudTrail \(p. 962\)](#). In addition, you can optionally [enable logging for data events in AWS CloudTrail](#).

A *trail* is a configuration that enables delivery of events as log files to an S3 bucket in a Region that you specify. CloudTrail logs for your Outposts buckets include a new field, `edgeDeviceDetails`, which identifies the Outpost where the specified bucket is located.

Additional log fields include the requested action, the date and time of the action, and the request parameters. CloudTrail log files are not an ordered stack trace of the public API calls, so they don't appear in any specific order.

The following example shows a CloudTrail log entry that demonstrates a `PutObject` action on `s3-outposts`.

```
{
    "eventVersion": "1.08",
    "userIdentity": {
        "type": "IAMUser",
        "principalId": "111122223333",
        "arn": "arn:aws:iam::111122223333:user/yourUserName",
        "accountId": "222222222222",
        "accessKeyId": "AKIAIOSFODNN7EXAMPLE"
    },
    "eventTime": "2019-02-01T03:26:37Z",
    "eventSource": "s3-outposts.amazonaws.com",
    "eventName": "PutObject",
    "awsRegion": "us-west-2",
    "sourceIPAddress": "123.45.67.89",
    "userAgent": "[ ]",
    "requestParameters": {
        "host": [
            "s3-outposts.us-west-2.amazonaws.com"
        ],
        "bucketName": "DOC-EXAMPLE-BUCKET1",
        "key": "testfile.txt",
        "acl": "public-read",
        "contentLength": 1024,
        "contentType": "text/plain"
    },
    "responseElements": {
        "ETag": "d41d8cd98f00b204e9800998ecf8427e"
    },
    "additionalEventData": {
        "SignatureVersion": "SigV4",
        "CipherSuite": "ECDHE-RSA-AES128-SHA",
        "AuthenticationMethod": "AuthHeader"
    }
}
```

```

    "accessKeyId": "AKIAIOSFODNN7EXAMPLE",
    "userName": "yourUserName"
},
"eventTime": "2020-11-30T15:44:33Z",
"eventSource": "s3-outposts.amazonaws.com",
"eventName": "PutObject",
"awsRegion": "us-east-1",
"sourceIPAddress": "26.29.66.20",
"userAgent": "aws-cli/1.18.39 Python/3.4.10 Darwin/18.7.0 botocore/1.15.39",
"requestParameters": {
    "expires": "Wed, 21 Oct 2020 07:28:00 GMT",
    "Content-Language": "english",
    "x-amz-server-side-encryption-customer-key-MD5": "wJalrXUtnFEMI/K7MDENG/bPxRfiCYEXAMPLEKEY",
    "ObjectCannedACL": "BucketOwnerFullControl",
    "x-amz-server-side-encryption": "Aes256",
    "Content-Encoding": "gzip",
    "Content-Length": "10",
    "Cache-Control": "no-cache",
    "Content-Type": "text/html; charset=UTF-8",
    "Content-Disposition": "attachment",
    "Content-MD5": "je7MtGbClwBF/2Zp9Utk/h3yCo8nvbEXAMPLEKEY",
    "x-amz-storage-class": "Outposts",
    "x-amz-server-side-encryption-customer-algorithm": "Aes256",
    "bucketName": "DOC-EXAMPLE-BUCKET1",
    "Key": "path/upload.sh"
},
"responseElements": {
    "x-amz-server-side-encryption-customer-key-MD5": "wJalrXUtnFEMI/K7MDENG/bPxRfiCYEXAMPLEKEY",
    "x-amz-server-side-encryption": "Aes256",
    "x-amz-version-id": "001",
    "x-amz-server-side-encryption-customer-algorithm": "Aes256",
    "ETag": "d41d8cd98f00b204e9800998ecf8427f"
},
"additionalEventData": {
    "CipherSuite": "ECDHE-RSA-AES128-SHA",
    "bytesTransferredIn": 10,
    "x-amz-id-2": "29xQBV2O+xOHKITvzY1suLv1i6A52E0zOX159fpfsItYd58JhXwKxXAXI4IQkp6",
    "SignatureVersion": "SigV4",
    "bytesTransferredOut": 20,
    "AuthenticationMethod": "AuthHeader"
},
"requestID": "8E96D972160306FA",
"eventID": "ee3b4e0c-ab12-459b-9998-0a5a6f2e4015",
"readOnly": false,
"resources": [
    {
        "accountId": "222222222222",
        "type": "AWS::S3Outposts::Object",
        "ARN": "arn:aws:s3-outposts:us-east-1:YYY:outpost/op-01ac5d28a6a232904/bucket/path/upload.sh"
    },
    {
        "accountId": "222222222222",
        "type": "AWS::S3Outposts::Bucket",
        "ARN": "arn:aws:s3-outposts:us-east-1:YYY:outpost/op-01ac5d28a6a232904/bucket/"
    }
],
"eventType": "AwsApiCall",
"managementEvent": false,
"recipientAccountId": "444455556666",
"sharedEventID": "02759a4c-c040-4758-b84b-7cbaaf17747a",
"edgeDeviceDetails": {
    "type": "outposts",
    "deviceID": "op-01ac5d28a6a232904"
}

```

```
    },
    "eventCategory": "Data"
}
```

Enabling CloudTrail event logging for S3 buckets and objects

You can use CloudTrail data events to get information about bucket and object-level requests in Amazon S3. To enable CloudTrail data events for all your buckets or for a list of specific buckets, you must [create a trail manually in CloudTrail](#).

Note

- The default setting for CloudTrail is to find only management events. Check to ensure that you have the data events enabled for your account.
- With an S3 bucket that is generating a high workload, you could quickly generate thousands of logs in a short amount of time. Be mindful of how long you choose to enable CloudTrail data events for a busy bucket.

CloudTrail stores Amazon S3 data event logs in an S3 bucket of your choosing. Consider using a bucket in a separate AWS account to better organize events from multiple buckets you might own into a central place for easier querying and analysis. AWS Organizations makes it easy to create an AWS account that is linked to the account owning the bucket that you are monitoring. For more information, see [What is AWS Organizations](#) in the *AWS Organizations User Guide*.

When you create a trail in CloudTrail, in the data events section, you can select the **Select all S3 buckets in your account** check box to log all object level events.

Note

- It's a best practice to create a lifecycle policy for your AWS CloudTrail data event bucket. Configure the lifecycle policy to periodically remove log files after the period of time you believe you need to audit them. Doing so reduces the amount of data that Athena analyzes for each query. For more information, see [Setting lifecycle configuration on a bucket \(p. 708\)](#).
- For more information about logging format, see [Logging Amazon S3 API calls using AWS CloudTrail \(p. 962\)](#).
- For examples of how to query CloudTrail logs, see the [AWS Big Data Blog post Analyze Security, Compliance, and Operational Activity Using AWS CloudTrail and Amazon Athena](#).

Enable logging for objects in a bucket using the console

You can use the Amazon S3 console to configure an AWS CloudTrail trail to log data events for objects in an S3 bucket. CloudTrail supports logging Amazon S3 object-level API operations such as `GetObject`, `DeleteObject`, and `PutObject`. These events are called *data events*.

By default, CloudTrail trails don't log data events, but you can configure trails to log data events for S3 buckets that you specify, or to log data events for all the Amazon S3 buckets in your AWS account. For more information, see [Logging Amazon S3 API calls using AWS CloudTrail \(p. 962\)](#).

CloudTrail does not populate data events in the CloudTrail event history. Additionally, not all bucket-level actions are populated in the CloudTrail event history. For more information about the Amazon S3 bucket-level API actions tracked by CloudTrail logging, see [Amazon S3 bucket-level actions tracked by CloudTrail logging \(p. 964\)](#). For more information about how to query CloudTrail logs, see the AWS Knowledge Center article about [using Amazon CloudWatch Logs filter patterns and Amazon Athena to query CloudTrail logs](#).

To configure a trail to log data events for an S3 bucket, you can use either the AWS CloudTrail console or the Amazon S3 console. If you are configuring a trail to log data events for all the Amazon S3 buckets in your AWS account, it's easier to use the CloudTrail console. For information about using the CloudTrail console to configure a trail to log S3 data events, see [Data events](#) in the *AWS CloudTrail User Guide*.

Important

Additional charges apply for data events. For more information, see [AWS CloudTrail pricing](#).

The following procedure shows how to use the Amazon S3 console to configure a CloudTrail trail to log data events for an S3 bucket.

To enable CloudTrail data events logging for objects in an S3 bucket

1. Sign in to the AWS Management Console and open the Amazon S3 console at <https://console.aws.amazon.com/s3/>.
2. In the **Buckets** list, choose the name of the bucket.
3. Choose **Properties**.
4. Under **AWS CloudTrail data events**, choose **Configure in CloudTrail**.

You can create a new CloudTrail trail or reuse an existing trail and configure Amazon S3 data events to be logged in your trail. For information about how to create trails in the CloudTrail console, see [Creating and updating a trail with the console](#) in the *AWS CloudTrail User Guide*. For information about how to configure Amazon S3 data event logging in the CloudTrail console, see [Logging data events for Amazon S3 Objects](#) in the *AWS CloudTrail User Guide*.

Note

If you use the CloudTrail console or the Amazon S3 console to configure a trail to log data events for an S3 bucket, the Amazon S3 console shows that object-level logging is enabled for the bucket.

To disable CloudTrail data events logging for objects in an S3 bucket

- To disable object-level logging for the bucket, you must open the CloudTrail console and remove the bucket name from the trail's **Data events**.

For information about enabling object-level logging when you create an S3 bucket, see [Creating a bucket](#) (p. 119).

For more information about CloudTrail logging with S3 buckets, see the following topics:

- [Viewing the properties for an S3 bucket](#) (p. 124)
- [Logging Amazon S3 API calls using AWS CloudTrail](#) (p. 962)
- [Working with CloudTrail Log Files](#) in the *AWS CloudTrail User Guide*

Identifying Amazon S3 requests using CloudTrail

In Amazon S3, you can identify requests using an AWS CloudTrail event log. AWS CloudTrail is the preferred way of identifying Amazon S3 requests, but if you are using Amazon S3 server access logs, see [the section called "Identifying S3 requests"](#) (p. 998).

Topics

- [Identifying requests made to Amazon S3 in a CloudTrail log](#) (p. 973)
- [Identifying Amazon S3 Signature Version 2 requests using CloudTrail](#) (p. 974)
- [Identifying access to S3 objects using CloudTrail](#) (p. 977)

Identifying requests made to Amazon S3 in a CloudTrail log

Events logged by CloudTrail are stored as compressed, GZipped JSON objects in your S3 bucket. To efficiently find requests, you should use a service like Amazon Athena to index and query the CloudTrail logs. For more information about CloudTrail and Athena, see [Querying AWS CloudTrail Logs](#) in the [Amazon Athena User Guide](#).

Using Athena with CloudTrail logs

After you set up CloudTrail to deliver events to a bucket, you should start to see objects go to your destination bucket on the Amazon S3 console. These are formatted as follows:

```
s3://myawsexamplebucket1/AWSLogs/111122223333/CloudTrail/Region/yyyy/mm/dd
```

Example — Use Athena to query CloudTrail event logs for specific requests

Locate your CloudTrail event logs:

```
s3://myawsexamplebucket1/AWSLogs/111122223333/CloudTrail/us-east-2/2019/04/14
```

With CloudTrail event logs, you can now create an Athena database and table to query them as follows:

1. Open the Athena console at <https://console.aws.amazon.com/athena/>.
2. Change the AWS Region to be the same as your CloudTrail destination S3 bucket.
3. In the query window, create an Athena database for your CloudTrail events.

```
CREATE DATABASE s3_cLOUDTRAIL_EVENTS_DB
```

4. Use the following query to create a table for all of your CloudTrail events in the bucket. Be sure to change the bucket name from `CloudTrail_myawsexamplebucket1` to your bucket's name. Also provide the `AWS_ACCOUNT_ID` CloudTrail that is used in your bucket.

```
CREATE EXTERNAL TABLE s3_cLOUDTRAIL_EVENTS_DB.cloudtrail_myawsexamplebucket1_table(
    eventversion STRING,
    useridentity STRUCT<
        type:STRING,
        principalid:STRING,
        arn:STRING,
        accountid:STRING,
        invokedby:STRING,
        accesskeyid:STRING,
        userName:STRING,
        sessioncontext:STRUCT<
            attributes:STRUCT<
                mfaauthenticated:STRING,
                creationdate:STRING>,
            sessionissuer:STRUCT<
                type:STRING,
                principalid:STRING,
                arn:STRING,
                accountId:STRING,
                userName:STRING>
        >
    >,
    eventtime STRING,
    eventsource STRING,
    eventname STRING,
    awsregion STRING,
    sourceipaddress STRING,
    useragent STRING,
```

```
    errorcode STRING,
    errormessage STRING,
    requestparameters STRING,
    responseelements STRING,
    additionaleventdata STRING,
    requestid STRING,
    eventid STRING,
    resources ARRAY<STRUCT<
        ARN:STRING,
        accountId:STRING,
        type:STRING>>,
    eventtype STRING,
    apiversion STRING,
    readonly STRING,
    recipientaccountid STRING,
    serviceeventdetails STRING,
    sharedeventid STRING,
    vpcendpointid STRING
)
ROW FORMAT SERDE 'com.amazon.emr.hive.serde.CloudTrailSerde'
STORED AS INPUTFORMAT 'com.amazon.emr.cloudtrail.CloudTrailInputFormat'
OUTPUTFORMAT 'org.apache.hadoop.hive.ql.io.HiveIgnoreKeyTextOutputFormat'
LOCATION 's3://myawsexamplebucket1/AWSLogs/11122223333/';
```

5. Test Athena to ensure that the query works.

```
SELECT * FROM s3_cloudtrail_events_db.cloudtrail_myawsexamplebucket1_table
WHERE eventsource='s3.amazonaws.com'
LIMIT 2;
```

Identifying Amazon S3 Signature Version 2 requests using CloudTrail

You can use a CloudTrail event log to identify which API signature version was used to sign a request in Amazon S3. This capability is important because support for Signature Version 2 will be turned off (deprecated). After that, Amazon S3 will no longer accept requests that use Signature Version 2, and all requests must use *Signature Version 4* signing.

We strongly recommend that you use CloudTrail to help determine whether any of your workflows are using Signature Version 2 signing. Remediate them by upgrading your libraries and code to use Signature Version 4 instead to prevent any impact to your business.

For more information, see [Announcement: AWS CloudTrail for Amazon S3 adds new fields for enhanced security auditing](#) in the AWS Discussion Forums.

Note

CloudTrail events for Amazon S3 include the signature version in the request details under the key name of 'additionalEventData'. To find the signature version on requests made for objects in Amazon S3 such as GETs, PUTs, and DELETEs, you must enable CloudTrail data events (which is turned off by default).

AWS CloudTrail is the preferred method for identifying Signature Version 2 requests. If you're using Amazon S3 server access logs, see [Identifying Signature Version 2 requests using Amazon S3 access logs \(p. 1001\)](#).

Topics

- [Athena query examples for identifying Amazon S3 Signature Version 2 requests \(p. 975\)](#)

- Partitioning Signature Version 2 data (p. 975)

Athena query examples for identifying Amazon S3 Signature Version 2 requests

Example — Select all Signature Version 2 events, and print only EventTime, S3 action, Request_Parameters, Region, SourceIP, and UserAgent

In the following Athena query, replace

`<s3_cLOUDTRAIL_EVENTS_DB.cloudtrail_myawsexamplebucket1_table>` with your Athena details, and increase or remove the limit as needed.

```
SELECT EventTime, EventName as S3_Action, requestParameters as Request_Parameters,
awsregion as AWS_Region, sourceipaddress as Source_IP, useragent as User_Agent
FROM s3_cLOUDTRAIL_EVENTS_DB.cloudtrail_myawsexamplebucket1_table
WHERE eventsource='s3.amazonaws.com'
AND json_extract_scalar(additionalEventData, '$.SignatureVersion')='SigV2'
LIMIT 10;
```

Example — Select all requesters that are sending Signature Version 2 traffic

```
SELECT useridentity.arn, Count(requestid) as RequestCount
FROM s3_cLOUDTRAIL_EVENTS_DB.cloudtrail_myawsexamplebucket1_table
WHERE eventsource='s3.amazonaws.com'
    and json_extract_scalar(additionalEventData, '$.SignatureVersion')='SigV2'
Group by useridentity.arn
```

Partitioning Signature Version 2 data

If you have a large amount of data to query, you can reduce the costs and runtime of Athena by creating a partitioned table.

To do this, create a new table with partitions as follows.

```
CREATE EXTERNAL TABLE
s3_cLOUDTRAIL_EVENTS_DB.cloudtrail_myawsexamplebucket1_table_partitioned(
    eventversion STRING,
    userIdentity STRUCT<
        type:STRING,
        principalId:STRING,
        arn:STRING,
        accountId:STRING,
        invokedBy:STRING,
        accessKeyId:STRING,
        userName:STRING,
        sessionContext:STRUCT<
            attributes:STRUCT<
                mfaAuthenticated:STRING,
                creationDate:STRING>,
            sessionIssuer:STRUCT<
                type:STRING,
                principalId:STRING,
                arn:STRING,
                accountId:STRING,
                userName:STRING>
```

```

        >
        >,
eventTime STRING,
eventSource STRING,
eventName STRING,
awsRegion STRING,
sourceIpAddress STRING,
userAgent STRING,
errorCode STRING,
errorMessage STRING,
requestParameters STRING,
responseElements STRING,
additionalEventData STRING,
requestId STRING,
eventId STRING,
resources ARRAY<STRUCT<ARN:STRING,accountId: STRING,type:STRING>>,
eventType STRING,
apiVersion STRING,
readOnly STRING,
recipientAccountId STRING,
serviceEventDetails STRING,
sharedEventId STRING,
vpcEndpointId STRING
)
PARTITIONED BY (region string, year string, month string, day string)
ROW FORMAT SERDE 'com.amazon.emr.hive.serde.CloudTrailSerde'
STORED AS INPUTFORMAT 'com.amazon.emr.cloudtrail.CloudTrailInputFormat'
OUTPUTFORMAT 'org.apache.hadoop.hive.ql.io.HiveIgnoreKeyTextOutputFormat'
LOCATION 's3://myawsexamplebucket1/AWSLogs/111122223333/';

```

Then, create the partitions individually. You can't get results from dates that you haven't created.

```

ALTER TABLE s3_cloudtrail_events_db.cloudtrail_myawsexamplebucket1_table_partitioned ADD
PARTITION (region= 'us-east-1', year= '2019', month= '02', day= '19') LOCATION 's3://
myawsexamplebucket1/AWSLogs/111122223333/CloudTrail/us-east-1/2019/02/19/'
    PARTITION (region= 'us-west-1', year= '2019', month= '02', day= '19') LOCATION 's3://
myawsexamplebucket1/AWSLogs/111122223333/CloudTrail/us-west-1/2019/02/19/'
    PARTITION (region= 'us-west-2', year= '2019', month= '02', day= '19') LOCATION 's3://
myawsexamplebucket1/AWSLogs/111122223333/CloudTrail/us-west-2/2019/02/19/'
    PARTITION (region= 'ap-southeast-1', year= '2019', month= '02', day= '19') LOCATION
's3://myawsexamplebucket1/AWSLogs/111122223333/CloudTrail/ap-southeast-1/2019/02/19/'
    PARTITION (region= 'ap-southeast-2', year= '2019', month= '02', day= '19') LOCATION
's3://myawsexamplebucket1/AWSLogs/111122223333/CloudTrail/ap-southeast-2/2019/02/19/'
    PARTITION (region= 'ap-northeast-1', year= '2019', month= '02', day= '19') LOCATION
's3://myawsexamplebucket1/AWSLogs/111122223333/CloudTrail/ap-northeast-1/2019/02/19/'
    PARTITION (region= 'eu-west-1', year= '2019', month= '02', day= '19') LOCATION 's3://
myawsexamplebucket1/AWSLogs/111122223333/CloudTrail/eu-west-1/2019/02/19/'
    PARTITION (region= 'sa-east-1', year= '2019', month= '02', day= '19') LOCATION 's3://
myawsexamplebucket1/AWSLogs/111122223333/CloudTrail/sa-east-1/2019/02/19/';

```

You can then make the request based on these partitions, and you don't need to load the full bucket.

```

SELECT useridentity.arn,
Count(requestid) AS RequestCount
FROM s3_cloudtrail_events_db.cloudtrail_myawsexamplebucket1_table_partitioned
WHERE eventsource='s3.amazonaws.com'
AND json_extract_scalar(additionalEventData, '$.SignatureVersion')='SigV2'
AND region='us-east-1'
AND year='2019'

```

```
AND month='02'  
AND day='19'  
Group by useridentity.arn
```

Identifying access to S3 objects using CloudTrail

You can use your AWS CloudTrail event log to identify Amazon S3 object access requests for data events such as `GetObject`, `DeleteObject`, and `PutObject`, and discover additional information about those requests.

The following example shows how to get all PUT object requests for Amazon S3 from the AWS CloudTrail event log.

Topics

- [Athena query examples for identifying Amazon S3 object access requests \(p. 977\)](#)

Athena query examples for identifying Amazon S3 object access requests

In the following Athena query examples, replace

`<s3_clouptrail_events_db.cloudtrail_myawsexamplebucket1_table>` with your Athena details, and modify the date range as needed.

Example — Select all events that have PUT object access requests, and print only EventTime, EventSource, SourceIP, UserAgent, BucketName, object, and UserARN

```
SELECT  
    eventTime,  
    eventName,  
    eventSource,  
    sourceIpAddress,  
    userAgent,  
    json_extract_scalar(requestParameters, '$.bucketName') as bucketName,  
    json_extract_scalar(requestParameters, '$.key') as object,  
    userIdentity.arn as userArn  
FROM  
    s3_clouptrail_events_db.cloudtrail_myawsexamplebucket_table  
WHERE  
    eventName = 'PutObject'  
    AND eventTime BETWEEN '2019-07-05T00:00:00Z' and '2019-07-06T00:00:00Z'
```

Example — Select all events that have GET object access requests, and print only EventTime, EventSource, SourceIP, UserAgent, BucketName, object, and UserARN

```
SELECT  
    eventTime,  
    eventName,  
    eventSource,  
    sourceIpAddress,  
    userAgent,  
    json_extract_scalar(requestParameters, '$.bucketName') as bucketName,  
    json_extract_scalar(requestParameters, '$.key') as object,  
    userIdentity.arn as userArn  
FROM  
    s3_clouptrail_events_db.cloudtrail_myawsexamplebucket_table  
WHERE  
    eventName = 'GetObject'  
    AND eventTime BETWEEN '2019-07-05T00:00:00Z' and '2019-07-06T00:00:00Z'
```

Example — Select all anonymous requester events to a bucket in a certain period and print only EventTime, EventName, EventSource, SourceIP, UserAgent, BucketName, UserARN, and AccountID

```
SELECT
    eventTime,
    eventName,
    eventSource,
    sourceIpAddress,
    userAgent,
    json_extract_scalar(requestParameters, '$.bucketName') as bucketName,
    userIdentity.arn as userArn,
    userIdentity.accountId
FROM
    s3_clouptrail_events_db.cloudtrail_myawsexamplebucket_table
WHERE
    userIdentity.accountId = 'ANONYMOUS_PRINCIPAL'
    AND eventTime BETWEEN '2019-07-05T00:00:00Z' and '2019-07-06T00:00:00Z'
```

Note

- These query examples can also be useful for security monitoring. You can review the results for PutObject or GetObject calls from unexpected or unauthorized IP addresses/requesters and for identifying any anonymous requests to your buckets.
- This query only retrieves information from the time at which logging was enabled.

If you are using Amazon S3 server access logs, see [Identifying object access requests using Amazon S3 access logs \(p. 1001\)](#).

Logging requests using server access logging

Server access logging provides detailed records for the requests that are made to a bucket. Server access logs are useful for many applications. For example, access log information can be useful in security and access audits. It can also help you learn about your customer base and understand your Amazon S3 bill.

Note

Server access logs don't record information about wrong-region redirect errors for Regions that launched after March 20, 2019. Wrong-region redirect errors occur when a request for an object or bucket is made outside the Region in which the bucket exists.

How do I enable log delivery?

To enable log delivery, perform the following basic steps. For details, see [Enabling Amazon S3 server access logging \(p. 980\)](#).

1. **Provide the name of the target bucket.** This bucket is where you want Amazon S3 to save the access logs as objects. Both the source and target buckets must be in the same AWS Region and owned by the same account.

You can have logs delivered to any bucket that you own that is in the same Region as the source bucket, including the source bucket itself. But for simpler log management, we recommend that you save access logs in a different bucket.

When your source bucket and target bucket are the same bucket, additional logs are created for the logs that are written to the bucket. We do not recommend doing this because it could result in a small increase in your storage billing. In addition, the extra logs about logs might make it harder to find the log that you are looking for. If you choose to save access logs in the source bucket, we recommend

that you specify a prefix for all log object keys so that the object names begin with a common string and the log objects are easier to identify.

[Key prefixes](#) are also useful to distinguish between source buckets when multiple buckets log to the same target bucket.

2. **(Optional) Assign a prefix to all Amazon S3 log object keys.** The prefix makes it simpler for you to locate the log objects. For example, if you specify the prefix value logs/, each log object that Amazon S3 creates begins with the logs/ prefix in its key.

```
logs/2013-11-01-21-32-16-E568B2907131C0C0
```

The key prefix can also help when you delete the logs. For example, you can set a lifecycle configuration rule for Amazon S3 to delete objects with a specific key prefix. For more information, see [Deleting Amazon S3 log files \(p. 998\)](#).

3. **(Optional) Set permissions in target grants so that others can access the generated logs.** By default, only the bucket owner always has full access to the log objects. If your target bucket (where your server access logs are stored) uses the bucket owner enforced setting for S3 Object Ownership to disable access control lists (ACLs), you can't grant permissions in target grants that use ACLs. However, you can update your bucket policy for the target bucket to grant access to others. For more information, see [Identity and access management in Amazon S3 \(p. 394\)](#) and [Permissions for log delivery \(p. 981\)](#).

Log object key format

Amazon S3 uses the following object key format for the log objects it uploads in the target bucket:

```
TargetPrefixYYYY-mm-DD-HH-MM-SS-UniqueString/
```

In the key, YYYY, mm, DD, HH, MM, and SS are the digits of the year, month, day, hour, minute, and seconds (respectively) when the log file was delivered. These dates and times are in Coordinated Universal Time (UTC).

A log file delivered at a specific time can contain records written at any point before that time. There is no way to know whether all log records for a certain time interval have been delivered or not.

The UniqueString component of the key is there to prevent overwriting of files. It has no meaning, and log processing software should ignore it.

The trailing slash / is required to denote the end of the prefix.

How are logs delivered?

Amazon S3 periodically collects access log records, consolidates the records in log files, and then uploads log files to your target bucket as log objects. If you enable logging on multiple source buckets that identify the same target bucket, the target bucket will have access logs for all those source buckets. However, each log object reports access log records for a specific source bucket.

Amazon S3 uses a special log delivery account to write server access logs. These writes are subject to the usual access control restrictions. We recommend that you update the bucket policy on the target bucket to grant access to the logging service principal (logging.s3.amazonaws.com) for access log delivery. However, you can also grant access for access log delivery to the S3 log delivery group through your bucket access control list (ACL). Granting access to the S3 log delivery group using your bucket ACL is not recommended.

When you enable server access logging and grant access for access log delivery through your bucket policy, you update the bucket policy on the target bucket to allow s3:PutObject access for the logging

service principal. If you use the Amazon S3 console to enable server access logging on a bucket, the console automatically updates the bucket policy on the target bucket to grant these permissions to the logging service principal. For more information about granting permissions for server access log delivery, see [Permissions for log delivery \(p. 981\)](#).

Bucket owner enforced setting for S3 Object Ownership

If the target bucket uses the bucket owner enforced setting for Object Ownership, ACLs are disabled and no longer affect permissions. You must update the bucket policy on the target bucket to grant access to the logging service principal. For more information about Object Ownership, see [Grant access to S3 log delivery group for server access logging \(p. 612\)](#).

Best effort server log delivery

Server access log records are delivered on a best effort basis. Most requests for a bucket that is properly configured for logging result in a delivered log record. Most log records are delivered within a few hours of the time that they are recorded, but they can be delivered more frequently.

The completeness and timeliness of server logging is not guaranteed. The log record for a particular request might be delivered long after the request was actually processed, or *it might not be delivered at all*. The purpose of server logs is to give you an idea of the nature of traffic against your bucket. It is rare to lose log records, but server logging is not meant to be a complete accounting of all requests.

It follows from the best-effort nature of the server logging feature that the usage reports available at the AWS portal (Billing and Cost Management reports on the [AWS Management Console](#)) might include one or more access requests that do not appear in a delivered server log.

Bucket logging status changes take effect over time

Changes to the logging status of a bucket take time to actually affect the delivery of log files. For example, if you enable logging for a bucket, some requests made in the following hour might be logged, while others might not. If you change the target bucket for logging from bucket A to bucket B, some logs for the next hour might continue to be delivered to bucket A, while others might be delivered to the new target bucket B. In all cases, the new settings eventually take effect without any further action on your part.

For more information about logging and log files, see the following sections:

Topics

- [Enabling Amazon S3 server access logging \(p. 980\)](#)
- [Amazon S3 server access log format \(p. 989\)](#)
- [Deleting Amazon S3 log files \(p. 998\)](#)
- [Using Amazon S3 access logs to identify requests \(p. 998\)](#)

Enabling Amazon S3 server access logging

Server access logging provides detailed records for the requests that are made to an Amazon S3 bucket. Server access logs are useful for many applications. For example, access log information can be useful in security and access audits. It can also help you learn about your customer base and understand your Amazon S3 bill.

By default, Amazon S3 doesn't collect server access logs. When you enable logging, Amazon S3 delivers access logs for a source bucket to a target bucket that you choose. The target bucket must be in the same AWS Region and AWS account as the source bucket, and must not have a default retention period configuration.

An access log record contains details about the requests that are made to a bucket. This information can include the request type, the resources that are specified in the request, and the time and date that the request was processed. For more information about logging basics, see [Logging requests using server access logging \(p. 978\)](#).

Important

- There is no extra charge for enabling server access logging on an Amazon S3 bucket. However, any log files that the system delivers to you will accrue the usual charges for storage. (You can delete the log files at any time.) We do not assess data transfer charges for log file delivery, but we do charge the normal data transfer rate for accessing the log files.
- Your target bucket should not have server access logging enabled. You can have logs delivered to any bucket that you own that is in the same Region as the source bucket, including the source bucket itself. However, this would cause an infinite loop of logs and is not recommended. For simpler log management, we recommend that you save access logs in a different bucket. For more information, see [How do I enable log delivery? \(p. 978\)](#)

You can enable or disable server access logging by using the Amazon S3 console, Amazon S3 API, the AWS Command Line Interface (AWS CLI), or AWS SDKs.

Before you enable server access logging, consider the following:

- You can use either a bucket policy or bucket access control lists (ACL) to grant log delivery permissions. However, we recommend that you use a bucket policy. If the target bucket uses the bucket owner enforced setting for Object Ownership, ACLs are disabled and no longer affect permissions. You must use a bucket policy to grant access permissions to the logging service principal. For more information, see [Permissions for log delivery \(p. 981\)](#).
- Adding *deny* conditions to a bucket policy might prevent Amazon S3 from delivering access logs.
- You can use [default bucket encryption](#) on the target bucket *only* if you use **AES256 (SSE-S3)**. Default encryption with AWS KMS keys (SSE-KMS) is not supported.
- You can't enable S3 Object Lock on the target bucket.

Permissions for log delivery

Amazon S3 uses a special log delivery account to write server access logs. These writes are subject to the usual access control restrictions. We recommend that you update the bucket policy on the target bucket to grant access to the logging service principal (`logging.s3.amazonaws.com`) for access log delivery.

To grant access using the bucket policy on the target bucket, you update the bucket policy to allow `s3:PutObject` access for the logging service principal. If you use the Amazon S3 console to enable server access logging, the console automatically updates the bucket policy on the target bucket to grant these permissions to the logging service principal. If you enable server access logging programmatically, you can manually update the bucket policy for the target bucket to grant access to the logging service principal.

You can alternately use bucket ACLs to grant access for access log delivery. You add a grant entry to the bucket ACL that grants `WRITE` and `READ_ACP` permissions to the S3 log delivery group. Granting access to the S3 log delivery group using your bucket ACL is not recommended.

Bucket owner enforced setting for S3 Object Ownership

If the target bucket uses the bucket owner enforced setting for Object Ownership, ACLs are disabled and no longer affect permissions. You must update the bucket policy for the target bucket to grant access to the logging service principal. You can't update your bucket ACL to grant access to the S3 log delivery group. You also can't include target grants in your [PutBucketLogging](#) configuration. For information about migrating existing bucket ACLs for access log delivery to a bucket policy, see [Grant access to S3](#)

log delivery group for server access logging (p. 612). For more information about Object Ownership see [Controlling ownership of objects and disabling ACLs for your bucket \(p. 601\)](#).

Grant permissions to the logging service principal using a bucket policy

This example bucket policy grants s3:PutObject permissions to the logging service principal (logging.s3.amazonaws.com). To use this bucket policy, replace the example values.

```
{  
    "Version": "2012-10-17",  
    "Statement": [  
        {  
            "Sid": "S3ServerAccessLogsPolicy",  
            "Effect": "Allow",  
            "Principal": {  
                "Service": "logging.s3.amazonaws.com"  
            },  
            "Action": [  
                "s3:PutObject"  
            ],  
            "Resource": "arn:aws:s3:::DOC-EXAMPLE-BUCKET/EXAMPLE-LOGGING-PREFIX*",  
            "Condition": {  
                "ArnLike": {  
                    "aws:SourceArn": "arn:aws:s3:::SOURCE-BUCKET-NAME"  
                },  
                "StringEquals": {  
                    "aws:SourceAccount": "SOURCE-ACCOUNT-ID"  
                }  
            }  
        }  
    ]  
}
```

Grant permissions to the log delivery group using the bucket ACL

While we do not recommend this approach, you can grant permissions to the log delivery group using bucket ACL. However, if the target bucket uses the bucket owner enforced setting for Object Ownership, you can't set bucket or object ACLs. You also can't include target grants in your [PutBucketLogging](#) configuration. You must use a bucket policy to grant access to the logging service principal (logging.s3.amazonaws.com). For more information, see [Permissions for log delivery \(p. 981\)](#).

In the bucket ACL, the log delivery group is represented by the following URL.

```
http://acs.amazonaws.com/groups/s3/LogDelivery
```

To grant WRITE and READ_ACP (ACL read) permissions, add the following grants to the target bucket ACL.

```
<Grant>  
    <Grantee xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:type="Group">  
        <URI>http://acs.amazonaws.com/groups/s3/LogDelivery</URI>  
    </Grantee>  
    <Permission>WRITE</Permission>  
</Grant>  
<Grant>  
    <Grantee xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:type="Group">  
        <URI>http://acs.amazonaws.com/groups/s3/LogDelivery</URI>  
    </Grantee>  
    <Permission>READ_ACP</Permission>  
</Grant>
```

For examples of adding ACL grants programmatically, see [Configuring ACLs \(p. 562\)](#).

Important

When you enable Amazon S3 server access logging using AWS CloudFormation on a bucket and use ACLs to grant access to the S3 log delivery group, you must also add "AccessControl": "LogDeliveryWrite" in the property field of your bucket. This is important because you can only grant those permissions by creating an ACL for the bucket, but you can't create custom ACLs for buckets in CloudFormation. You can only use canned ACLs.

To enable server access logging

Use the following examples to enable server access logging using the AWS Management Console, AWS CLI, REST API, and AWS SDK for .NET.

Using the S3 console

1. Sign in to the AWS Management Console and open the Amazon S3 console at <https://console.aws.amazon.com/s3/>.
2. In the **Buckets** list, choose the name of the bucket that you want to enable server access logging for.
3. Choose **Properties**.
4. In the **Server access logging** section, choose **Edit**.
5. Under **Server access logging**, select **Enable**.
6. For **Target bucket**, enter the name of the bucket that you want to receive the log record objects.

The target bucket must be in the same Region as the source bucket and must not have a default retention period configuration.

7. Choose **Save changes**.

When you enable server access logging on a bucket, the console both enables logging on the source bucket and updates the bucket policy for the target bucket to grant s3:PutObject permissions to the logging service principal (logging.s3.amazonaws.com). For more information about this bucket policy, see [Grant permissions to the logging service principal using a bucket policy \(p. 982\)](#).

You can view the logs in the target bucket. After you enable server access logging, it might take a few hours before the logs are delivered to the target bucket. For more information about how and when logs are delivered, see [How are logs delivered? \(p. 979\)](#).

For more information, see [Viewing the properties for an S3 bucket \(p. 124\)](#).

Using the REST API

To enable logging, you submit a **PUT Bucket logging** request to add the logging configuration on the source bucket. The request specifies the target bucket and, optionally, the prefix to be used with all log object keys.

The following example identifies **LOGBUCKET** as the target bucket and **logs/** as the prefix.

```
<BucketLoggingStatus xmlns="http://doc.s3.amazonaws.com/2006-03-01">
  <LoggingEnabled>
    <TargetBucket>LOGBUCKET</TargetBucket>
    <TargetPrefix>logs/</TargetPrefix>
  </LoggingEnabled>
</BucketLoggingStatus>
```

The log objects are written and owned by the S3 log delivery account, and the bucket owner is granted full permissions on the log objects. You can optionally use target grants to grant permissions to other users so that they can access the logs. For more information, see [PUT Bucket logging](#).

Note

If the target bucket uses the bucket owner enforced setting for Object Ownership, you can't use target grants to grant permissions to other users. To grant permissions to others, you can use update the bucket policy on the target bucket. For more information, see [Permissions for log delivery \(p. 981\)](#).

Amazon S3 also provides the [GET Bucket logging](#) API to retrieve logging configuration on a bucket. To delete the logging configuration, you send the PUT Bucket logging request with an empty BucketLoggingStatus.

```
<BucketLoggingStatus xmlns="http://doc.s3.amazonaws.com/2006-03-01">
</BucketLoggingStatus>
```

You can use either the Amazon S3 API or the AWS SDK wrapper libraries to enable logging on a bucket.

Using the AWS SDKs

.NET

The following C# example enables logging on a bucket. You must create two buckets, a source bucket and a target bucket. The example first updates the bucket ACL on the target bucket and grants the log delivery group the necessary permissions to write logs to the target bucket and then enables logging on the source bucket.

This example won't work on target buckets that use the bucket owner enforced setting for Object Ownership.

If the target bucket uses the bucket owner enforced setting for Object Ownership, you can't set bucket or object ACLs. You also can't include target grants in your [PutBucketLogging](#) configuration. You must use a bucket policy to grant access to the logging service principal (`logging.s3.amazonaws.com`). For more information, see [Permissions for log delivery \(p. 981\)](#).

Example

```
using Amazon;
using Amazon.S3;
using Amazon.S3.Model;
using System;
using System.Threading.Tasks;

namespace Amazon.DocSamples.S3
{
    class ServerAccessLoggingTest
    {
        private const string bucketName = "*** bucket name for which to enable logging ***";
        private const string targetBucketName = "*** bucket name where you want access logs stored ***";
        private const string logObjectKeyPrefix = "Logs";
        // Specify your bucket region (an example region is shown).
        private static readonly RegionEndpoint bucketRegion = RegionEndpoint.USWest2;
        private static IAmazonS3 client;

        public static void Main()
        {
            client = new AmazonS3Client(bucketRegion);
            EnableLoggingAsync().Wait();
        }

        private static async Task EnableLoggingAsync()
        {
```

```

        try
        {
            // Step 1 - Grant Log Delivery group permission to write log to the target
            await GrantPermissionsToWriteLogsAsync();
            // Step 2 - Enable logging on the source bucket.
            await EnableDisableLoggingAsync();
        }
        catch (AmazonS3Exception e)
        {
            Console.WriteLine("Error encountered on server. Message:{0}' when writing
an object", e.Message);
        }
        catch (Exception e)
        {
            Console.WriteLine("Unknown encountered on server. Message:{0}' when
writing an object", e.Message);
        }
    }

    private static async Task GrantPermissionsToWriteLogsAsync()
    {
        var bucketACL = new S3AccessControlList();
        var aclResponse = client.GetACL(new GetACLRequest { BucketName =
targetBucketName });
        bucketACL = aclResponse.AccessControlList;
        bucketACL.AddGrant(new S3Grantee { URI = "http://acs.amazonaws.com/groups/s3/
LogDelivery" }, S3Permission.WRITE);
        bucketACL.AddGrant(new S3Grantee { URI = "http://acs.amazonaws.com/groups/s3/
LogDelivery" }, S3Permission.READ_ACP);
        var setACLRequest = new PutACLRequest
        {
            AccessControlList = bucketACL,
            BucketName = targetBucketName
        };
        await client.PutACLAsync(setACLRequest);
    }

    private static async Task EnableDisableLoggingAsync()
    {
        var loggingConfig = new S3BucketLoggingConfig
        {
            TargetBucketName = targetBucketName,
            TargetPrefix = logObjectKeyPrefix
        };

        // Send request.
        var putBucketLoggingRequest = new PutBucketLoggingRequest
        {
            BucketName = bucketName,
            LoggingConfig = loggingConfig
        };
        await client.PutBucketLoggingAsync(putBucketLoggingRequest);
    }
}

```

Using the AWS CLI

We recommend that you create a dedicated logging bucket in each AWS Region that you have S3 buckets in. Then have the Amazon S3 access log delivered to that S3 bucket. For more information and examples, see [put-bucket-logging](#) in the *AWS CLI Reference*.

If the target bucket uses the bucket owner enforced setting for Object Ownership, you can't set bucket or object ACLs. You also can't include target grants in your [PutBucketLogging](#) configuration. You must use a

bucket policy to grant access to the logging service principal (`logging.s3.amazonaws.com`). For more information, see [Permissions for log delivery \(p. 981\)](#).

Example — Enable access logs with five buckets across two Regions

In this example, you have the following five buckets:

- 1-awsexamplebucket1-us-east-1
- 2-awsexamplebucket1-us-east-1
- 3-awsexamplebucket1-us-east-1
- 1-awsexamplebucket1-us-west-2
- 2-awsexamplebucket1-us-west-2

1. Create two logging buckets in the following Regions:

- awsexamplebucket1-logs-us-east-1
- awsexamplebucket1-logs-us-west-2

2. Then enable the Amazon S3 access logs as follows:

- 1-awsexamplebucket1-us-east-1 logs to the S3 bucket awsexamplebucket1-logs-us-east-1 with prefix 1-awsexamplebucket1-us-east-1
- 2-awsexamplebucket1-us-east-1 logs to the S3 bucket awsexamplebucket1-logs-us-east-1 with prefix 2-awsexamplebucket1-us-east-1
- 3-awsexamplebucket1-us-east-1 logs to the S3 bucket awsexamplebucket1-logs-us-east-1 with prefix 3-awsexamplebucket1-us-east-1
- 1-awsexamplebucket1-us-west-2 logs to the S3 bucket awsexamplebucket1-logs-us-west-2 with prefix 1-awsexamplebucket1-us-west-2
- 2-awsexamplebucket1-us-west-2 logs to the S3 bucket awsexamplebucket1-logs-us-west-2 with prefix 2-awsexamplebucket1-us-west-2

3. Grant permissions for server access log delivery using a bucket ACL or a bucket policy:

- **Update the bucket policy (Recommended)** – To grant permissions to the logging service principal, use `put-bucket-policy`:

```
aws s3api put-bucket-policy --bucket awsexamplebucket1-logs --policy file://policy.json
```

`Policy.json` is a JSON document in the current folder that contains the bucket policy. To use this bucket policy, replace the example values.

```
{  
    "Version": "2012-10-17",  
    "Statement": [  
        {  
            "Sid": "S3ServerAccessLogsPolicy",  
            "Effect": "Allow",  
            "Principal": {  
                "Service": "logging.s3.amazonaws.com"  
            },  
            "Action": [  
                "s3:PutObject"  
            ],  
            "Resource": "arn:aws:s3:::awsexamplebucket1-logs/*",  
            "Condition": {  
                "ArnLike": {  
                    "AWSRegion": "us-east-1"  
                }  
            }  
        }  
    ]  
}
```

```
        "aws:SourceArn": "arn:aws:s3:::SOURCE-BUCKET-NAME"  
    },  
    "StringEquals": {  
        "aws:SourceAccount": "SOURCE-ACCOUNT-ID"  
    }  
}  
]  
}
```

- **Update the bucket ACL** – To grant permissions to the S3 log delivery group, use `put-bucket-acl`.

```
aws s3api put-bucket-acl --bucket awsexamplebucket1-logs --grant-write  
    URI=http://acs.amazonaws.com/groups/s3/LogDelivery --grant-read-acp URI=http://  
    acs.amazonaws.com/groups/s3/LogDelivery
```

4. Then, apply the logging policy.

```
aws s3api put-bucket-logging --bucket awsexamplebucket1 --bucket-logging-status file://  
logging.json
```

`Logging.json` is a JSON document in the current folder that contains the logging configuration. If a bucket uses the bucket owner enforced setting for Object Ownership, your logging configuration can't contain target grants. For more information, see [Permissions for log delivery \(p. 981\)](#).

Example – Logging.json without target grants

The following example `Logging.json` file doesn't contain target grants and can be applied to a bucket that uses the bucket owner enforced setting for Object Ownership.

```
{  
    "LoggingEnabled": {  
        "TargetBucket": "awsexamplebucket1-logs",  
        "TargetPrefix": "awsexamplebucket1/"  
    }  
}
```

Example – Logging.json with target grants

The following example `Logging.json` file contains target grants.

If the target bucket uses the bucket owner enforced setting for Object Ownership, you can't include target grants in your `PutBucketLogging` configuration. For more information, see [Permissions for log delivery \(p. 981\)](#).

```
{  
    "LoggingEnabled": {  
        "TargetBucket": "awsexamplebucket1-logs",  
        "TargetPrefix": "awsexamplebucket1/",  
        "TargetGrants": [  
            {
```

```
        "Grantee": {
            "Type": "AmazonCustomerByEmail",
            "EmailAddress": "user@example.com"
        },
        "Permission": "FULL_CONTROL"
    }
}
}
```

5. Use a bash script to add access logging for all the buckets in your account.

Note

This script only works if all your buckets are in the same Region. If you have buckets in multiple Regions, you must adjust the script.

Example – Grant access with bucket policies and add logging for the buckets in your account

```
loggingBucket='awsexamplebucket1-logs'
region='us-west-2'

# Create Logging bucket
aws s3 mb s3://$loggingBucket --region $region

aws s3api put-bucket-policy --bucket $loggingBucket --policy file://policy.json

# List buckets in this account
buckets=$(aws s3 ls | awk '{print $3}')

# Put bucket logging on each bucket
for bucket in $buckets
do
printf '%{
"LoggingEnabled": {
    "TargetBucket": "%s",
    "TargetPrefix": "%s/"
}
}' "$loggingBucket" "$bucket" > logging.json
aws s3api put-bucket-logging --bucket $bucket --bucket-logging-status file://
logging.json
echo "$bucket done"
done

rm logging.json

echo "Complete"
```

Example – Grant access with bucket ACLs and add logging for the buckets in your account

```
loggingBucket='awsexamplebucket1-logs'
region='us-west-2'

# Create Logging bucket
aws s3 mb s3://$loggingBucket --region $region

aws s3api put-bucket-acl --bucket $loggingBucket --grant-write URI=http://
acs.amazonaws.com/groups/s3/LogDelivery --grant-read-acp URI=http://acs.amazonaws.com/
groups/s3/LogDelivery
```

```

# List buckets in this account
buckets=$(aws s3 ls | awk '{print $3}')"

# Put bucket logging on each bucket
for bucket in $buckets
    do printf '%{
    "LoggingEnabled": {
        "TargetBucket": "%s",
        "TargetPrefix": "%s/"
    }
}' "$loggingBucket" "$bucket" > logging.json
    aws s3api put-bucket-logging --bucket $bucket --bucket-logging-status file://
logging.json
    echo "$bucket done"
done

rm logging.json

echo "Complete"

```

Amazon S3 server access log format

Server access logging provides detailed records for the requests that are made to an Amazon S3 bucket. You can use server access logs for security and access audits, learn about your customer base, or understand your Amazon S3 bill. This section describes the format and other details about Amazon S3 server access log files.

Server access log files consist of a sequence of newline-delimited log records. Each log record represents one request and consists of space-delimited fields.

The following is an example log consisting of five log records.

```

79a59df900b949e55d96a1e698fbacedfd6e09d98eacf8f8d5218e7cd47ef2be
awsexamplebucket1 [06/Feb/2019:00:00:38 +0000] 192.0.2.3
79a59df900b949e55d96a1e698fbacedfd6e09d98eacf8f8d5218e7cd47ef2be 3E57427F3EXAMPLE
REST.GET.VERSIONING - "GET /awsexamplebucket1?versioning HTTP/1.1" 200 - 113 - 7 -
"--" "S3Console/0.4" - s91zHyrFp76ZVxRcpX9+5cjAnEH2ROUnkd2BHF1a6UkFVdtjf5mKR3/eTPFvsiP/
XVLi31234= SigV2 ECDHE-RSA-AES128-GCM-SHA256 AuthHeader awsexamplebucket1.s3.us-
west-1.amazonaws.com TLSV1.1
79a59df900b949e55d96a1e698fbacedfd6e09d98eacf8f8d5218e7cd47ef2be
awsexamplebucket1 [06/Feb/2019:00:00:38 +0000] 192.0.2.3
79a59df900b949e55d96a1e698fbacedfd6e09d98eacf8f8d5218e7cd47ef2be 891CE47D2EXAMPLE
REST.GET.LOGGING_STATUS - "GET /awsexamplebucket1?logging HTTP/1.1" 200 - 242
- 11 - "--" "S3Console/0.4" - 9vKBE6vMhrNiWHZmb2L0mXOcqPGzQOI5XLnCtZNPxev+Hf
+7tpT6sxDwDty4LHBuOZJG96N1234= SigV2 ECDHE-RSA-AES128-GCM-SHA256 AuthHeader
awsexamplebucket1.s3.us-west-1.amazonaws.com TLSV1.1
79a59df900b949e55d96a1e698fbacedfd6e09d98eacf8f8d5218e7cd47ef2be
awsexamplebucket1 [06/Feb/2019:00:00:38 +0000] 192.0.2.3
79a59df900b949e55d96a1e698fbacedfd6e09d98eacf8f8d5218e7cd47ef2be A1206F460EXAMPLE
REST.GET.BUCKETPOLICY - "GET /awsexamplebucket1?policy HTTP/1.1" 404
NoSuchBucketPolicy 297 - 38 - "--" "S3Console/0.4" - BNaBsXZQQDbssi6xMBdBU2sLt
+Yf5kZDmeBUP35sFoKa3sLLeMC78iwEIWxs99CRUrbS4n11234= SigV2 ECDHE-RSA-AES128-GCM-SHA256
AuthHeader awsexamplebucket1.s3.us-west-1.amazonaws.com TLSV1.1
79a59df900b949e55d96a1e698fbacedfd6e09d98eacf8f8d5218e7cd47ef2be
awsexamplebucket1 [06/Feb/2019:00:01:00 +0000] 192.0.2.3
79a59df900b949e55d96a1e698fbacedfd6e09d98eacf8f8d5218e7cd47ef2be 7B4A0FABBEXAMPLE
REST.GET.VERSIONING - "GET /awsexamplebucket1?versioning HTTP/1.1" 200 - 113 - 33 - "--"
"S3Console/0.4" - Ke1bUcazaN1jWuUlPJaxF64cQVpUEhoZKEG/hmy/gijN/I1DeWqDfFvnpybfEseEME/
u7ME1234= SigV2 ECDHE-RSA-AES128-GCM-SHA256 AuthHeader awsexamplebucket1.s3.us-
west-1.amazonaws.com TLSV1.1

```

```
79a59df900b949e55d96a1e698fbacedfd6e09d98eacf8f8d5218e7cd47ef2be
awsexamplebucket1 [06/Feb/2019:00:01:57 +0000] 192.0.2.3
79a59df900b949e55d96a1e698fbacedfd6e09d98eacf8f8d5218e7cd47ef2be
DD6CC733AEXAMPLE REST.PUT.OBJECT s3-dg.pdf "PUT /awsexamplebucket1/
s3-dg.pdf HTTP/1.1" 200 - - 4406583 41754 28 "-" "S3Console/0.4" -
10S62Zv81kBW7BB6SX4XJ48o6kpcl6LPwEoizzQ0xJd5qDSCTLX0TgS37kYUBKQW3+bPdrgr1234= SigV4 ECDHE-
RSA-AES128-SHA AuthHeader awsexamplebucket1.s3.us-west-1.amazonaws.com TLSV1.1
```

Note

Any field can be set to – to indicate that the data was unknown or unavailable, or that the field was not applicable to this request.

Topics

- [Log record fields \(p. 990\)](#)
- [Additional logging for copy operations \(p. 994\)](#)
- [Custom access log information \(p. 998\)](#)
- [Programming considerations for extensible server access log format \(p. 998\)](#)

Log record fields

The following list describes the log record fields.

Bucket Owner

The canonical user ID of the owner of the source bucket. The canonical user ID is another form of the AWS account ID. For more information about the canonical user ID, see [AWS account identifiers](#) in the *AWS General Reference*. For information about how to find the canonical user ID for your account, see [Finding the canonical user ID for your AWS account](#).

Example entry

```
79a59df900b949e55d96a1e698fbacedfd6e09d98eacf8f8d5218e7cd47ef2be
```

Bucket

The name of the bucket that the request was processed against. If the system receives a malformed request and cannot determine the bucket, the request will not appear in any server access log.

Example entry

```
awsexamplebucket1
```

Time

The time at which the request was received; these dates and times are in Coordinated Universal Time (UTC). The format, using `strftime()` terminology, is as follows: [%d/%b/%Y:%H:%M:%S %z]

Example entry

```
[06/Feb/2019:00:00:38 +0000]
```

Remote IP

The apparent internet address of the requester. Intermediate proxies and firewalls might obscure the actual address of the machine making the request.

Example entry

```
192.0.2.3
```

Requester

The canonical user ID of the requester, or a – for unauthenticated requests. If the requester was an IAM user, this field returns the requester's IAM user name along with the AWS root account that the IAM user belongs to. This identifier is the same one used for access control purposes.

Example entry

```
79a59df900b949e55d96a1e698fbacedfd6e09d98eacf8f8d5218e7cd47ef2be
```

Request ID

A string generated by Amazon S3 to uniquely identify each request.

Example entry

```
3E57427F33A59F07
```

Operation

The operation listed here is declared as SOAP.*operation*, REST.*HTTP_method.resource_type*, WEBSITE.*HTTP_method.resource_type*, or BATCH.DELETE.OBJECT, or S3.action.resource_type for [Lifecycle and logging \(p. 719\)](#).

Example entry

```
REST.PUT.OBJECT
```

Key

The "key" part of the request, URL encoded, or "-" if the operation does not take a key parameter.

Example entry

```
/photos/2019/08/puppy.jpg
```

Request-URI

The Request-URI part of the HTTP request message.

Example Entry

```
"GET /awsexamplebucket1/photos/2019/08/puppy.jpg?x-foo=bar HTTP/1.1"
```

HTTP status

The numeric HTTP status code of the response.

Example entry

```
200
```

Error Code

The Amazon S3 [Error code \(p. 1202\)](#), or "-" if no error occurred.

Example entry

```
NoSuchBucket
```

Bytes Sent

The number of response bytes sent, excluding HTTP protocol overhead, or "-" if zero.

Example entry

```
2662992
```

Object Size

The total size of the object in question.

Example entry

```
3462992
```

Total Time

The number of milliseconds the request was in flight from the server's perspective. This value is measured from the time your request is received to the time that the last byte of the response is sent. Measurements made from the client's perspective might be longer due to network latency.

Example entry

```
70
```

Turn-Around Time

The number of milliseconds that Amazon S3 spent processing your request. This value is measured from the time the last byte of your request was received until the time the first byte of the response was sent.

Example entry

```
10
```

Referer

The value of the HTTP Referer header, if present. HTTP user-agents (for example, browsers) typically set this header to the URL of the linking or embedding page when making a request.

Example entry

```
"http://www.amazon.com/webservices"
```

User-Agent

The value of the HTTP User-Agent header.

Example entry

```
"curl/7.15.1"
```

Version Id

The version ID in the request, or "-" if the operation does not take a `versionId` parameter.

Example entry

```
3HL4kqtJvjVBH40Nrjfkd
```

Host Id

The x-amz-id-2 or Amazon S3 extended request ID.

Example entry

```
s9lzMHyFp76ZVxRcpX9+5cjAnEH2ROuNkd2BHfIa6UkFVdtjf5mKR3/eTPFvsip/XV/VLi31234=
```

Signature Version

The signature version, `SigV2` or `SigV4`, that was used to authenticate the request or a – for unauthenticated requests.

Example entry

```
SigV2
```

Cipher Suite

The Secure Sockets Layer (SSL) cipher that was negotiated for HTTPS request or a – for HTTP.

Example entry

```
ECDHE-RSA-AES128-GCM-SHA256
```

Authentication Type

The type of request authentication used, `AuthHeader` for authentication headers, `QueryString` for query string (pre-signed URL) or a – for unauthenticated requests.

Example entry

```
AuthHeader
```

Host Header

The endpoint used to connect to Amazon S3.

Example entry

```
s3.us-west-2.amazonaws.com
```

Some older Regions support legacy endpoints. You may see these endpoints in your server access logs or AWS CloudTrail logs. For more information, see [Legacy endpoints \(p. 1180\)](#). For a complete list of Amazon S3 Regions and endpoints, see [Amazon S3 endpoints and quotas](#) in the *Amazon Web Services General Reference*.

TLS version

The Transport Layer Security (TLS) version negotiated by the client. The value is one of following: `TLSv1`, `TLSv1.1`, `TLSv1.2`; or – if TLS wasn't used.

Example entry

```
TL Sv1.2
```

Access Point ARN (Amazon Resource Name)

The Amazon Resource Name (ARN) of the access point of the request. If access point ARN is malformed or not used, the field will contain a '-'. For more information on access points, see [Using access points \(p. 310\)](#). For more information on ARNs, see the topic on [Amazon Resource Name \(ARN\)](#) in the *AWS Reference Guide*.

Example entry

```
arn:aws:s3:us-east-1:123456789012:accesspoint/example-AP
```

Additional logging for copy operations

A copy operation involves a `GET` and a `PUT`. For that reason, we log two records when performing a copy operation. The previous section describes the fields related to the `PUT` part of the operation. The following list describes the fields in the record that relate to the `GET` part of the copy operation.

Bucket Owner

The canonical user ID of the bucket that stores the object being copied. The canonical user ID is another form of the AWS account ID. For more information about the canonical user ID, see [AWS account identifiers](#) in the *AWS General Reference*. For information about how to find the canonical user ID for your account, see [Finding the canonical user ID for your AWS account](#).

Example entry

```
79a59df900b949e55d96a1e698fbacedfd6e09d98eacf8f8d5218e7cd47ef2be
```

Bucket

The name of the bucket that stores the object being copied.

Example entry

```
awsexamplebucket1
```

Time

The time at which the request was received; these dates and times are in Coordinated Universal time (UTC). The format, using `strftime()` terminology, is as follows: `[%d/%B/%Y:%H:%M:%S %z]`

Example entry

```
[06/Feb/2019:00:00:38 +0000]
```

Remote IP

The apparent internet address of the requester. Intermediate proxies and firewalls might obscure the actual address of the machine making the request.

Example entry

```
192.0.2.3
```

Requester

The canonical user ID of the requester, or a - for unauthenticated requests. If the requester was an IAM user, this field will return the requester's IAM user name along with the AWS root account that the IAM user belongs to. This identifier is the same one used for access control purposes.

Example entry

```
79a59df900b949e55d96a1e698fbacedfd6e09d98eacf8f8d5218e7cd47ef2be
```

Request ID

A string generated by Amazon S3 to uniquely identify each request.

Example entry

```
3E57427F33A59F07
```

Operation

The operation listed here is declared as SOAP.*operation*, REST.*HTTP_method.resource_type*, WEBSITE.*HTTP_method.resource_type*, or BATCH.DELETE.OBJECT.

Example entry

```
REST.COPY.OBJECT_GET
```

Key

The "key" of the object being copied or "-" if the operation does not take a key parameter.

Example entry

```
/photos/2019/08/puppy.jpg
```

Request-URI

The Request-URI part of the HTTP request message.

Example entry

```
"GET /awsexamplebucket1/photos/2019/08/puppy.jpg?x-foo=bar"
```

HTTP status

The numeric HTTP status code of the GET portion of the copy operation.

Example entry

```
200
```

Error Code

The Amazon S3 [Error code \(p. 1202\)](#), of the GET portion of the copy operation or "-" if no error occurred.

Example entry

```
NoSuchBucket
```

Bytes Sent

The number of response bytes sent, excluding HTTP protocol overhead, or "-" if zero.

Example entry

```
2662992
```

Object Size

The total size of the object in question.

Example entry

```
3462992
```

Total Time

The number of milliseconds the request was in flight from the server's perspective. This value is measured from the time your request is received to the time that the last byte of the response is sent. Measurements made from the client's perspective might be longer due to network latency.

Example entry

```
70
```

Turn-Around Time

The number of milliseconds that Amazon S3 spent processing your request. This value is measured from the time the last byte of your request was received until the time the first byte of the response was sent.

Example entry

```
10
```

Referer

The value of the HTTP Referer header, if present. HTTP user-agents (for example, browsers) typically set this header to the URL of the linking or embedding page when making a request.

Example entry

```
"http://www.amazon.com/webservices"
```

User-Agent

The value of the HTTP User-Agent header.

Example entry

```
"curl/7.15.1"
```

Version Id

The version ID of the object being copied or "-" if the `x-amz-copy-source` header didn't specify a `versionId` parameter as part of the copy source.

Example Entry

```
3HL4kqtJvjVBH40Nrjfkd
```

Host Id

The x-amz-id-2 or Amazon S3 extended request ID.

Example entry

```
s9lzMHyFp76ZVxRcpX9+5cjAnEH2ROuNkd2BHfIa6UkFVdtjf5mKR3/eTPFvsip/XV/VLi31234=
```

Signature Version

The signature version, SigV2 or SigV4, that was used to authenticate the request or a – for unauthenticated requests.

Example entry

```
SigV2
```

Cipher Suite

The Secure Sockets Layer (SSL) cipher that was negotiated for HTTPS request or a – for HTTP.

Example entry

```
ECDHE-RSA-AES128-GCM-SHA256
```

Authentication Type

The type of request authentication used, AuthHeader for authentication headers, QueryString for query string (presigned URL) or a – for unauthenticated requests.

Example entry

```
AuthHeader
```

Host Header

The endpoint used to connect to Amazon S3.

Example entry

```
s3.us-west-2.amazonaws.com
```

Some older Regions support legacy endpoints. You might see these endpoints in your server access logs or AWS CloudTrail logs. For more information, see [Legacy endpoints \(p. 1180\)](#). For a complete list of Amazon S3 Regions and endpoints, see [Amazon S3 endpoints and quotas](#) in the *Amazon Web Services General Reference*.

TLS version

The Transport Layer Security (TLS) version negotiated by the client. The value is one of following: TLSv1, TLSv1.1, TLSv1.2; or – if TLS wasn't used.

Example entry

```
TLSv1.2
```

Access Point ARN (Amazon Resource Name)

The Amazon Resource Name (ARN) of the access point of the request. If access point ARN is malformed or not used, the field will contain a '-'. For more information on access points, see [Using access points \(p. 310\)](#). For more information on ARNs, see the topic on [Amazon Resource Name \(ARN\)](#) in the *AWS Reference Guide*.

Example entry

```
arn:aws:s3:us-east-1:123456789012:accesspoint/example-AP
```

Custom access log information

You can include custom information to be stored in the access log record for a request. To do this, add a custom query-string parameter to the URL for the request. Amazon S3 ignores query-string parameters that begin with "x-", but includes those parameters in the access log record for the request, as part of the Request-URI field of the log record.

For example, a GET request for "s3.amazonaws.com/awsexamplebucket1/photos/2019/08/puppy.jpg?x-user=johndoe" works the same as the request for "s3.amazonaws.com/awsexamplebucket1/photos/2019/08/puppy.jpg", except that the "x-user=johndoe" string is included in the Request-URI field for the associated log record. This functionality is available in the REST interface only.

Programming considerations for extensible server access log format

Occasionally we might extend the access log record format by adding new fields to the end of each line. Therefore, you should write any code that parses server access logs to handle trailing fields that it might not understand.

Deleting Amazon S3 log files

An Amazon S3 bucket with server access logging enabled can accumulate many server log objects over time. Your application might need these access logs for a specific period after they are created, and after that, you might want to delete them. You can use Amazon S3 Lifecycle configuration to set rules so that S3 automatically queues these objects for deletion at the end of their life.

You can define a lifecycle configuration for a subset of objects in your S3 bucket by using a shared prefix (that is, objects that have names that begin with a common string). If you specified a prefix in your server access logging configuration, you can set a lifecycle configuration rule to delete log objects that have that prefix.

For example, if your log objects have the prefix logs/, you can set a lifecycle configuration rule to delete all objects in the bucket that have the prefix logs/ after a specified period of time.

For more information about lifecycle configuration, see [Managing your storage lifecycle \(p. 701\)](#).

For general information about server access logging, see [Logging requests using server access logging \(p. 978\)](#).

Using Amazon S3 access logs to identify requests

You can identify Amazon S3 requests using Amazon S3 access logs.

Note

- We recommend that you use AWS CloudTrail data events instead of Amazon S3 access logs. CloudTrail data events are easier to set up and contain more information. For more information, see [Identifying Amazon S3 requests using CloudTrail \(p. 972\)](#).
- Depending on how many access requests you get, it might require more resources or time to analyze your logs.

Topics

- [Querying access logs for requests using Amazon Athena \(p. 999\)](#)
- [Identifying Signature Version 2 requests using Amazon S3 access logs \(p. 1001\)](#)
- [Identifying object access requests using Amazon S3 access logs \(p. 1001\)](#)

Querying access logs for requests using Amazon Athena

You can identify Amazon S3 requests with Amazon S3 access logs using Amazon Athena.

Amazon S3 stores server access logs as objects in an S3 bucket. It is often easier to use a tool that can analyze the logs in Amazon S3. Athena supports analysis of S3 objects and can be used to query Amazon S3 access logs.

Example

The following example shows how you can query Amazon S3 server access logs in Amazon Athena.

Note

To specify an Amazon S3 location in an Athena query, you need to format the *target* bucket name and *target* prefix where your logs are delivered as an S3 URI, as follows: s3://*DOC-EXAMPLE-BUCKET1*-logs/prefix/

1. Open the Athena console at <https://console.aws.amazon.com/athena/>.
2. In the Query Editor, run a command similar to the following.

```
create database s3_access_logs_db
```

Note

It's a best practice to create the database in the same AWS Region as your S3 bucket.

3. In the Query Editor, run a command similar to the following to create a table schema in the database that you created in step 2. The STRING and BIGINT data type values are the access log properties. You can query these properties in Athena. For LOCATION, enter the S3 bucket and prefix path as noted earlier.

```
CREATE EXTERNAL TABLE `s3_access_logs_db.mybucket_logs`(`  
`bucketowner` STRING,  
`bucket_name` STRING,  
`requestdatetime` STRING,  
`remoteip` STRING,  
`requester` STRING,  
`requestid` STRING,  
`operation` STRING,  
`key` STRING,  
`request_uri` STRING,  
`httpstatus` STRING,  
`errorcode` STRING,
```

4. In the navigation pane, under **Database**, choose your database.
 5. Under **Tables**, choose **Preview table** next to your table name.

In the **Results** pane, you should see data from the server access logs, such as `bucketowner`, `bucket`, `requestdatetime`, and so on. This means that you successfully created the Athena table. You can now query the Amazon S3 server access logs.

Example — Show who deleted an object and when (timestamp, IP address, and IAM user)

```
SELECT RequestDateTime, RemoteIP, Requester, Key  
FROM s3_access_logs_db.mybucket_logs  
WHERE key = 'images/picture.jpg' AND operation like '%DELETE%';
```

Example — Show all operations that were performed by an IAM user

```
SELECT *
FROM s3_access_logs_db.mybucket_logs
WHERE requester='arn:aws:iam::123456789123:user/user_name';
```

Example — Show all operations that were performed on an object in a specific time period

```
SELECT *
FROM s3_access_logs_db.mybucket_logs
WHERE Key='prefix/images/picture.jpg'
    AND parse_datetime(RequestDateTime, 'dd/MMM/yyyy:HH:mm:ss z')
    BETWEEN parse_datetime('2017-02-18:07:00:00', 'yyyy-MM-dd:HH:mm:ss')
    AND parse_datetime('2017-02-18:08:00:00', 'yyyy-MM-dd:HH:mm:ss');
```

Example — Show how much data was transferred by a specific IP address in a specific time period

```
SELECT SUM(bytessent) AS uploadTotal,  
       SUM(objectsize) AS downloadTotal,  
       SUM(bytessent + objectsize) AS Total  
  FROM s3_access_logs_db.mybucket_logs  
 WHERE RemoteIP='1.2.3.4'  
   AND parse_datetime(RequestDateTime, 'dd/MMM/yyyy:HH:mm:ss Z')  
   BETWEEN parse_datetime('2017-06-01', 'yyyy-MM-dd')  
     AND parse_datetime('2017-07-01', 'yyyy-MM-dd');
```

Note

To reduce the time that you retain your log, you can create an Amazon S3 Lifecycle policy for your server access logs bucket. Configure the lifecycle policy to remove log files periodically. Doing so reduces the amount of data that Athena analyzes for each query. For more information, see [Setting lifecycle configuration on a bucket \(p. 708\)](#).

Identifying Signature Version 2 requests using Amazon S3 access logs

Amazon S3 support for Signature Version 2 will be turned off (deprecated). After that, Amazon S3 will no longer accept requests that use Signature Version 2, and all requests must use *Signature Version 4* signing. You can identify Signature Version 2 access requests using Amazon S3 access logs.

Note

- We recommend that you use AWS CloudTrail data events instead of Amazon S3 access logs. CloudTrail data events are easier to set up and contain more information. For more information, see [Identifying Amazon S3 Signature Version 2 requests using CloudTrail \(p. 974\)](#).

Example — Show all requesters that are sending Signature Version 2 traffic

```
SELECT requester, Sigv, Count(Sigv) as SigCount  
  FROM s3_access_logs_db.mybucket_logs  
 GROUP BY requester, Sigv;
```

Identifying object access requests using Amazon S3 access logs

You can use queries on Amazon S3 server access logs to identify Amazon S3 object access requests, for operations such as GET, PUT, and DELETE, and discover further information about those requests.

The following Amazon Athena query example shows how to get all PUT object requests for Amazon S3 from the server access log.

Example — Show all requesters that are sending PUT object requests in a certain period

```
SELECT Bucket, Requester, RemoteIP, Key, HTTPStatus, ErrorCode, RequestDateTime  
  FROM s3_access_logs_db  
 WHERE Operation='REST.PUT.OBJECT' AND  
       parse_datetime(RequestDateTime, 'dd/MMM/yyyy:HH:mm:ss Z')  
   BETWEEN parse_datetime('2019-07-01:00:42:42', 'yyyy-MM-dd:HH:mm:ss')  
     AND
```

```
parse_datetime('2019-07-02:00:42:42', 'yyyy-MM-dd:HH:mm:ss')
```

The following Amazon Athena query example shows how to get all GET object requests for Amazon S3 from the server access log.

Example — Show all requesters that are sending GET object requests in a certain period

```
SELECT Bucket, Requester, RemoteIP, Key, HTTPStatus, ErrorCode, RequestDateTime
FROM s3_access_logs_db
WHERE Operation='REST.GET.OBJECT' AND
parse_datetime(RequestDateTime,'dd/MMM/yyyy:HH:mm:ss Z')
BETWEEN parse_datetime('2019-07-01:00:42:42','yyyy-MM-dd:HH:mm:ss')
AND
parse_datetime('2019-07-02:00:42:42','yyyy-MM-dd:HH:mm:ss')
```

The following Amazon Athena query example shows how to get all anonymous requests to your S3 buckets from the server access log.

Example — Show all anonymous requesters that are making requests to a bucket in a certain period

```
SELECT Bucket, Requester, RemoteIP, Key, HTTPStatus, ErrorCode, RequestDateTime
FROM s3_access_logs_db.mybucket_logs
WHERE Requester IS NULL AND
parse_datetime(RequestDateTime,'dd/MMM/yyyy:HH:mm:ss Z')
BETWEEN parse_datetime('2019-07-01:00:42:42','yyyy-MM-dd:HH:mm:ss')
AND
parse_datetime('2019-07-02:00:42:42','yyyy-MM-dd:HH:mm:ss')
```

Note

- You can modify the date range as needed to suit your needs.
- These query examples might also be useful for security monitoring. You can review the results for PutObject or GetObject calls from unexpected or unauthorized IP addresses/requesters and for identifying any anonymous requests to your buckets.
- This query only retrieves information from the time at which logging was enabled.
- If you are using Amazon S3 AWS CloudTrail logs, see [Identifying access to S3 objects using CloudTrail \(p. 977\)](#).

Monitoring metrics with Amazon CloudWatch

Amazon CloudWatch metrics for Amazon S3 can help you understand and improve the performance of applications that use Amazon S3. There are several ways that you can use CloudWatch with Amazon S3.

Daily storage metrics for buckets

Monitor bucket storage using CloudWatch, which collects and processes storage data from Amazon S3 into readable, daily metrics. These storage metrics for Amazon S3 are reported once per day and are provided to all customers at no additional cost.

Request metrics

Monitor Amazon S3 requests to quickly identify and act on operational issues. The metrics are available at 1-minute intervals after some latency for processing. These CloudWatch metrics

are billed at the same rate as the Amazon CloudWatch custom metrics. For information about CloudWatch pricing, see [Amazon CloudWatch pricing](#). To learn how to opt in to getting these metrics, see [CloudWatch metrics configurations \(p. 1011\)](#).

When enabled, request metrics are reported for all object operations. By default, these 1-minute metrics are available at the Amazon S3 bucket level. You can also define a filter for the metrics using a shared prefix, object tag, or access point:

- **Access point** – Access points are named network endpoints that are attached to buckets and simplify managing data access at scale for shared datasets in S3. With the access point filter, you can gain insights into your access point usage. For more information about access points, see [Monitoring and logging access points \(p. 311\)](#).
- **Prefix** – Although the Amazon S3 data model is a flat structure, you can use prefixes to infer a hierarchy. A prefix is similar to a directory name that enables you to group similar objects together in a bucket. The S3 console supports prefixes with the concept of folders. If you filter by prefix, objects that have the same prefix are included in the metrics configuration. For more information about prefixes, see [Organizing objects using prefixes \(p. 243\)](#).
- **Tags** – Tags are key-value name pairs that you can add to objects. Tags help you find and organize objects easily. You can also use tags as a filter for metrics configurations so that only objects with those tags are included in the metrics configuration. For more information about object tags, see [Categorizing your storage using tags \(p. 825\)](#).

To align these metrics to specific business applications, workflows, or internal organizations, you can filter on a shared prefix, object tag, or access point.

Replication metrics

Replication metrics – Monitor the total number of S3 API operations that are pending replication, the total size of objects pending replication, and the maximum replication time to the destination Region. Replication rules that have S3 Replication Time Control (S3 RTC) or S3 replication metrics enabled will publish replication metrics.

For more information, see [Monitoring progress with replication metrics and Amazon S3 event notifications \(p. 805\)](#) or [Meeting compliance requirements using S3 Replication Time Control \(S3 RTC\) \(p. 807\)](#).

Amazon S3 Storage Lens metrics

You can publish S3 Storage Lens usage and activity metrics to Amazon CloudWatch to create a unified view of your operational health in CloudWatch [dashboards](#). S3 Storage Lens metrics are available in the AWS/S3/Storage-Lens namespace. The CloudWatch publishing option is available for S3 Storage Lens dashboards upgraded to *advanced metrics and recommendations*. You can enable the CloudWatch publishing option for a new or existing dashboard configuration in S3 Storage Lens.

For more information, see [Monitor S3 Storage Lens metrics in CloudWatch \(p. 1068\)](#).

All CloudWatch statistics are retained for a period of 15 months so that you can access historical information and gain a better perspective on how your web application or service is performing. For more information, see [What is Amazon CloudWatch?](#) in the *Amazon CloudWatch User Guide*.

For more information, see the following topics.

Topics

- [Metrics and dimensions \(p. 1004\)](#)
- [Accessing CloudWatch metrics \(p. 1011\)](#)
- [CloudWatch metrics configurations \(p. 1011\)](#)

Metrics and dimensions

The storage metrics and dimensions that Amazon S3 sends to CloudWatch are listed below.

Topics

- [Amazon S3 daily storage metrics for buckets in CloudWatch \(p. 1004\)](#)
- [Amazon S3 request metrics in CloudWatch \(p. 1005\)](#)
- [Amazon S3 replication metrics in CloudWatch \(p. 1007\)](#)
- [Amazon S3 Storage Lens metrics in CloudWatch \(p. 1008\)](#)
- [Amazon S3 on Outposts metrics in CloudWatch \(p. 1008\)](#)
- [Amazon S3 dimensions in CloudWatch \(p. 1008\)](#)
- [Amazon S3 Storage Lens dimensions in CloudWatch \(p. 1010\)](#)

Amazon S3 daily storage metrics for buckets in CloudWatch

The AWS/S3 namespace includes the following daily storage metrics for buckets.

Metric	Description
BucketSizeBytes	<p>The amount of data in bytes stored in a bucket in the STANDARD storage class, INTELLIGENT_TIERING storage class, Standard-Infrequent Access (STANDARD_IA) storage class, OneZone-Infrequent Access (ONEZONE_IA), Reduced Redundancy Storage (RRS) class, S3 Glacier Instant Retrieval storage class, Deep Archive Storage (S3 Glacier Deep Archive) class or, S3 Glacier Flexible Retrieval (GLACIER) storage class. This value is calculated by summing the size of all objects and metadata in the bucket (both current and noncurrent objects), including the size of all parts for all incomplete multipart uploads to the bucket.</p> <p>Valid storage type filters: StandardStorage, IntelligentTieringFASTorage, IntelligentTieringIAStorage, IntelligentTieringAAStorage, IntelligentTieringAIAStorage, IntelligentTieringDAAStorage, StandardIAStorage, StandardIASizeOverhead, StandardIAObjectOverhead, OneZoneIAStorage, OneZoneIASizeOverhead, ReducedRedundancyStorage, GlacierInstantRetrievalSizeOverhead, GlacierInstantRetrievalStorage, GlacierStorage, GlacierStagingStorage, GlacierObjectOverhead, GlacierS3ObjectOverhead, DeepArchiveStorage, DeepArchiveObjectOverhead, DeepArchiveS3ObjectOverhead and, DeepArchiveStagingStorage (see the StorageType dimension)</p> <p>Units: Bytes</p> <p>Valid statistics: Average</p>
NumberOfObjects	<p>The total number of objects stored in a bucket for all storage classes. This value is calculated by counting all objects in the bucket (both current and noncurrent objects) and the total number of parts for all incomplete multipart uploads to the bucket.</p> <p>Valid storage type filters: AllStorageTypes (see the StorageType dimension)</p>

Metric	Description
	Units: Count Valid statistics: Average

Amazon S3 request metrics in CloudWatch

The AWS/S3 namespace includes the following request metrics.

Metric	Description
AllRequests	The total number of HTTP requests made to an Amazon S3 bucket, regardless of type. If you're using a metrics configuration with a filter, then this metric only returns the HTTP requests that meet the filter's requirements. Units: Count Valid statistics: Sum
GetRequests	The number of HTTP GET requests made for objects in an Amazon S3 bucket. This doesn't include list operations. Units: Count Valid statistics: Sum Note Paginated list-oriented requests, like List Multipart Uploads , List Parts , Get Bucket Object versions , and others, are not included in this metric.
PutRequests	The number of HTTP PUT requests made for objects in an Amazon S3 bucket. Units: Count Valid statistics: Sum
DeleteRequests	The number of HTTP DELETE requests made for objects in an Amazon S3 bucket. This also includes Delete Multiple Objects requests. This metric shows the number of requests, not the number of objects deleted. Units: Count Valid statistics: Sum
HeadRequests	The number of HTTP HEAD requests made to an Amazon S3 bucket. Units: Count Valid statistics: Sum
PostRequests	The number of HTTP POST requests made to an Amazon S3 bucket. Units: Count Valid statistics: Sum

Metric	Description
	<p>Note Delete Multiple Objects and SELECT Object Content requests are not included in this metric.</p>
SelectRequests	<p>The number of Amazon S3 SELECT Object Content requests made for objects in an Amazon S3 bucket.</p> <p>Units: Count</p> <p>Valid statistics: Sum</p>
SelectBytesScanned	<p>The number of bytes of data scanned with Amazon S3 SELECT Object Content requests in an Amazon S3 bucket.</p> <p>Units: Bytes</p> <p>Valid statistics: Average (bytes per request), Sum (bytes per period), Sample Count, Min, Max (same as p100), any percentile between p0.0 and p99.9</p>
SelectBytesReturned	<p>The number of bytes of data returned with Amazon S3 SELECT Object Content requests in an Amazon S3 bucket.</p> <p>Units: Bytes</p> <p>Valid statistics: Average (bytes per request), Sum (bytes per period), Sample Count, Min, Max (same as p100), any percentile between p0.0 and p99.9</p>
ListRequests	<p>The number of HTTP requests that list the contents of a bucket.</p> <p>Units: Count</p> <p>Valid statistics: Sum</p>
BytesDownloaded	<p>The number of bytes downloaded for requests made to an Amazon S3 bucket, where the response includes a body.</p> <p>Units: Bytes</p> <p>Valid statistics: Average (bytes per request), Sum (bytes per period), Sample Count, Min, Max (same as p100), any percentile between p0.0 and p99.9</p>
BytesUploaded	<p>The number of bytes uploaded that contain a request body, made to an Amazon S3 bucket.</p> <p>Units: Bytes</p> <p>Valid statistics: Average (bytes per request), Sum (bytes per period), Sample Count, Min, Max (same as p100), any percentile between p0.0 and p99.9</p>
4xxErrors	<p>The number of HTTP 4xx client error status code requests made to an Amazon S3 bucket with a value of either 0 or 1. The <code>average</code> statistic shows the error rate, and the <code>sum</code> statistic shows the count of that type of error, during each period.</p> <p>Units: Count</p> <p>Valid statistics: Average (reports per request), Sum (reports per period), Min, Max, Sample Count</p>

Metric	Description
5xxErrors	<p>The number of HTTP 5xx server error status code requests made to an Amazon S3 bucket with a value of either 0 or 1. The <code>average</code> statistic shows the error rate, and the <code>sum</code> statistic shows the count of that type of error, during each period.</p> <p>Units: Counts</p> <p>Valid statistics: Average (reports per request), Sum (reports per period), Min, Max, Sample Count</p>
FirstByteLatency	<p>The per-request time from the complete request being received by an Amazon S3 bucket to when the response starts to be returned.</p> <p>Units: Milliseconds</p> <p>Valid statistics: Average, Sum, Min, Max(same as p100), Sample Count, any percentile between p0.0 and p100</p>
TotalRequestLatency	<p>The elapsed per-request time from the first byte received to the last byte sent to an Amazon S3 bucket. This includes the time taken to receive the request body and send the response body, which is not included in <code>FirstByteLatency</code>.</p> <p>Units: Milliseconds</p> <p>Valid statistics: Average, Sum, Min, Max(same as p100), Sample Count, any percentile between p0.0 and p100</p>

Amazon S3 replication metrics in CloudWatch

You can monitor the progress of replication with S3 replication metrics by tracking bytes pending, operations pending, and replication latency. For more information, see [Monitoring progress with replication metrics](#).

Note

You can enable alarms for your replication metrics on Amazon CloudWatch. When you set up alarms for your replication metrics, set the **Missing data treatment** field to **Treat missing data as ignore (maintain the alarm state)**.

Metric	Description
ReplicationLatency	<p>The maximum number of seconds by which the replication destination Region is behind the source Region for a given replication rule.</p> <p>Units: Seconds</p> <p>Valid statistics: Max</p>
BytesPendingReplication	<p>The total number of bytes of objects pending replication for a given replication rule.</p> <p>Units: Bytes</p> <p>Valid statistics: Max</p>
OperationsPendingReplication	<p>The number of operations pending replication for a given replication rule.</p>

Metric	Description
	Units: Counts Valid statistics: Max

Amazon S3 Storage Lens metrics in CloudWatch

You can publish S3 Storage Lens usage and activity metrics to Amazon CloudWatch to create a unified view of your operational health in CloudWatch dashboards. S3 Storage Lens metrics are published to the AWS/S3/Storage-Lens namespace in CloudWatch. The CloudWatch publishing option is available for S3 Storage Lens dashboards upgraded to advanced metrics and recommendations.

For a list of S3 Storage Lens metrics published to CloudWatch, see [Amazon S3 Storage Lens metrics glossary \(p. 1081\)](#). For a complete list of dimensions, see [Dimensions \(p. 1070\)](#).

Amazon S3 on Outposts metrics in CloudWatch

The S3Outposts namespace includes the following metrics for Amazon S3 on Outposts buckets. You can monitor the total number of S3 on Outposts bytes provisioned, the total free bytes available for objects, and the total size of all objects for a given bucket.

Note

S3 on Outposts supports only the following metrics, and no other Amazon S3 metrics. Because S3 on Outposts has fixed capacity, you can create CloudWatch alerts that alert you when your storage utilization exceeds a certain threshold.

Metric	Description
OutpostTotalBytes	The total provisioned capacity in bytes for an Outpost. Units: Bytes Period: 5 minutes
OutpostFreeBytes	The count of free bytes available on an Outpost to store customer data. Units: Bytes Period: 5 minutes
BucketUsedBytes	The total size of all objects for the given bucket. Units: Counts Period: 5 minutes
AccountUsedBytes	The total size of all objects for the specified Outposts account. Units: Bytes Period: 5 minutes

Amazon S3 dimensions in CloudWatch

The following dimensions are used to filter Amazon S3 metrics.

Dimension	Description
BucketName	This dimension filters the data that you request for the identified bucket only.
StorageType	<p>This dimension filters the data that you have stored in a bucket by the following types of storage:</p> <ul style="list-style-type: none"> • StandardStorage – The number of bytes used for objects in the STANDARD storage class. • IntelligentTieringAAStorage – The number of bytes used for objects in the Archive Access tier of the INTELLIGENT_TIERING storage class. • IntelligentTieringAIAStorage – The number of bytes used for objects in the Archive Instant Access tier of the INTELLIGENT_TIERING storage class. • IntelligentTieringDAAStorage – The number of bytes used for objects in the Deep Archive Access tier of the INTELLIGENT_TIERING storage class. • IntelligentTieringFAStorage – The number of bytes used for objects in the Frequent Access tier of the INTELLIGENT_TIERING storage class. • IntelligentTieringIAStorage – The number of bytes used for objects in the Infrequent Access tier of the INTELLIGENT_TIERING storage class. • StandardIASStorage – The number of bytes used for objects in the Standard-Infrequent Access (STANDARD_IA) storage class. • StandardIASizeOverhead – The number of bytes used for objects smaller than 128 KB in size in the STANDARD_IA storage class. • IntAAObjectOverhead – For each object in the INTELLIGENT_TIERING storage class in the Archive Access tier, S3 Glacier adds 32 KB of storage for index and related metadata. This extra data is necessary to identify and restore your object. You are charged S3 Glacier rates for this additional storage. • IntAAS3ObjectOverhead – For each object in the INTELLIGENT_TIERING storage class in the Archive Access tier, Amazon S3 uses 8 KB of storage for the name of the object and other metadata. You are charged STANDARD rates for this additional storage. • IntDAAObjectOverhead – For each object in the INTELLIGENT_TIERING storage class in the Deep Archive Access tier, S3 Glacier adds 32 KB of storage for index and related metadata. This extra data is necessary to identify and restore your object. You are charged S3 Glacier Deep Archive storage rates for this additional storage. • IntDAAS3ObjectOverhead – For each object in the INTELLIGENT_TIERING storage class in the Deep Archive Access tier, Amazon S3 adds 8 KB of storage for index and related metadata. This extra data is necessary to identify and restore your object. You are charged STANDARD rates for this additional storage. • OneZoneIASStorage – The number of bytes used for objects in the S3 One Zone-Infrequent Access (ONEZONE_IA) storage class.

Dimension	Description
	<ul style="list-style-type: none"> • OneZoneIASizeOverhead – The number of bytes used for objects smaller than 128 KB in size in the ONEZONE_IA storage class. • ReducedRedundancyStorage – The number of bytes used for objects in the Reduced Redundancy Storage (RRS) class. • GlacierInstantRetrievalSizeOverhead – The number of bytes used for objects smaller than 128 KB in size in the S3 Glacier Instant Retrieval storage class. • GlacierInstantRetrievalStorage – The number of bytes used for objects in the S3 Glacier Instant Retrieval storage class. • GlacierStorage – The number of bytes used for objects in the S3 Glacier Flexible Retrieval storage class. • GlacierStagingStorage – The number of bytes used for parts of Multipart objects before the CompleteMultipartUpload request is completed on objects in the S3 Glacier Flexible Retrieval storage class. • GlacierObjectOverhead – For each archived object, S3 Glacier adds 32 KB of storage for index and related metadata. This extra data is necessary to identify and restore your object. You are charged S3 Glacier Flexible Retrieval rates for this additional storage. • GlacierS3ObjectOverhead – For each object archived to S3 Glacier Flexible Retrieval, Amazon S3 uses 8 KB of storage for the name of the object and other metadata. You are charged STANDARD rates for this additional storage. • DeepArchiveStorage – The number of bytes used for objects in the S3 Glacier Deep Archive storage class. • DeepArchiveObjectOverhead – For each archived object, S3 Glacier adds 32 KB of storage for index and related metadata. This extra data is necessary to identify and restore your object. You are charged S3 Glacier Deep Archive rates for this additional storage. • DeepArchiveS3ObjectOverhead – For each object archived to S3 Glacier Deep Archive, Amazon S3 uses 8 KB of storage for the name of the object and other metadata. You are charged STANDARD rates for this additional storage. • DeepArchiveStagingStorage – The number of bytes used for parts of Multipart objects before the CompleteMultipartUpload request is completed on objects in the S3 Glacier Deep Archive storage class.
FilterId	This dimension filters metrics configurations that you specify for request metrics on a bucket, for example, a prefix or a tag. You specify a filter ID when you create a metrics configuration. For more information, see Creating a metrics configuration .

Amazon S3 Storage Lens dimensions in CloudWatch

For a list of dimensions used to filter S3 Storage Lens metrics in CloudWatch, see [Dimensions \(p. 1070\)](#).

Accessing CloudWatch metrics

You can use the following procedures to view the storage metrics for Amazon S3. To get the Amazon S3 metrics involved, you must set a start and end timestamp. For metrics for any given 24-hour period, set the time period to 86400 seconds, the number of seconds in a day. Also, remember to set the **BucketName** and **StorageType** dimensions.

Using the AWS CLI

For example, if you use the AWS CLI to get the average of a specific bucket's size in bytes, you could use the following command.

```
aws cloudwatch get-metric-statistics --metric-name BucketSizeBytes --namespace AWS/S3
--start-time 2016-10-19T00:00:00Z --end-time 2016-10-20T00:00:00Z --statistics Average
--unit Bytes --region us-west-2 --dimensions Name=BucketName,Value=ExampleBucket
Name=StorageType,Value=StandardStorage --period 86400 --output json
```

This example produces the following output.

```
{
    "Datapoints": [
        {
            "Timestamp": "2016-10-19T00:00:00Z",
            "Average": 1025328.0,
            "Unit": "Bytes"
        }
    ],
    "Label": "BucketSizeBytes"
}
```

Using the S3 console

To view metrics using the CloudWatch console

1. Open the CloudWatch console at <https://console.aws.amazon.com/cloudwatch/>.
2. In the navigation pane, choose **Metrics**.
3. Choose the **S3** namespace.
4. (Optional) To view a metric, enter the metric name in the search box.
5. (Optional) To filter by the **StorageType** dimension, enter the name of the storage class in the search box.

To view a list of valid metrics stored for your AWS account using the AWS CLI

- At a command prompt, use the following command.

```
aws cloudwatch list-metrics --namespace "AWS/S3"
```

CloudWatch metrics configurations

With Amazon CloudWatch request metrics for Amazon S3, you can receive 1-minute CloudWatch metrics, set CloudWatch alarms, and access CloudWatch dashboards to view near-real-time operations

and performance of your Amazon S3 storage. For applications that depend on cloud storage, these metrics let you quickly identify and act on operational issues. When enabled, these 1-minute metrics are available at the Amazon S3 bucket-level, by default.

If you want to get the CloudWatch request metrics for the objects in a bucket, you must create a metrics configuration for the bucket. For more information, see [Creating a CloudWatch metrics configuration for all the objects in your bucket \(p. 1013\)](#).

You can also use a shared prefix, object tags, or an access point to define a filter for the metrics collected. This method of defining a filter allows you to align metrics filters to specific business applications, workflows, or internal organizations. For more information, see [Creating a metrics configuration that filters by prefix, object tag, or access point \(p. 1014\)](#). For more information about the CloudWatch metrics that are available and the differences between storage and request metrics, see [Monitoring metrics with Amazon CloudWatch \(p. 1002\)](#).

Keep the following in mind when using metrics configurations:

- You can have a maximum of 1,000 metrics configurations per bucket.
- You can choose which objects in a bucket to include in metrics configurations by using filters. You can filter on a shared prefix, object tag, or access point to align metrics filters to specific business applications, workflows, or internal organizations. To request metrics for the entire bucket, create a metrics configuration without a filter.
- Metrics configurations are necessary only to enable request metrics. Bucket-level daily storage metrics are always turned on, and are provided at no additional cost. Currently, it's not possible to get daily storage metrics for a filtered subset of objects.
- Each metrics configuration enables the full set of [available request metrics \(p. 1005\)](#). Operation-specific metrics (such as `PostRequests`) are reported only if there are requests of that type for your bucket or your filter.
- Request metrics are reported for object-level operations. They are also reported for operations that list bucket contents, like `GET Bucket (List Objects)`, `GET Bucket Object Versions`, and `List Multipart Uploads`, but they are not reported for other operations on buckets.
- Request metrics support filtering by prefix, object tags, or access point, but storage metrics do not.

Best-effort CloudWatch metrics delivery

CloudWatch metrics are delivered on a best-effort basis. Most requests for an Amazon S3 object that have request metrics result in a data point being sent to CloudWatch.

The completeness and timeliness of metrics are not guaranteed. The data point for a particular request might be returned with a timestamp that is later than when the request was actually processed. The data point for a minute might be delayed before being available through CloudWatch, or it might not be delivered at all. CloudWatch request metrics give you an idea of the nature of traffic against your bucket in near-real time. It is not meant to be a complete accounting of all requests.

It follows from the best-effort nature of this feature that the reports available at the [Billing & Cost Management Dashboard](#) might include one or more access requests that do not appear in the bucket metrics.

For more information about working with CloudWatch metrics in Amazon S3, see the following topics.

Topics

- [Creating a CloudWatch metrics configuration for all the objects in your bucket \(p. 1013\)](#)
- [Creating a metrics configuration that filters by prefix, object tag, or access point \(p. 1014\)](#)
- [Deleting a metrics filter \(p. 1016\)](#)

Creating a CloudWatch metrics configuration for all the objects in your bucket

When you configure request metrics, you can create a CloudWatch metrics configuration for all the objects in your bucket, or you can filter by prefix, object tag, or access point. The procedures in this topic show you how to create a configuration for all the objects in your bucket. To create a configuration that filters by object tag, prefix, or access point, see [Creating a metrics configuration that filters by prefix, object tag, or access point \(p. 1014\)](#).

There are three types of Amazon CloudWatch metrics for Amazon S3: storage metrics, request metrics, and replication metrics. Storage metrics are reported once per day and are provided to all customers at no additional cost. Request metrics are available at one-minute intervals after some latency for processing. Request metrics are billed at the standard CloudWatch rate. You must opt in to request metrics by configuring them in the console or using the Amazon S3 API.

For more information about CloudWatch metrics for Amazon S3, see [Monitoring metrics with Amazon CloudWatch \(p. 1002\)](#).

You can add metrics configurations to a bucket using the Amazon S3 console, the AWS Command Line Interface (AWS CLI), or the Amazon S3 REST API.

Using the S3 console

1. Sign in to the AWS Management Console and open the Amazon S3 console at <https://console.aws.amazon.com/s3/>.
2. In the **Buckets** list, choose the name of the bucket that contains the objects you want request metrics for.
3. Choose the **Metrics** tab.
4. Under **Bucket metrics**, choose **View additional charts**.
5. Choose the **Request metrics** tab.
6. Choose **Create filter**.
7. In the **Filter name** box, enter your filter name.

Names can only contain letters, numbers, periods, dashes, and underscores. We recommend using the name `EntireBucket` for a filter that applies to all objects.

8. Under **Filter scope**, choose **This filter applies to all objects in the bucket**.

You can also define a filter so that the metrics are only collected and reported on a subset of objects in the bucket. For more information, see [Creating a metrics configuration that filters by prefix, object tag, or access point \(p. 1014\)](#).

9. Choose **Save changes**.
10. On the **Request metrics** tab, under **Filters**, choose the filter that you just created.

After about 15 minutes, CloudWatch begins tracking these request metrics. You can see them on the **Request metrics** tab. You can see graphs for the metrics on the Amazon S3 or CloudWatch console. Request metrics are billed at the standard CloudWatch rate. For more information, see [Amazon CloudWatch pricing](#).

Using the REST API

You can also add metrics configurations programmatically with the Amazon S3 REST API. For more information about adding and working with metrics configurations, see the following topics in the [Amazon Simple Storage Service API Reference](#):

- [PUT Bucket Metric Configuration](#)
- [GET Bucket Metric Configuration](#)
- [List Bucket Metric Configuration](#)
- [DELETE Bucket Metric Configuration](#)

Using the AWS CLI

1. Install and set up the AWS CLI. For instructions, see [Installing, updating, and uninstalling the AWS CLI in the AWS Command Line Interface User Guide](#).
2. Open a terminal.
3. Run the following command to add a metrics configuration.

```
aws s3api put-bucket-metrics-configuration --endpoint https://s3.us-west-2.amazonaws.com --bucket bucket-name --id metrics-config-id --metrics-configuration '{"Id":"'metrics-config-id'"}
```

Creating a metrics configuration that filters by prefix, object tag, or access point

There are three types of Amazon CloudWatch metrics for Amazon S3: storage metrics, request metrics, and replication metrics. Storage metrics are reported once per day and are provided to all customers at no additional cost. Request metrics are available at one-minute intervals after some latency for processing. Request metrics are billed at the standard CloudWatch rate. You must opt in to request metrics by configuring them in the console or using the Amazon S3 API.

For more information about CloudWatch metrics for Amazon S3, see [Monitoring metrics with Amazon CloudWatch \(p. 1002\)](#).

When you configure CloudWatch metrics, you can create a filter for all the objects in your bucket, or you can filter the configuration into groups of related objects within a single bucket. You can filter objects in a bucket for inclusion in a metrics configuration based on one or more of the following filter types:

- **Object key name prefix** – Although the Amazon S3 data model is a flat structure, you can infer a hierarchy by using a prefix. The Amazon S3 console supports these prefixes with the concept of folders. If you filter by prefix, objects that have the same prefix are included in the metrics configuration. For more information about prefixes, see [Organizing objects using prefixes \(p. 243\)](#).
- **Tag** – You can add tags, which are key-value name pairs, to objects. Tags help you find and organize objects easily. You can also use tags as filters for metrics configurations. For more information about object tags, see [Categorizing your storage using tags \(p. 825\)](#).
- **Access point** – S3 Access Points are named network endpoints that are attached to buckets and simplify managing data access at scale for shared datasets in S3. When you create an access point filter, Amazon S3 includes requests to the access point that you specify in the metrics configuration. For more information, see [Monitoring and logging access points \(p. 311\)](#).

Note

When you create a metrics configuration that filters by access point, you must use the access point Amazon Resource Name (ARN), not the access point alias. Make sure that you use the ARN for the access point itself, not the ARN for a specific object. For more information about access point ARNs, see [Using access points \(p. 310\)](#).

If you specify a filter, only requests that operate on single objects can match the filter and be included in the reported metrics. Requests like [Delete Multiple Objects](#) and [List](#) requests don't return any metrics for configurations with filters.

To request more complex filtering, choose two or more elements. Only objects that have all of those elements are included in the metrics configuration. If you don't set filters, all of the objects in the bucket are included in the metrics configuration.

Using the S3 console

1. Sign in to the AWS Management Console and open the Amazon S3 console at <https://console.aws.amazon.com/s3/>.
2. In the **Buckets** list, choose the name of the bucket that contains the objects that you want request metrics for.
3. Choose the **Metrics** tab.
4. Under **Bucket metrics**, choose **View additional charts**.
5. Choose the **Request metrics** tab.
6. Choose **Create filter**.
7. In the **Filter name** box, enter your filter name.

Names can contain only letters, numbers, periods, dashes, and underscores.
8. Under **Filter scope**, choose **Limit the scope of this filter using a prefix, object tags, and an S3 Access Point, or a combination of all three**.
9. Under **Filter type**, choose at least one filter type: **Prefix**, **Object tags**, or **Access Point**.
10. To define a prefix filter and limit the scope of the filter to a single path, in the **Prefix** box, enter a prefix.
11. To define an object tags filter, under **Object tags**, choose **Add tag**, and then enter a tag **Key** and **Value**.
12. To define an access point filter, in the **S3 Access Point** field, enter the access point ARN, or choose **Browse S3** to navigate to the access point.

Important

You cannot enter an access point alias. You must enter the ARN for the access point itself, not the ARN for a specific object.

13. Choose **Save changes**.

Amazon S3 creates a filter that uses the prefix, tags, or access point that you specified.

14. On the **Request metrics** tab, under **Filters**, choose the filter that you just created.

You have now created a filter that limits the request metrics scope by prefix, object tags, or access point. About 15 minutes after CloudWatch begins tracking these request metrics, you can see charts for the metrics on both the Amazon S3 and CloudWatch consoles. Request metrics are billed at the standard CloudWatch rate. For more information, see [Amazon CloudWatch pricing](#).

You can also configure request metrics at the bucket level. For information, see [Creating a CloudWatch metrics configuration for all the objects in your bucket \(p. 1013\)](#).

Using the REST API

You can also add metrics configurations programmatically with the Amazon S3 REST API. For more information about adding and working with metrics configurations, see the following topics in the *Amazon Simple Storage Service API Reference*:

- [PUT Bucket Metric Configuration](#)
- [GET Bucket Metric Configuration](#)
- [List Bucket Metric Configuration](#)
- [DELETE Bucket Metric Configuration](#)

Using the AWS CLI

1. Install and set up the AWS CLI. For instructions, see [Installing, updating, and uninstalling the AWS CLI](#) in the *AWS Command Line Interface User Guide*.
2. Open a terminal.
3. To add a metrics configuration, run one of the following commands:

Example : To filter by prefix

```
aws s3api put-bucket-metrics-configuration --bucket DOC-EXAMPLE-BUCKET1 --  
id metrics-config-id --metrics-configuration '{"Id":"'metrics-config-id'", "Filter":  
{"Prefix":"prefix1"}' '
```

Example : To filter by tags

```
aws s3api put-bucket-metrics-configuration --bucket DOC-EXAMPLE-BUCKET1 --id metrics-  
config-id --metrics-configuration '{"Id":"'metrics-config-id'", "Filter": {"Tag": {"Key":  
"string", "Value": "string"}}}' '
```

Example : To filter by access point

```
aws s3api put-bucket-metrics-configuration --bucket DOC-EXAMPLE-BUCKET1 --  
id metrics-config-id --metrics-configuration '{"Id":"'metrics-config-id'", "Filter":  
{"AccessPointArn": "arn:aws:s3:Region:account-id:accesspoint/access-point-name"}' '
```

Example : To filter by prefix, tags, and access point

```
aws s3api put-bucket-metrics-configuration --endpoint https://s3.Region.amazonaws.com  
--bucket DOC-EXAMPLE-BUCKET1 --id metrics-config-id --metrics-configuration '  
{  
    "Id": "metrics-config-id",  
    "Filter": {  
        "And": {  
            "Prefix": "string",  
            "Tags": [  
                {  
                    "Key": "string",  
                    "Value": "string"  
                }  
            ],  
            "AccessPointArn": "arn:aws:s3:Region:account-id:accesspoint/access-point-  
name"  
        }  
    }  
}'
```

Deleting a metrics filter

You can delete an Amazon CloudWatch request metrics filter if you no longer need it. When you delete a filter, you are no longer charged for request metrics that use that *specific filter*. However, you will continue to be charged for any other filter configurations that exist.

When you delete a filter, you can no longer use the filter for request metrics. Deleting a filter cannot be undone.

For information about creating a request metrics filter, see the following topics:

- [Creating a CloudWatch metrics configuration for all the objects in your bucket \(p. 1013\)](#)
- [Creating a metrics configuration that filters by prefix, object tag, or access point \(p. 1014\)](#)

Using the S3 console

1. Sign in to the AWS Management Console and open the Amazon S3 console at <https://console.aws.amazon.com/s3/>.
2. In the **Buckets** list, choose your bucket name.
3. Choose the **Metrics** tab.
4. Under **Bucket metrics**, choose **View additional charts**.
5. Choose the **Request metrics** tab.
6. Choose **Manage filters**.
7. Choose your filter.
Important
Deleting a filter cannot be undone.
8. Choose **Delete**.
Amazon S3 deletes your filter.

Using the REST API

You can also add metrics configurations programmatically with the Amazon S3 REST API. For more information about adding and working with metrics configurations, see the following topics in the *Amazon Simple Storage Service API Reference*:

- [PUT Bucket Metric Configuration](#)
- [GET Bucket Metric Configuration](#)
- [List Bucket Metric Configuration](#)
- [DELETE Bucket Metric Configuration](#)

Amazon S3 Event Notifications

You can use the Amazon S3 Event Notifications feature to receive notifications when certain events happen in your S3 bucket. To enable notifications, add a notification configuration that identifies the events that you want Amazon S3 to publish. Make sure that it also identifies the destinations where you want Amazon S3 to send the notifications. You store this configuration in the *notification* subresource that's associated with a bucket. For more information, see [Bucket configuration options \(p. 116\)](#). Amazon S3 provides an API for you to manage this subresource.

Important

Amazon S3 event notifications are designed to be delivered at least once. Typically, event notifications are delivered in seconds but can sometimes take a minute or longer.

Overview of Amazon S3 Event Notifications

Currently, Amazon S3 can publish notifications for the following events:

- New object created events
- Object removal events
- Restore object events
- Reduced Redundancy Storage (RRS) object lost events

- Replication events
- S3 Lifecycle expiration events
- S3 Lifecycle transition events
- S3 Intelligent-Tiering automatic archival events
- Object tagging events
- Object ACL PUT events

For full descriptions of all the supported event types, see [Supported event types for SQS, SNS, and Lambda \(p. 1020\)](#).

Amazon S3 can send event notification messages to the following destinations. You specify the Amazon Resource Name (ARN) value of these destinations in the notification configuration.

- Amazon Simple Notification Service (Amazon SNS) topics
- Amazon Simple Queue Service (Amazon SQS) queues
- AWS Lambda function

For more information, see [Supported event destinations \(p. 1018\)](#).

Warning

If your notification writes to the same bucket that triggers the notification, it could cause an execution loop. For example, if the bucket triggers a Lambda function each time an object is uploaded, and the function uploads an object to the bucket, then the function indirectly triggers itself. To avoid this, use two buckets, or configure the trigger to only apply to a prefix used for incoming objects.

For more information and an example of using Amazon S3 notifications with AWS Lambda, see [Using AWS Lambda with Amazon S3](#) in the *AWS Lambda Developer Guide*.

For more information about the number of event notification configurations that you can create per bucket, see [Amazon S3 service quotas](#) in *AWS General Reference*.

For more information about event notifications, see the following sections.

Topics

- [Event notification types and destinations \(p. 1018\)](#)
- [Using Amazon SQS, Amazon SNS, and Lambda \(p. 1022\)](#)
- [Using EventBridge \(p. 1041\)](#)

Event notification types and destinations

Amazon S3 supports several event notification types and destinations where the notifications can be published. You can specify the event type and destination when configuring your event notifications.

Topics

- [Supported event destinations \(p. 1018\)](#)
- [Supported event types for SQS, SNS, and Lambda \(p. 1020\)](#)
- [Supported event types for Amazon EventBridge \(p. 1022\)](#)

Supported event destinations

Amazon S3 can send event notification messages to the following destinations.

- Amazon Simple Notification Service (Amazon SNS) topics
- Amazon Simple Queue Service (Amazon SQS) queues
- AWS Lambda
- Amazon EventBridge

Note

You must grant Amazon S3 permissions to post messages to an Amazon SNS topic or an Amazon SQS queue. You must also grant Amazon S3 permission to invoke an AWS Lambda function on your behalf. For instructions on how to grant these permissions, see [Granting permissions to publish event notification messages to a destination \(p. 1023\)](#).

Amazon SNS topic

Amazon SNS is a flexible, fully managed push messaging service. You can use this service to push messages to mobile devices or distributed services. With SNS, you can publish a message once, and deliver it one or more times. Currently, Standard SNS is only allowed as an S3 event notification destination, whereas SNS FIFO is not allowed.

Amazon SNS both coordinates and manages sending and delivering messages to subscribing endpoints or clients. You can use the Amazon SNS console to create an Amazon SNS topic that your notifications can be sent to.

The topic must be in the same AWS Region as your Amazon S3 bucket. For instructions on how to create an Amazon SNS topic, see [Getting started with Amazon SNS](#) in the *Amazon Simple Notification Service Developer Guide* and the [Amazon SNS FAQ](#).

Before you can use the Amazon SNS topic that you created as an event notification destination, you need the following:

- The Amazon Resource Name (ARN) for the Amazon SNS topic
- A valid Amazon SNS topic subscription. With it, topic subscribers are notified when a message is published to your Amazon SNS topic.

Amazon SQS queue

Amazon SQS offers reliable and scalable hosted queues for storing messages as they travel between computers. You can use Amazon SQS to transmit any volume of data without requiring other services to be always available. You can use the Amazon SQS console to create an Amazon SQS queue that your notifications can be sent to.

The Amazon SQS queue must be in the same AWS Region as your Amazon S3 bucket. For instructions on how to create an Amazon SQS queue, see [What is Amazon Simple Queue Service](#) and [Getting started with Amazon SQS](#) in the *Amazon Simple Queue Service Developer Guide*.

Before you can use the Amazon SQS queue as an event notification destination, you need the following:

- The Amazon Resource Name (ARN) for the Amazon SQS queue

Lambda function

You can use AWS Lambda to extend other AWS services with custom logic, or create your own backend that operates at AWS scale, performance, and security. With Lambda, you can create discrete, event-driven applications that run only when needed. You can also use it to scale these applications automatically from a few requests a day to thousands a second.

Lambda can run custom code in response to Amazon S3 bucket events. You upload your custom code to Lambda and create what's called a Lambda function. When Amazon S3 detects an event of a specific

type, it can publish the event to AWS Lambda and invoke your function in Lambda. In response, Lambda runs your function. One event type it might detect, for example, is an object created event.

You can use the AWS Lambda console to create a Lambda function that uses the AWS infrastructure to run the code on your behalf. The Lambda function must be in the same Region as your S3 bucket. You must also have the name or the ARN of a Lambda function to set up the Lambda function as an event notification destination.

Warning

If your notification writes to the same bucket that triggers the notification, it could cause an execution loop. For example, if the bucket triggers a Lambda function each time an object is uploaded, and the function uploads an object to the bucket, then the function indirectly triggers itself. To avoid this, use two buckets, or configure the trigger to only apply to a prefix used for incoming objects.

For more information and an example of using Amazon S3 notifications with AWS Lambda, see [Using AWS Lambda with Amazon S3](#) in the *AWS Lambda Developer Guide*.

Amazon EventBridge

Amazon EventBridge is a serverless event bus, which receives events from AWS services. You can set up rules to match events and deliver them to targets, such as an AWS service or an HTTP endpoint. For more information, see [What is EventBridge](#) in the *Amazon EventBridge User Guide*.

Unlike other destinations, you can either enable or disable events to be delivered to EventBridge for a bucket. If you enable delivery, all events are sent to EventBridge. Moreover, you can use EventBridge rules to route events to additional targets.

Supported event types for SQS, SNS, and Lambda

Amazon S3 can publish events of the following types. You specify these event types in the notification configuration.

Event types	Description
<code>s3:TestEvent</code>	When a notification is enabled, Amazon S3 publishes a test notification. This is to ensure that the topic exists and that the bucket owner has permission to publish the specified topic. If enabling the notification fails, you don't receive a test notification.
<code>s3:ObjectCreated:*</code> <code>s3:ObjectCreated:Put</code> <code>s3:ObjectCreated:Post</code> <code>s3:ObjectCreated:Copy</code> <code>s3:ObjectCreated:CompleteMultipartUpload</code>	Amazon S3 API operations such as PUT, POST, and COPY can create an object. With these event types, you can enable notifications when an object is created using a specific API operation. Alternatively, you can use the <code>s3:ObjectCreated:*</code> event type to request notification regardless of the API that was used to create an object. You don't receive event notifications from failed operations. <code>s3:ObjectCreated:CompleteMultipartUpload</code> includes objects that are created using UploadPartCopy for Copy operations.
<code>s3:ObjectRemoved:*</code> <code>s3:ObjectRemoved:Delete</code> <code>s3:ObjectRemoved:DeleteMarkerCreated</code>	By using the <code>ObjectRemoved</code> event types, you can enable notification when an object or a batch of objects is removed from a bucket.

Event types	Description
	<p>You can request notification when an object is deleted or a versioned object is permanently deleted by using the <code>s3:ObjectRemoved:Delete</code> event type. Alternatively, you can request notification when a delete marker is created for a versioned object using <code>s3:ObjectRemoved:DeleteMarkerCreated</code>. For instructions on how to delete versioned objects, see Deleting object versions from a versioning-enabled bucket (p. 659). You can also use a wildcard <code>s3:ObjectRemoved:*</code> to request notification anytime an object is deleted.</p> <p>These event notifications don't alert you for automatic deletes from Lifecycle policies or from failed operations.</p>
<code>s3:ObjectRestore:*</code> <code>s3:ObjectRestore:Post</code> <code>s3:ObjectRestore:Completed</code> <code>s3:ObjectRestore:Delete</code>	<p>By using <i>ObjectRestore</i> event types, you can receive notifications for event initiation and completion when restoring objects from the S3 Glacier Flexible Retrieval storage class and S3 Glacier Deep Archive storage class. You can also receive notifications for when the restored copy of an object expires.</p> <p>The <code>s3:ObjectRestore:Post</code> event type notifies you of object restoration initiation. The <code>s3:ObjectRestore:Completed</code> event type notifies you of restoration completion. The <code>s3:ObjectRestore:Delete</code> event type notifies you when the temporary copy of a restored object expires.</p>
<code>s3:ReducedRedundancyLostObject</code>	You receive this notification event when Amazon S3 detects that an object of the RRS storage class is lost.
<code>s3:Replication:*</code> <code>s3:Replication:OperationFailedReplication</code> <code>s3:Replication:OperationMissedThreshold</code> <code>s3:Replication:OperationReplicatedAfterThreshold</code> <code>s3:Replication:OperationNotTracked</code>	<p>By using the <i>Replication</i> event types, you can receive notifications for replication configurations that have S3 replication metrics or S3 Replication Time Control (S3 RTC) enabled. You can monitor the minute-by-minute progress of replication events by tracking bytes pending, operations pending, and replication latency. For information about replication metrics, see Monitoring progress with replication metrics and Amazon S3 event notifications (p. 805)</p> <p>The <code>s3:Replication:OperationFailedReplication</code> event type notifies you when an object that was eligible for replication failed to replicate. The <code>s3:Replication:OperationMissedThreshold</code> event type notifies you when an object that was eligible for replication exceeds the 15-minute threshold for replication.</p> <p>The <code>s3:Replication:OperationReplicatedAfterThreshold</code> event type notifies you when an object that was eligible for replication that uses S3 Replication Time Control replicates after the 15-minute threshold. The <code>s3:Replication:OperationNotTracked</code> event type notifies you when an object that was eligible for replication that uses S3 Replication Time Control but is no longer tracked by replication metrics.</p>

Event types	Description
<code>s3:LifecycleExpiration:*</code> <code>s3:LifecycleExpiration:Delete</code>	By using the <i>LifecycleExpiration</i> event types, you can receive a notification when Amazon S3 deletes an object based on your S3 Lifecycle configuration.
<code>s3:LifecycleExpiration:DeleteMarkerCreated</code>	The <code>s3:LifecycleExpiration:Delete</code> event type notifies you when an object in an unversioned bucket is deleted. It also notifies you when an object version is permanently deleted by an S3 Lifecycle configuration. The <code>s3:LifecycleExpiration:DeleteMarkerCreated</code> event type notifies you when S3 Lifecycle creates a delete marker when a current version of an object in versioned bucket is deleted.
<code>s3:LifecycleTransition</code>	You receive this notification event when an object is transitioned to another Amazon S3 storage class by an S3 Lifecycle configuration.
<code>s3:IntelligentTiering</code>	You receive this notification event when an object within the S3 Intelligent-Tiering storage class moved to the Archive Access tier or Deep Archive Access tier.
<code>s3:ObjectTagging:*</code> <code>s3:ObjectTagging:Put</code> <code>s3:ObjectTagging:Delete</code>	By using the <i>ObjectTagging</i> event types, you can enable notification when an object tag is added or deleted from an object. The <code>s3:ObjectTagging:Put</code> event type notifies you when a tag is PUT on an object or an existing tag is updated. The <code>s3:ObjectTagging:Delete</code> event type notifies you when a tag is removed from an object.
<code>s3:ObjectAcl:Put</code>	You receive this notification event when an ACL is PUT on an object or when an existing ACL is changed. An event is not generated when a request results in no change to an object's ACL.

Supported event types for Amazon EventBridge

For a list of event types Amazon S3 will send to Amazon EventBridge, see [Using EventBridge \(p. 1041\)](#)

Using Amazon SQS, Amazon SNS, and Lambda

Enabling notifications is a bucket-level operation. You store notification configuration information in the *notification* subresource that's associated with a bucket. After you create or change the bucket notification configuration, it usually takes about five minutes for the changes to take effect. When the notification is first enabled, an `s3:TestEvent` occurs. You can use any of the following methods to manage notification configuration:

- **Using the Amazon S3 console** — You can use the console UI to set a notification configuration on a bucket without having to write any code. For more information, see [Enabling and configuring event notifications using the Amazon S3 console \(p. 1025\)](#).
- **Programmatically using the AWS SDKs** — Internally, both the console and the SDKs call the Amazon S3 REST API to manage *notification* subresources that are associated with the bucket. For examples of notification configurations that use AWS SDK, see [Walkthrough: Configuring a bucket for notifications \(SNS topic or SQS queue\) \(p. 1028\)](#).

Note

You can also make the Amazon S3 REST API calls directly from your code. However, this can be cumbersome because to do so you must write code to authenticate your requests.

Regardless of the method that you use, Amazon S3 stores the notification configuration as XML in the *notification* subresource that's associated with a bucket. For information about bucket subresources, see [Bucket configuration options \(p. 116\)](#).

Topics

- [Granting permissions to publish event notification messages to a destination \(p. 1023\)](#)
- [Enabling and configuring event notifications using the Amazon S3 console \(p. 1025\)](#)
- [Configuring event notifications programmatically \(p. 1027\)](#)
- [Walkthrough: Configuring a bucket for notifications \(SNS topic or SQS queue\) \(p. 1028\)](#)
- [Configuring event notifications using object key name filtering \(p. 1033\)](#)
- [Event message structure \(p. 1038\)](#)

Granting permissions to publish event notification messages to a destination

You must grant the Amazon S3 principal the necessary permissions to call the relevant API to publish messages to an SNS topic, an SQS queue, or a Lambda function. This is so that Amazon S3 can publish event notification messages to a destination.

Topics

- [Granting permissions to invoke an AWS Lambda function \(p. 1023\)](#)
- [Granting permissions to publish messages to an SNS topic or an SQS queue \(p. 1023\)](#)

Granting permissions to invoke an AWS Lambda function

Amazon S3 publishes event messages to AWS Lambda by invoking a Lambda function and providing the event message as an argument.

When you use the Amazon S3 console to configure event notifications on an Amazon S3 bucket for a Lambda function, the console sets up the necessary permissions on the Lambda function. This is so that Amazon S3 has permissions to invoke the function from the bucket. For more information, see [Enabling and configuring event notifications using the Amazon S3 console \(p. 1025\)](#).

You can also grant Amazon S3 permissions from AWS Lambda to invoke your Lambda function. For more information, see [Tutorial: Using AWS Lambda with Amazon S3](#) in the *AWS Lambda Developer Guide*.

Granting permissions to publish messages to an SNS topic or an SQS queue

To grant Amazon S3 permissions to publish messages to the SNS topic or SQS queue, attach an AWS Identity and Access Management (IAM) policy to the destination SNS topic or SQS queue.

For an example of how to attach a policy to an SNS topic or an SQS queue, see [Walkthrough: Configuring a bucket for notifications \(SNS topic or SQS queue\) \(p. 1028\)](#). For more information about permissions, see the following topics:

- [Example cases for Amazon SNS access control](#) in the *Amazon Simple Notification Service Developer Guide*

- [Identity and access management in Amazon SQS](#) in the *Amazon Simple Queue Service Developer Guide*

IAM policy for a destination SNS topic

The following is an example of an AWS Identity and Access Management (IAM) policy that you attach to the destination SNS topic. For instructions on how to use this policy to set up a destination Amazon SNS topic for event notifications, see [Walkthrough: Configuring a bucket for notifications \(SNS topic or SQS queue\) \(p. 1028\)](#).

```
{  
    "Version": "2012-10-17",  
    "Id": "example-ID",  
    "Statement": [  
        {  
            "Sid": "Example SNS topic policy",  
            "Effect": "Allow",  
            "Principal": {  
                "Service": "s3.amazonaws.com"  
            },  
            "Action": [  
                "SNS:Publish"  
            ],  
            "Resource": "SNS-topic-ARN",  
            "Condition": {  
                "ArnLike": {  
                    "aws:SourceArn": "arn:aws:s3:::bucket-name"  
                },  
                "StringEquals": {  
                    "aws:SourceAccount": "bucket-owner-account-id"  
                }  
            }  
        }  
    ]  
}
```

IAM policy for a destination SQS queue

The following is an example of an IAM policy that you attach to the destination SQS queue. For instructions on how to use this policy to set up a destination Amazon SQS queue for event notifications, see [Walkthrough: Configuring a bucket for notifications \(SNS topic or SQS queue\) \(p. 1028\)](#).

To use this policy, you must update the Amazon SQS queue ARN, bucket name, and bucket owner's AWS account ID.

```
{  
    "Version": "2012-10-17",  
    "Id": "example-ID",  
    "Statement": [  
        {  
            "Sid": "example-statement-ID",  
            "Effect": "Allow",  
            "Principal": {  
                "Service": "s3.amazonaws.com"  
            },  
            "Action": [  
                "SQS:SendMessage"  
            ],  
            "Resource": "arn:aws:sqs:Region:account-id:queue-name",  
            "Condition": {  
                "ArnLike": {  
                    "aws:SourceArn": "arn:aws:s3:::awsexamplebucket1"  
                }  
            }  
        }  
    ]  
}
```

```
        },
        "StringEquals": {
            "aws:SourceAccount": "bucket-owner-account-id"
        }
    }
}
```

For both the Amazon SNS and Amazon SQS IAM policies, you can specify the `StringLike` condition in the policy instead of the `ArnLike` condition.

```
"Condition": {
    "StringLike": { "aws:SourceArn": "arn:aws:s3:::*:bucket-name" }
}
```

AWS KMS key policy

If the SQS queue or SNS topics are encrypted with an AWS Key Management Service (AWS KMS) customer managed key, you must grant the Amazon S3 service principal permission to work with the encrypted topics or queue. To grant the Amazon S3 service principal permission, add the following statement to the key policy for the customer managed key.

```
{
    "Version": "2012-10-17",
    "Id": "example-ID",
    "Statement": [
        {
            "Sid": "example-statement-ID",
            "Effect": "Allow",
            "Principal": {
                "Service": "s3.amazonaws.com"
            },
            "Action": [
                "kms:GenerateDataKey",
                "kms:Decrypt"
            ],
            "Resource": "*"
        }
    ]
}
```

For more information about AWS KMS key policies, see [Using key policies in AWS KMS](#) in the *AWS Key Management Service Developer Guide*.

For more information about using server-side encryption with AWS KMS for Amazon SQS and Amazon SNS, see the following:

- [Key management](#) in the *Amazon Simple Notification Service Developer Guide*.
- [Key management](#) in the *Amazon Simple Queue Service Developer Guide*.
- [Encrypting messages published to Amazon SNS with AWS KMS](#) in the *AWS Compute Blog*.

Enabling and configuring event notifications using the Amazon S3 console

You can enable certain Amazon S3 bucket events to send a notification message to a destination whenever those events occur. This section explains how to use the Amazon S3 console to enable event

notifications. For information about how to use event notifications with the AWS SDKs and the Amazon S3 REST APIs, see [Configuring event notifications programmatically \(p. 1027\)](#).

Prerequisites: Before you can enable event notifications for your bucket, you must set up one of the destination types and then configure permissions. For more information, see [Supported event destinations \(p. 1018\)](#) and [Granting permissions to publish event notification messages to a destination \(p. 1023\)](#).

Topics

- [Enabling Amazon SNS, Amazon SQS, or Lambda notifications using the Amazon S3 console \(p. 1026\)](#)

Enabling Amazon SNS, Amazon SQS, or Lambda notifications using the Amazon S3 console

To enable and configure event notifications for an S3 bucket

1. Sign in to the AWS Management Console and open the Amazon S3 console at <https://console.aws.amazon.com/s3/>.
2. In the **Buckets** list, choose the name of the bucket that you want to enable events for.
3. Choose **Properties**.
4. Navigate to the **Event Notifications** section and choose **Create event notification**.
5. In the **General configuration** section, specify descriptive event name for your event notification. Optionally, you can also specify a prefix and a suffix to limit the notifications to objects with keys ending in the specified characters.
 - a. Enter a description for the **Event name**.
If you don't enter a name, a globally unique identifier (GUID) is generated and used for the name.
 - b. (Optional) To filter event notifications by prefix, enter a **Prefix**.
For example, you can set up a prefix filter so that you receive notifications only when files are added to a specific folder (for example, `images/`).
 - c. (Optional) To filter event notifications by suffix, enter a **Suffix**.
For more information, see [Configuring event notifications using object key name filtering \(p. 1033\)](#).
6. In the **Event types** section, select one or more event types that you want to receive notifications for.
For a list of the different event types, see [Supported event types for SQS, SNS, and Lambda \(p. 1020\)](#).
7. In the **Destination** section, choose the event notification destination.

Note

Before you can publish event notifications, you must grant the Amazon S3 principal the necessary permissions to call the relevant API. This is so that it can publish notifications to a Lambda function, SNS topic, or SQS queue.

- a. Select the destination type: **Lambda Function**, **SNS Topic**, or **SQS Queue**.
- b. After you choose your destination type, choose a function, topic, or queue from the list.
- c. Or, if you prefer to specify an Amazon Resource Name (ARN), select **Enter ARN** and enter the ARN.

For more information, see [Supported event destinations \(p. 1018\)](#).

8. Choose **Save changes**, and Amazon S3 sends a test message to the event notification destination.

Configuring event notifications programmatically

By default, notifications aren't enabled for any type of event. Therefore, the *notification* subresource initially stores an empty configuration.

```
<NotificationConfiguration xmlns="http://s3.amazonaws.com/doc/2006-03-01/">
</NotificationConfiguration>
```

To enable notifications for events of specific types, you replace the XML with the appropriate configuration that identifies the event types you want Amazon S3 to publish and the destination where you want the events published. For each destination, you add a corresponding XML configuration.

To publish event messages to an SQS queue

To set an SQS queue as the notification destination for one or more event types, add the *QueueConfiguration*.

```
<NotificationConfiguration>
<QueueConfiguration>
  <Id>optional-id-string</Id>
  <Queue>sqs-queue-arn</Queue>
  <Event>event-type</Event>
  <Event>event-type</Event>
  ...
</QueueConfiguration>
...
</NotificationConfiguration>
```

To publish event messages to an SNS topic

To set an SNS topic as the notification destination for specific event types, add the *TopicConfiguration*.

```
<NotificationConfiguration>
<TopicConfiguration>
  <Id>optional-id-string</Id>
  <Topic>sns-topic-arn</Topic>
  <Event>event-type</Event>
  <Event>event-type</Event>
  ...
</TopicConfiguration>
...
</NotificationConfiguration>
```

To invoke the AWS Lambda function and provide an event message as an argument

To set a Lambda function as the notification destination for specific event types, add the *CloudFunctionConfiguration*.

```
<NotificationConfiguration>
<CloudFunctionConfiguration>
  <Id>optional-id-string</Id>
  <CloudFunction>cloud-function-arn</CloudFunction>
  <Event>event-type</Event>
  <Event>event-type</Event>
  ...
</CloudFunctionConfiguration>
</NotificationConfiguration>
```

```
</CloudFunctionConfiguration>
...
</NotificationConfiguration>
```

To remove all notifications configured on a bucket

To remove all notifications configured on a bucket, save an empty `<NotificationConfiguration/>` element in the *notification* subresource.

When Amazon S3 detects an event of the specific type, it publishes a message with the event information. For more information, see [Event message structure \(p. 1038\)](#).

For more information about configuring event notifications, see the following topics:

- [Walkthrough: Configuring a bucket for notifications \(SNS topic or SQS queue\) \(p. 1028\)](#)
- [Configuring event notifications using object key name filtering \(p. 1033\)](#)

Walkthrough: Configuring a bucket for notifications (SNS topic or SQS queue)

You can receive Amazon S3 notifications using Amazon Simple Notification Service (Amazon SNS) or Amazon Simple Queue Service (Amazon SQS). In this walkthrough, you add a notification configuration to your bucket using an Amazon SNS topic and an Amazon SQS queue.

Topics

- [Walkthrough summary \(p. 1028\)](#)
- [Step 1: Create an Amazon SQS queue \(p. 1029\)](#)
- [Step 2: Create an Amazon SNS topic \(p. 1030\)](#)
- [Step 3: Add a notification configuration to your bucket \(p. 1031\)](#)
- [Step 4: Test the setup \(p. 1033\)](#)

Walkthrough summary

This walkthrough helps you do the following:

- Publish events of the `s3:ObjectCreated:*` type to an Amazon SQS queue.
- Publish events of the `s3:ReducedRedundancyLostObject` type to an Amazon SNS topic.

For information about notification configuration, see [Using Amazon SQS, Amazon SNS, and Lambda \(p. 1022\)](#).

You can do all these steps using the console, without writing any code. In addition, code examples using AWS SDKs for Java and .NET are also provided to help you add notification configurations programmatically.

The procedure includes the following steps:

1. Create an Amazon SQS queue.

Using the Amazon SQS console, create an SQS queue. You can access any messages Amazon S3 sends to the queue programmatically. But, for this walkthrough, you verify notification messages in the console.

You attach an access policy to the queue to grant Amazon S3 permission to post messages.

2. Create an Amazon SNS topic.

Using the Amazon SNS console, create an SNS topic and subscribe to the topic. That way, any events posted to it are delivered to you. You specify email as the communications protocol. After you create a topic, Amazon SNS sends an email. You use the link in the email to confirm the topic subscription.

You attach an access policy to the topic to grant Amazon S3 permission to post messages.

3. Add notification configuration to a bucket.

Step 1: Create an Amazon SQS queue

Follow the steps to create and subscribe to an Amazon Simple Queue Service (Amazon SQS) queue.

1. Using the Amazon SQS console, create a queue. For instructions, see [Getting Started with Amazon SQS](#) in the *Amazon Simple Queue Service Developer Guide*.
2. Replace the access policy that's attached to the queue with the following policy.
 - a. In the Amazon SQS console, in the **Queues** list, choose the queue name.
 - b. On the **Access policy** tab, choose **Edit**.
 - c. Replace the access policy that's attached to the queue. In it, provide your Amazon SQS ARN, source bucket name, and bucket owner account ID.

```
{  
    "Version": "2012-10-17",  
    "Id": "example-ID",  
    "Statement": [  
        {  
            "Sid": "example-statement-ID",  
            "Effect": "Allow",  
            "Principal": {  
                "Service": "s3.amazonaws.com"  
            },  
            "Action": [  
                "SQS:SendMessage"  
            ],  
            "Resource": "SQS-queue-ARN",  
            "Condition": {  
                "ArnLike": {  
                    "aws:SourceArn": "arn:aws:s3:::awsexamplebucket1"  
                },  
                "StringEquals": {  
                    "aws:SourceAccount": "bucket-owner-account-id"  
                }  
            }  
        }  
    ]  
}
```

- d. Choose **Save**.
3. (Optional) If the Amazon SQS queue or the Amazon SNS topic is server-side encryption enabled with AWS Key Management Service (AWS KMS), add the following policy to the associated symmetric encryption customer managed key.

You must add the policy to a customer managed key because you cannot modify the AWS managed key for Amazon SQS or Amazon SNS.

```
{  
    "Version": "2012-10-17",  
    "Id": "example-ID",
```

```
"Statement": [
    {
        "Sid": "example-statement-ID",
        "Effect": "Allow",
        "Principal": {
            "Service": "s3.amazonaws.com"
        },
        "Action": [
            "kms:GenerateDataKey",
            "kms:Decrypt"
        ],
        "Resource": "*"
    }
]
```

For more information about using SSE for Amazon SQS and Amazon SNS with AWS KMS, see the following:

- [Key management](#) in the *Amazon Simple Notification Service Developer Guide*.
 - [Key management](#) in the *Amazon Simple Queue Service Developer Guide*.
4. Note the queue ARN.

The SQS queue that you created is another resource in your AWS account. It has a unique Amazon Resource Name (ARN). You need this ARN in the next step. The ARN is of the following format:

```
arn:aws:sqs:aws-region:account-id:queue-name
```

Step 2: Create an Amazon SNS topic

Follow the steps to create and subscribe to an Amazon SNS topic.

1. Using Amazon SNS console, create a topic. For instructions, see [Creating an Amazon SNS topic](#) in the *Amazon Simple Notification Service Developer Guide*.
2. Subscribe to the topic. For this exercise, use email as the communications protocol. For instructions, see [Subscribing to an Amazon SNS topic](#) in the *Amazon Simple Notification Service Developer Guide*.

- You get an email requesting you to confirm your subscription to the topic. Confirm the subscription.
3. Replace the access policy attached to the topic with the following policy. In it, provide your SNS topic ARN, bucket name, and bucket owner's account ID.

```
{
    "Version": "2012-10-17",
    "Id": "example-ID",
    "Statement": [
        {
            "Sid": "Example SNS topic policy",
            "Effect": "Allow",
            "Principal": {
                "Service": "s3.amazonaws.com"
            },
            "Action": [
                "SNS:Publish"
            ],
            "Resource": "SNS-topic-ARN",
            "Condition": {
                "ArnLike": {
                    "aws:SourceArn": "arn:aws:s3:::bucket-name"
                },
            }
        }
]
```

```
        "StringEquals": {
            "aws:SourceAccount": "bucket-owner-account-id"
        }
    }
}
```

4. Note the topic ARN.

The SNS topic you created is another resource in your AWS account, and it has a unique ARN. You will need this ARN in the next step. The ARN will be of the following format:

```
arn:aws:sns:aws-region:account-id:topic-name
```

Step 3: Add a notification configuration to your bucket

You can enable bucket notifications either by using the Amazon S3 console or programmatically by using AWS SDKs. Choose any one of the options to configure notifications on your bucket. This section provides code examples using the AWS SDKs for Java and .NET.

Option A: Enable notifications on a bucket using the console

Using the Amazon S3 console, add a notification configuration requesting Amazon S3 to do the following:

- Publish events of the **All object create events** type to your Amazon SQS queue.
- Publish events of the **Object in RRS lost** type to your Amazon SNS topic.

After you save the notification configuration, Amazon S3 posts a test message, which you get via email.

For instructions, see [Enabling and configuring event notifications using the Amazon S3 console \(p. 1025\)](#).

Option B: Enable notifications on a bucket using the AWS SDKs

.NET

The following C# code example provides a complete code listing that adds a notification configuration to a bucket. You must update the code and provide your bucket name and SNS topic ARN. For instructions on how to create and test a working sample, see [Running the Amazon S3 .NET Code Examples \(p. 1193\)](#).

```
using Amazon;
using Amazon.S3;
using Amazon.S3.Model;
using System;
using System.Collections.Generic;
using System.Threading.Tasks;

namespace Amazon.DocSamples.S3
{
    class EnableNotificationsTest
    {
        private const string bucketName = "*** bucket name ***";
        private const string snsTopic = "*** SNS topic ARN ***";
        private const string sqsQueue = "*** SQS topic ARN ***";
        // Specify your bucket region (an example region is shown).
        private static readonly RegionEndpoint bucketRegion = RegionEndpoint.USWest2;
```

```

private static IAmazonS3 client;

public static void Main()
{
    client = new AmazonS3Client(bucketRegion);
    EnableNotificationAsync().Wait();
}

static async Task EnableNotificationAsync()
{
    try
    {
        PutBucketNotificationRequest request = new PutBucketNotificationRequest
        {
            BucketName = bucketName
        };

        TopicConfiguration c = new TopicConfiguration
        {
            Events = new List<EventType> { EventType.ObjectCreatedCopy },
            Topic = snsTopic
        };
        request.TopicConfigurations = new List<TopicConfiguration>();
        request.TopicConfigurations.Add(c);
        request.QueueConfigurations = new List<QueueConfiguration>();
        request.QueueConfigurations.Add(new QueueConfiguration()
        {
            Events = new List<EventType> { EventType.ObjectCreatedPut },
            Queue = sqsQueue
        });

        PutBucketNotificationResponse response = await
client.PutBucketNotificationAsync(request);
    }
    catch (AmazonS3Exception e)
    {
        Console.WriteLine("Error encountered on server. Message:'{0}' ", e.Message);
    }
    catch (Exception e)
    {
        Console.WriteLine("Unknown error encountered on server. Message:'{0}' ", e.Message);
    }
}
}

```

Java

The following example shows how to add a notification configuration to a bucket. For instructions on how to create and test a working sample, see [Testing the Amazon S3 Java Code Examples \(p. 1192\)](#).

```
import com.amazonaws.AmazonServiceException;
import com.amazonaws.SdkClientException;
import com.amazonaws.auth.profile.ProfileCredentialsProvider;
import com.amazonaws.regions.Regions;
import com.amazonaws.services.s3.AmazonS3;
import com.amazonaws.services.s3.AmazonS3ClientBuilder;
import com.amazonaws.services.s3.model.*;

import java.io.IOException;
```

```
import java.util.EnumSet;

public class EnableNotificationOnABucket {

    public static void main(String[] args) throws IOException {
        String bucketName = "*** Bucket name ***";
        Regions clientRegion = Regions.DEFAULT_REGION;
        String snsTopicARN = "*** SNS Topic ARN ***";
        String sqsQueueARN = "*** SQS Queue ARN ***";

        try {
            AmazonS3 s3Client = AmazonS3ClientBuilder.standard()
                .withCredentials(new ProfileCredentialsProvider())
                .withRegion(clientRegion)
                .build();
            BucketNotificationConfiguration notificationConfiguration = new
BucketNotificationConfiguration();

            // Add an SNS topic notification.
            notificationConfiguration.addConfiguration("snsTopicConfig",
                new TopicConfiguration(snsTopicARN,
                    EnumSet.of(S3Event.ObjectCreated)));

            // Add an SQS queue notification.
            notificationConfiguration.addConfiguration("sqsQueueConfig",
                new QueueConfiguration(sqsQueueARN,
                    EnumSet.of(S3Event.ObjectCreated)));

            // Create the notification configuration request and set the bucket
            notification configuration.
            SetBucketNotificationConfigurationRequest request = new
SetBucketNotificationConfigurationRequest(
                bucketName, notificationConfiguration);
            s3Client.setBucketNotificationConfiguration(request);
        } catch (AmazonServiceException e) {
            // The call was transmitted successfully, but Amazon S3 couldn't process
            // it, so it returned an error response.
            e.printStackTrace();
        } catch (SdkClientException e) {
            // Amazon S3 couldn't be contacted for a response, or the client
            // couldn't parse the response from Amazon S3.
            e.printStackTrace();
        }
    }
}
```

Step 4: Test the setup

Now, you can test the setup by uploading an object to your bucket and verifying the event notification in the Amazon SQS console. For instructions, see [Receiving a Message](#) in the *Amazon Simple Queue Service Developer Guide "Getting Started"* section.

Configuring event notifications using object key name filtering

When configuring an Amazon S3 event notification, you must specify which supported Amazon S3 event types cause Amazon S3 to send the notification. If an event type that you didn't specify occurs in your S3 bucket, Amazon S3 doesn't send the notification.

You can configure notifications to be filtered by the prefix and suffix of the key name of objects. For example, you can set up a configuration where you're sent a notification only when image files with a ".jpg" file name extension are added to a bucket. Or, you can have a configuration that delivers a

notification to an Amazon SNS topic when an object with the prefix "images/" is added to the bucket, while having notifications for objects with a "logs/" prefix in the same bucket delivered to an AWS Lambda function.

You can set up notification configurations that use object key name filtering in the Amazon S3 console. You can do so by using Amazon S3 APIs through the AWS SDKs or the REST APIs directly. For information about using the console UI to set a notification configuration on a bucket, see [Enabling and configuring event notifications using the Amazon S3 console \(p. 1025\)](#).

Amazon S3 stores the notification configuration as XML in the *notification* subresource associated with a bucket as described in [Using Amazon SQS, Amazon SNS, and Lambda \(p. 1022\)](#). You use the *Filter* XML structure to define the rules for notifications to be filtered by the prefix or suffix of an object key name. For information about the *Filter* XML structure, see [PUT Bucket notification](#) in the *Amazon Simple Storage Service API Reference*.

If an Amazon S3 event notification is configured to use object key name filtering, notifications are only published for objects with a certain key name prefix or suffix. A wild-card character ("*") can't be used in filters as a prefix or suffix to represent any character. If you use any special characters in the value of the prefix or suffix, you must enter them in [URL-encoded \(percent-encoded\) format](#). For more information, see [Object key naming guidelines \(p. 151\)](#).

Notification configurations that use *Filter* cannot define filtering rules with overlapping prefixes, overlapping suffixes, or prefix and suffix overlapping. The following sections have examples of valid notification configurations with object key name filtering. They also contain examples of notification configurations that are not valid because of prefix and suffix overlapping.

Topics

- [Examples of valid notification configurations with object key name filtering \(p. 1034\)](#)
- [Examples of notification configurations with invalid prefix and suffix overlapping \(p. 1036\)](#)

Examples of valid notification configurations with object key name filtering

The following notification configuration contains a queue configuration identifying an Amazon SQS queue for Amazon S3 to publish events to of the `s3:ObjectCreated:Put` type. The events are published whenever an object that has a prefix of `images/` and a `jpg` suffix is PUT to a bucket.

```
<NotificationConfiguration>
  <QueueConfiguration>
    <Id>1</Id>
    <Filter>
      <S3Key>
        <FilterRule>
          <Name>prefix</Name>
          <Value>images/</Value>
        </FilterRule>
        <FilterRule>
          <Name>suffix</Name>
          <Value>.jpg</Value>
        </FilterRule>
      </S3Key>
    </Filter>
    <Queue>arn:aws:sqs:us-west-2:444455556666:s3notificationqueue</Queue>
    <Event>s3:ObjectCreated:Put</Event>
  </QueueConfiguration>
</NotificationConfiguration>
```

The following notification configuration has multiple non-overlapping prefixes. The configuration defines that notifications for PUT requests in the `images/` folder go to queue-A, while notifications for PUT requests in the `logs/` folder go to queue-B.

```

<NotificationConfiguration>
  <QueueConfiguration>
    <Id>1</Id>
    <Filter>
      <S3Key>
        <FilterRule>
          <Name>prefix</Name>
          <Value>images/</Value>
        </FilterRule>
      </S3Key>
    </Filter>
    <Queue>arn:aws:sqs:us-west-2:444455556666:sqs-queue-A</Queue>
    <Event>s3:ObjectCreated:Put</Event>
  </QueueConfiguration>
  <QueueConfiguration>
    <Id>2</Id>
    <Filter>
      <S3Key>
        <FilterRule>
          <Name>prefix</Name>
          <Value>logs/</Value>
        </FilterRule>
      </S3Key>
    </Filter>
    <Queue>arn:aws:sqs:us-west-2:444455556666:sqs-queue-B</Queue>
    <Event>s3:ObjectCreated:Put</Event>
  </QueueConfiguration>
</NotificationConfiguration>

```

The following notification configuration has multiple non-overlapping suffixes. The configuration defines that all .jpg images newly added to the bucket are processed by Lambda cloud-function-A, and all newly added .png images are processed by cloud-function-B. The .png and .jpg suffixes aren't overlapping even though they have the same last letter. If a given string can end with both suffixes, the two suffixes are considered overlapping. A string can't end with both .png and .jpg, so the suffixes in the example configuration aren't overlapping suffixes.

```

<NotificationConfiguration>
  <CloudFunctionConfiguration>
    <Id>1</Id>
    <Filter>
      <S3Key>
        <FilterRule>
          <Name>suffix</Name>
          <Value>.jpg</Value>
        </FilterRule>
      </S3Key>
    </Filter>
    <CloudFunction>arn:aws:lambda:us-west-2:444455556666:cloud-function-A</CloudFunction>
    <Event>s3:ObjectCreated:Put</Event>
  </CloudFunctionConfiguration>
  <CloudFunctionConfiguration>
    <Id>2</Id>
    <Filter>
      <S3Key>
        <FilterRule>
          <Name>suffix</Name>
          <Value>.png</Value>
        </FilterRule>
      </S3Key>
    </Filter>
    <CloudFunction>arn:aws:lambda:us-west-2:444455556666:cloud-function-B</CloudFunction>
    <Event>s3:ObjectCreated:Put</Event>
  </CloudFunctionConfiguration>
</NotificationConfiguration>

```

```
</NotificationConfiguration>
```

Your notification configurations that use `Filter` can't define filtering rules with overlapping prefixes for the same event types. They can only do so, if the overlapping prefixes that are used with suffixes that don't overlap. The following example configuration shows how objects created with a common prefix but non-overlapping suffixes can be delivered to different destinations.

```
<NotificationConfiguration>
  <CloudFunctionConfiguration>
    <Id>1</Id>
    <Filter>
      <S3Key>
        <FilterRule>
          <Name>prefix</Name>
          <Value>images</Value>
        </FilterRule>
        <FilterRule>
          <Name>suffix</Name>
          <Value>.jpg</Value>
        </FilterRule>
      </S3Key>
    </Filter>
    <CloudFunction>arn:aws:lambda:us-west-2:444455556666:cloud-function-A</CloudFunction>
    <Event>s3:ObjectCreated:Put</Event>
  </CloudFunctionConfiguration>
  <CloudFunctionConfiguration>
    <Id>2</Id>
    <Filter>
      <S3Key>
        <FilterRule>
          <Name>prefix</Name>
          <Value>images</Value>
        </FilterRule>
        <FilterRule>
          <Name>suffix</Name>
          <Value>.png</Value>
        </FilterRule>
      </S3Key>
    </Filter>
    <CloudFunction>arn:aws:lambda:us-west-2:444455556666:cloud-function-B</CloudFunction>
    <Event>s3:ObjectCreated:Put</Event>
  </CloudFunctionConfiguration>
</NotificationConfiguration>
```

Examples of notification configurations with invalid prefix and suffix overlapping

For the most part, your notification configurations that use `Filter` can't define filtering rules with overlapping prefixes, overlapping suffixes, or overlapping combinations of prefixes and suffixes for the same event types. You can have overlapping prefixes as long as the suffixes don't overlap. For an example, see [Configuring event notifications using object key name filtering \(p. 1033\)](#).

You can use overlapping object key name filters with different event types. For example, you can create a notification configuration that uses the prefix `image/` for the `ObjectCreated:Put` event type and the prefix `image/` for the `ObjectRemoved:*` event type.

You get an error if you try to save a notification configuration that has invalid overlapping name filters for the same event types when using the Amazon S3 console or API. This section shows examples of notification configurations that aren't valid because of overlapping name filters.

Any existing notification configuration rule is assumed to have a default prefix and suffix that match any other prefix and suffix, respectively. The following notification configuration isn't valid because it has

overlapping prefixes. Specifically, the root prefix overlaps with any other prefix. The same thing is true if you use a suffix instead of a prefix in this example. The root suffix overlaps with any other suffix.

```
<NotificationConfiguration>
  <TopicConfiguration>
    <Topic>arn:aws:sns:us-west-2:444455556666:sns-notification-one</Topic>
    <Event>s3:ObjectCreated:*</Event>
  </TopicConfiguration>
  <TopicConfiguration>
    <Topic>arn:aws:sns:us-west-2:444455556666:sns-notification-two</Topic>
    <Event>s3:ObjectCreated:*</Event>
    <Filter>
      <S3Key>
        <FilterRule>
          <Name>prefix</Name>
          <Value>images</Value>
        </FilterRule>
      </S3Key>
    </Filter>
  </TopicConfiguration>
</NotificationConfiguration>
```

The following notification configuration isn't valid because it has overlapping suffixes. If a given string can end with both suffixes, the two suffixes are considered overlapping. A string can end with jpg and pg. So, the suffixes overlap. The same is true for prefixes. If a given string can begin with both prefixes, the two prefixes are considered overlapping.

```
<NotificationConfiguration>
  <TopicConfiguration>
    <Topic>arn:aws:sns:us-west-2:444455556666:sns-topic-one</Topic>
    <Event>s3:ObjectCreated:*</Event>
    <Filter>
      <S3Key>
        <FilterRule>
          <Name>suffix</Name>
          <Value>jpg</Value>
        </FilterRule>
      </S3Key>
    </Filter>
  </TopicConfiguration>
  <TopicConfiguration>
    <Topic>arn:aws:sns:us-west-2:444455556666:sns-topic-two</Topic>
    <Event>s3:ObjectCreated:Put</Event>
    <Filter>
      <S3Key>
        <FilterRule>
          <Name>suffix</Name>
          <Value>pg</Value>
        </FilterRule>
      </S3Key>
    </Filter>
  </TopicConfiguration>
</NotificationConfiguration>
```

The following notification configuration isn't valid because it has overlapping prefixes and suffixes.

```
<NotificationConfiguration>
  <TopicConfiguration>
    <Topic>arn:aws:sns:us-west-2:444455556666:sns-topic-one</Topic>
    <Event>s3:ObjectCreated:*</Event>
    <Filter>
      <S3Key>
        <FilterRule>
```

```

<Name>prefix</Name>
<Value>images</Value>
</FilterRule>
<FilterRule>
<Name>suffix</Name>
<Value>jpg</Value>
</FilterRule>
</S3Key>
</Filter>
</TopicConfiguration>
<TopicConfiguration>
<Topic>arn:aws:sns:us-west-2:444455556666:sns-topic-two</Topic>
<Event>s3:ObjectCreated:Put</Event>
<Filter>
<S3Key>
<FilterRule>
<Name>suffix</Name>
<Value>jpg</Value>
</FilterRule>
</S3Key>
</Filter>
</TopicConfiguration>
</NotificationConfiguration>

```

Event message structure

The notification message that Amazon S3 sends to publish an event is in the JSON format.

For a general overview and instructions on configuring event notifications, see [Amazon S3 Event Notifications \(p. 1017\)](#).

This example shows *version 2.2* of the event notification JSON structure. Amazon S3 uses *versions 2.1, 2.2, and 2.3* of this event structure. Amazon S3 uses version 2.2 for cross-Region replication event notifications. It uses version 2.3 for S3 Lifecycle, S3 Intelligent-Tiering, object ACL, object tagging, and object restoration delete events. These versions contain extra information specific to these operations. Versions 2.2 and 2.3 are otherwise compatible with version 2.1, which Amazon S3 currently uses for all other event notification types.

```
{
    "Records": [
        {
            "eventVersion": "2.2",
            "eventSource": "aws:s3",
            "awsRegion": "us-west-2",
            "eventTime": "The time, in ISO-8601 format, for example, 1970-01-01T00:00:00.000Z, when Amazon S3 finished processing the request",
            "eventName": "event-type",
            "userIdentity": {
                "principalId": "Amazon-customer-ID-of-the-user-who-caused-the-event"
            },
            "requestParameters": {
                "sourceIPAddress": "ip-address-where-request-came-from"
            },
            "responseElements": {
                "x-amz-request-id": "Amazon S3 generated request ID",
                "x-amz-id-2": "Amazon S3 host that processed the request"
            },
            "s3": {
                "s3SchemaVersion": "1.0",
                "configurationId": "ID found in the bucket notification configuration",
                "bucket": {
                    "name": "bucket-name",
                    "ownerIdentity": {

```

```

        "principalId": "Amazon-customer-ID-of-the-bucket-owner"
    },
    "arn": "bucket-ARN"
},
"object": {
    "key": "object-key",
    "size": "object-size in bytes",
    "eTag": "object eTag",
    "versionId": "object version if bucket is versioning-enabled, otherwise
null",
    "sequencer": "a string representation of a hexadecimal value used to
determine event sequence, only used with PUTs and DELETES"
}
},
"glacierEventData": {
    "restoreEventData": {
        "lifecycleRestorationExpiryTime": "The time, in ISO-8601 format, for
example, 1970-01-01T00:00:00.000Z, of Restore Expiry",
        "lifecycleRestoreStorageClass": "Source storage class for restore"
    }
}
]
}
}

```

Note the following about the event message structure:

- The `eventVersion` key value contains a major and minor version in the form `<major>.<minor>`.

The major version is incremented if Amazon S3 makes a change to the event structure that's not backward compatible. This includes removing a JSON field that's already present or changing how the contents of a field are represented (for example, a date format).

The minor version is incremented if Amazon S3 adds new fields to the event structure. This might occur if new information is provided for some or all existing events. This might also occur if new information is provided on only newly introduced event types. Applications should ignore new fields to stay forward compatible with new minor versions of the event structure.

If new event types are introduced but the structure of the event is otherwise unmodified, the event version doesn't change.

To ensure that your applications can parse the event structure correctly, we recommend that you do an equal-to comparison on the major version number. To ensure that the fields that are expected by your application are present, we also recommend doing a greater-than-or-equal-to comparison on the minor version.

- The `eventName` references the list of [event notification types](#) but doesn't contain the `s3:` prefix.
- The `responseElements` key value is useful if you want to trace a request by following up with AWS Support. Both `x-amz-request-id` and `x-amz-id-2` help Amazon S3 trace an individual request. These values are the same as those that Amazon S3 returns in the response to the request that initiates the events. This is so they can be used to match the event to the request.
- The `s3` key provides information about the bucket and object involved in the event. The object key name value is URL encoded. For example, "red flower.jpg" becomes "red+flower.jpg" (Amazon S3 returns "application/x-www-form-urlencoded" as the content type in the response).
- The `sequencer` key provides a way to determine the sequence of events. Event notifications aren't guaranteed to arrive in the same order that the events occurred. However, notifications from events that create objects (PUTs) and delete objects contain a `sequencer`. It can be used to determine the order of events for a given object key.

If you compare the `sequencer` strings from two event notifications on the same object key, the event notification with the greater `sequencer` hexadecimal value is the event that occurred later. If you're

using event notifications to maintain a separate database or index of your Amazon S3 objects, we recommend that you compare and store the `sequencer` values as you process each event notification.

Note the following:

- You can't use `sequencer` to determine order for events on different object keys.
- The sequencers can be of different lengths. So, to compare these values, first right pad the shorter value with zeros, and then do a lexicographical comparison.
- The `glacierEventData` key is only visible for `s3:ObjectRestore:Completed` events.
- The `restoreEventData` key contains attributes that are related to your restore request.
- The `replicationEventData` key is only visible for replication events.
- The `intelligentTieringEventData` key is only visible for S3 Intelligent-Tiering events.
- The `lifecycleEventData` key is only visible for S3 Lifecycle transition events.

Example messages

The following are examples of Amazon S3 event notification messages.

Amazon S3 test message

After you configure an event notification on a bucket, Amazon S3 sends the following test message.

```
{  
    "Service": "Amazon S3",  
    "Event": "s3:TestEvent",  
    "Time": "2014-10-13T15:57:02.089Z",  
    "Bucket": "bucketname",  
    "RequestId": "5582815E1AEA5ADF",  
    "HostId": "8cLeGAmw098X5cv4Zkwcmo8vvZa3eH3eKxsPzbB9wrR+YstdA6Knx4Ip8EXAMPLE"  
}
```

Example message when an object is created using a PUT request

The following message is an example of a message Amazon S3 sends to publish an `s3:ObjectCreated:Put` event.

```
{  
    "Records": [  
        {  
            "eventVersion": "2.1",  
            "eventSource": "aws:s3",  
            "awsRegion": "us-west-2",  
            "eventTime": "1970-01-01T00:00:00.000Z",  
            "eventName": "ObjectCreated:Put",  
            "userIdentity": {  
                "principalId": "AIDAJDPLRKG7UEXAMPLE"  
            },  
            "requestParameters": {  
                "sourceIPAddress": "127.0.0.1"  
            },  
            "responseElements": {  
                "x-amz-request-id": "C3D13FE58DE4C810",  
                "x-amz-id-2": "FMyUVURIY8/IgAtTv8xRjskZQpcIZ9KG4V5Wp6S7S/JRWeUWerMUE5JgHvANOjpD"  
            },  
            "s3": {  
                "s3SchemaVersion": "1.0",  
                "configurationId": "testConfigRule",  
                "bucket": {  
                    "name": "mybucket"  
                },  
                "object": {  
                    "key": "myobject",  
                    "size": 1048576,  
                    "type": "Image/JPEG",  
                    "versionId": "12345678901234567890123456789012"  
                }  
            }  
        }  
    ]  
}
```

```

        "name": "mybucket",
        "ownerIdentity": {
            "principalId": "A3NL1KOZZKExample"
        },
        "arn": "arn:aws:s3:::mybucket"
    },
    "object": {
        "key": "HappyFace.jpg",
        "size": 1024,
        "eTag": "d41d8cd98f00b204e9800998ecf8427e",
        "versionId": "096fKKXTRTl3on89fVO.nfljtsv6qko",
        "sequencer": "0055AED6DCD90281E5"
    }
}
]
}
}

```

For a definition of each IAM identification prefix (for example, AIDA, AROA, AGPA), see [IAM identifiers](#) in the *IAM User Guide*.

Using EventBridge

Amazon S3 can send events to Amazon EventBridge whenever certain events happen in your bucket. Unlike other destinations, you don't need to select which event types you want to deliver. After EventBridge is enabled, all events below are sent to EventBridge. You can use EventBridge rules to route events to additional targets. The following lists the events Amazon S3 sends to EventBridge.

Event type	Description
<i>Object Created</i>	An object was created. The reason field in the event message structure indicates which S3 API was used to create the object: PutObject , POST Object , CopyObject , or CompleteMultipartUpload .
<i>Object Deleted (DeleteObject)</i> <i>Object Deleted (Lifecycle expiration)</i>	An object was deleted. When an object is deleted using an S3 API call, the reason field is set to DeleteObject. When an object is deleted by an S3 Lifecycle expiration rule, the reason field is set to Lifecycle Expiration. For more information, see Expiring objects (p. 707) . When an unversioned object is deleted, or a versioned object is permanently deleted, the deletion-type field is set to Permanently Deleted. When a delete marker is created for a versioned object, the deletion-type field is set to Delete Marker Created. For more information, see Deleting object versions from a versioning-enabled bucket (p. 659) .
<i>Object Restore Initiated</i>	An object restore was initiated from S3 Glacier or S3 Glacier Deep Archive storage class or from S3 Intelligent-Tiering Archive Access or Deep Archive Access tier. For more information, see Working with archived objects (p. 670) .
<i>Object Restore Completed</i>	An object restore was completed.
<i>Object Restore Expired</i>	The temporary copy of an object restored from S3 Glacier or S3 Glacier Deep Archive expired and was deleted.

Event type	Description
<i>Object Storage Class Changed</i>	An object was transitioned to a different storage class. For more information, see Transitioning objects using Amazon S3 Lifecycle (p. 703) .
<i>Object Access Tier Changed</i>	An object was transitioned to the S3 Intelligent-Tiering Archive Access tier or Deep Archive Access tier. For more information, see Amazon S3 Intelligent-Tiering (p. 693) .
<i>Object ACL Updated</i>	An object's access control list (ACL) was set using PutObjectACL. An event is not generated when a request results in no change to an object's ACL. For more information, see Access control list (ACL) overview (p. 554) .
<i>Object Tags Added</i>	A set of tags was added to an object using PutObjectTagging. For more information, see Categorizing your storage using tags (p. 825) .
<i>Object Tags Deleted</i>	All tags were removed from an object using DeleteObjectTagging. For more information, see Categorizing your storage using tags (p. 825) .

Note

For more information about how Amazon S3 event types map to EventBridge event types, see [Amazon EventBridge mapping and troubleshooting \(p. 1046\)](#).

You can use Amazon S3 Event Notifications with EventBridge to write rules that take actions when an event occurs in your bucket. For example, you can have it send you a notification. For more information, see [What is EventBridge](#) in the *Amazon EventBridge User Guide*.

For information about pricing, see [Amazon EventBridge pricing](#).

Topics

- [Amazon EventBridge permissions \(p. 1042\)](#)
- [Enabling Amazon EventBridge \(p. 1042\)](#)
- [EventBridge event message structure \(p. 1043\)](#)
- [Amazon EventBridge mapping and troubleshooting \(p. 1046\)](#)

Amazon EventBridge permissions

Amazon S3 does not require any additional permissions to deliver events to Amazon EventBridge.

Enabling Amazon EventBridge

You can enable Amazon EventBridge using the S3 console, AWS Command Line Interface (AWS CLI), or Amazon S3 REST API.

Using the S3 console

To enable EventBridge event delivery in the S3 console.

1. Sign in to the AWS Management Console and open the Amazon S3 console at <https://console.aws.amazon.com/s3/>.
2. In the **Buckets** list, choose the name of the bucket that you want to enable events for.

3. Choose **Properties**.
4. Navigate to the **Event Notifications** section and find the **Amazon EventBridge** subsection. Choose **Edit**.
5. Under **Send notifications to Amazon EventBridge for all events in this bucket** choose **On**.

Note

After you enable EventBridge, it takes around five minutes for the changes to take effect.

Using the AWS CLI

The following example creates a bucket notification configuration for bucket **DOC-EXAMPLE-BUCKET1** with Amazon EventBridge enabled.

```
aws s3api put-bucket-notification-configuration --bucket DOC-EXAMPLE-BUCKET1 --  
notification-configuration '{ "EventBridgeConfiguration": {} }'
```

Using the REST API

You can programmatically enable Amazon EventBridge on a bucket by calling the Amazon S3 REST API. For more information see, see [PutBucketNotificationConfiguration](#) in the *Amazon Simple Storage Service API Reference*.

The following example shows the XML used to create a bucket notification configuration with Amazon EventBridge enabled.

```
<NotificationConfiguration xmlns="http://s3.amazonaws.com/doc/2006-03-01/">  
  <EventBridgeConfiguration>  
    </EventBridgeConfiguration>  
</NotificationConfiguration>
```

Creating EventBridge rules

Once enabled you can create Amazon EventBridge rules for certain tasks. For example, you can send email notifications when an object is created. For a full tutorial, see [Tutorial: Send a notification when an Amazon S3 object is created](#) in the *Amazon EventBridge User Guide*.

EventBridge event message structure

The notification message that Amazon S3 sends to publish an event is in the JSON format. When Amazon S3 sends an event to Amazon EventBridge, the following fields are present.

- **version** — Currently 0 (zero) for all events.
- **id** — A Version 4 UUID generated for every event.
- **detail-type** — The type of event that's being sent. See [Using EventBridge \(p. 1041\)](#) for a list of event types.
- **source** — Identifies the service that generated the event.
- **account** — The 12-digit AWS account ID of the bucket owner.
- **time** — The time the event occurred.
- **region** — Identifies the AWS Region of the bucket.
- **resource** — A JSON array that contains the Amazon Resource Name (ARN) of the bucket.
- **detail** — A JSON object that contains information about the event. For more information about what can be included in this field, see [Event message detail field \(p. 1046\)](#).

Event message structure examples

The following are examples of some of the Amazon S3 event notification messages that can be sent to Amazon EventBridge.

Object created

```
{  
    "version": "0",  
    "id": "17793124-05d4-b198-2fde-7edec63b103",  
    "detail-type": "Object Created",  
    "source": "aws.s3",  
    "account": "111122223333",  
    "time": "2021-11-12T00:00:00Z",  
    "region": "ca-central-1",  
    "resources": [  
        "arn:aws:s3:::DOC-EXAMPLE-BUCKET1"  
    ],  
    "detail": {  
        "version": "0",  
        "bucket": {  
            "name": "DOC-EXAMPLE-BUCKET1"  
        },  
        "object": {  
            "key": "example-key",  
            "size": 5,  
            "etag": "b1946ac92492d2347c6235b4d2611184",  
            "version-id": "IYV3p45BT0ac8hjHg1houSdS1a.Mro8e",  
            "sequencer": "617f08299329d189"  
        },  
        "request-id": "N4N7GDK58NMKJ12R",  
        "requester": "123456789012",  
        "source-ip-address": "1.2.3.4",  
        "reason": "PutObject"  
    }  
}
```

Object deleted (using DeleteObject)

```
{  
    "version": "0",  
    "id": "2ee9cc15-d022-99ea-1fb8-1b1bac4850f9",  
    "detail-type": "Object Deleted",  
    "source": "aws.s3",  
    "account": "111122223333",  
    "time": "2021-11-12T00:00:00Z",  
    "region": "ca-central-1",  
    "resources": [  
        "arn:aws:s3:::DOC-EXAMPLE-BUCKET1"  
    ],  
    "detail": {  
        "version": "0",  
        "bucket": {  
            "name": "DOC-EXAMPLE-BUCKET1"  
        },  
        "object": {  
            "key": "example-key",  
            "etag": "d41d8cd98f00b204e9800998ecf8427e",  
            "version-id": "1QW9g1Z99LUNbvaayVpW9xD1OLU.qxgF",  
            "sequencer": "617f0837b476e463"  
        },  
        "request-id": "0BH729840619AG5K",  
        "status": "Success"  
    }  
}
```

```
        "requester": "123456789012",
        "source-ip-address": "1.2.3.4",
        "reason": "DeleteObject",
        "deletion-type": "Delete Marker Created"
    }
}
```

Object deleted (using lifecycle expiration)

```
{
    "version": "0",
    "id": "ad1de317-e409-eba2-9552-30113f8d88e3",
    "detail-type": "Object Deleted",
    "source": "aws.s3",
    "account": "111122223333",
    "time": "2021-11-12T00:00:00Z",
    "region": "ca-central-1",
    "resources": [
        "arn:aws:s3:::DOC-EXAMPLE-BUCKET1"
    ],
    "detail": {
        "version": "0",
        "bucket": {
            "name": "DOC-EXAMPLE-BUCKET1"
        },
        "object": {
            "key": "example-key",
            "etag": "d41d8cd98f00b204e9800998ecf8427e",
            "version-id": "mtB0cV.jejK63XkRNceanNMC.qXPWLeK",
            "sequencer": "617b3980000000000"
        },
        "request-id": "20EB74C14654DC47",
        "requester": "s3.amazonaws.com",
        "reason": "Lifecycle Expiration",
        "deletion-type": "Delete Marker Created"
    }
}
```

Object restore completed

```
{
    "version": "0",
    "id": "6924de0d-13e2-6bbf-c0c1-b903b753565e",
    "detail-type": "Object Restore Completed",
    "source": "aws.s3",
    "account": "111122223333",
    "time": "2021-11-12T00:00:00Z",
    "region": "ca-central-1",
    "resources": [
        "arn:aws:s3:::DOC-EXAMPLE-BUCKET1"
    ],
    "detail": {
        "version": "0",
        "bucket": {
            "name": "DOC-EXAMPLE-BUCKET1"
        },
        "object": {
            "key": "example-key",
            "size": 5,
            "etag": "b1946ac92492d2347c6235b4d2611184",
            "version-id": "KKsjUC1.6gIjqtvhfg5AdMI0eCePIiT3"
        }
    }
}
```

```
        },
        "request-id": "189F19CB7FB1B6A4",
        "requester": "s3.amazonaws.com",
        "restore-expiry-time": "2021-11-13T00:00:00Z",
        "source-storage-class": "GLACIER"
    }
}
```

Event message detail field

The detail field contains a JSON object with information about the event. The following fields may be present in the detail field.

- **version** — Currently 0 (zero) for all events.
- **bucket** — Information about the Amazon S3 bucket involved in the event.
- **object** — Information about the Amazon S3 object involved in the event.
- **request-id** — Request ID in S3 response.
- **requester** — AWS account ID or AWS service principal of requester.
- **source-ip-address** — Source IP address of S3 request. Only present for events triggered by an S3 request.
- **reason** — For **Object Created** events, the S3 API used to create the object: [PutObject](#), [POST Object](#), [CopyObject](#), or [CompleteMultipartUpload](#). For **Object Deleted** events, this is set to [DeleteObject](#) when an object is deleted by an S3 API call, or [Lifecycle Expiration](#) when an object is deleted by an S3 Lifecycle expiration rule. For more information, see [Expiring objects \(p. 707\)](#).
- **deletion-type** — For **Object Deleted** events, when an unversioned object is deleted, or a versioned object is permanently deleted, this is set to [Permanently Deleted](#). When a delete marker is created for a versioned object, this is set to [Delete Marker Created](#). For more information, see [Deleting object versions from a versioning-enabled bucket \(p. 659\)](#).
- **restore-expiry-time** — For **Object Restore Completed** events, the time when the temporary copy of the object will be deleted from S3. For more information, see [Working with archived objects \(p. 670\)](#).
- **source-storage-class** — For **Object Restore Initiated** and **Object Restore Completed** events, the storage class of the object being restored. For more information, see [Working with archived objects \(p. 670\)](#).
- **destination-storage-class** — For **Object Storage Class Changed** events, the new storage class of the object. For more information, see [Transitioning objects using Amazon S3 Lifecycle \(p. 703\)](#).
- **destination-access-tier** — For **Object Access Tier Changed** events, the new access tier of the object. For more information, see [Amazon S3 Intelligent-Tiering \(p. 693\)](#).

Amazon EventBridge mapping and troubleshooting

The following table describes how Amazon S3 event types are mapped to Amazon EventBridge event types.

S3 event type	Amazon EventBridge detail type
ObjectCreated:Put	Object Created
ObjectCreated:Post	
ObjectCreated:Copy	
ObjectCreated:CompleteMultipartUpload	

S3 event type	Amazon EventBridge detail type
ObjectRemoved:Delete	Object Deleted
ObjectRemoved:DeleteMarkerCreated	
LifecycleExpiration:Delete	
LifecycleExpiration:DeleteMarkerCreated	
ObjectRestore:Post	Object Restore Initiated
ObjectRestore:Completed	Object Restore Completed
ObjectRestore:Delete	Object Restore Expired
LifecycleTransition	Object Storage Class Changed
IntelligentTiering	Object Access Tier Changed
ObjectTagging:Put	Object Tags Added
ObjectTagging:Delete	Object Tags Deleted
ObjectAcl:Put	Object ACL Updated

Amazon EventBridge troubleshooting

For information about how to troubleshoot EventBridge, see [Troubleshooting Amazon EventBridge](#) in the *Amazon EventBridge User Guide*.

Using analytics and insights

You can use analytics and insights in Amazon S3 to understand, analyze, and optimize your storage usage . For more information, see the topics below.

Topics

- [Amazon S3 analytics – Storage Class Analysis \(p. 1048\)](#)
- [Assessing your storage activity and usage with Amazon S3 Storage Lens \(p. 1053\)](#)
- [Tracing Amazon S3 requests using AWS X-Ray \(p. 1114\)](#)

Amazon S3 analytics – Storage Class Analysis

By using Amazon S3 analytics *Storage Class Analysis* you can analyze storage access patterns to help you decide when to transition the right data to the right storage class. This new Amazon S3 analytics feature observes data access patterns to help you determine when to transition less frequently accessed STANDARD storage to the STANDARD_IA (IA, for infrequent access) storage class. For more information about storage classes, see [Using Amazon S3 storage classes \(p. 688\)](#).

After storage class analysis observes the infrequent access patterns of a filtered set of data over a period of time, you can use the analysis results to help you improve your lifecycle policies. You can configure storage class analysis to analyze all the objects in a bucket. Or, you can configure filters to group objects together for analysis by common prefix (that is, objects that have names that begin with a common string), by object tags, or by both prefix and tags. You'll most likely find that filtering by object groups is the best way to benefit from storage class analysis.

Important

Storage class analysis only provides recommendations for Standard to Standard IA classes.

You can have multiple storage class analysis filters per bucket, up to 1,000, and will receive a separate analysis for each filter. Multiple filter configurations allow you analyze specific groups of objects to improve your lifecycle policies that transition objects to STANDARD_IA.

Storage class analysis provides storage usage visualizations in the Amazon S3 console that are updated daily. You can also export this daily usage data to an S3 bucket and view them in a spreadsheet application, or with business intelligence tools, like Amazon QuickSight.

There are costs associated with the storage class analysis. For pricing information, see *Management and replication Amazon S3 pricing*.

Topics

- [How do I set up storage class analysis? \(p. 1048\)](#)
- [How do I use storage class analysis? \(p. 1049\)](#)
- [How can I export storage class analysis data? \(p. 1050\)](#)
- [Configuring storage class analysis \(p. 1051\)](#)

How do I set up storage class analysis?

You set up storage class analysis by configuring what object data you want to analyze. You can configure storage class analysis to do the following:

- **Analyze the entire contents of a bucket.**

You'll receive an analysis for all the objects in the bucket.

- **Analyze objects grouped together by prefix and tags.**

You can configure filters that group objects together for analysis by prefix, or by object tags, or by a combination of prefix and tags. You receive a separate analysis for each filter you configure. You can have multiple filter configurations per bucket, up to 1,000.

- **Export analysis data.**

When you configure storage class analysis for a bucket or filter, you can choose to have the analysis data exported to a file each day. The analysis for the day is added to the file to form a historic analysis log for the configured filter. The file is updated daily at the destination of your choice. When selecting data to export, you specify a destination bucket and optional destination prefix where the file is written.

You can use the Amazon S3 console, the REST API, or the AWS CLI or AWS SDKs to configure storage class analysis.

- For information about how to configure storage class analysis in the Amazon S3 console, see [Configuring storage class analysis \(p. 1051\)](#).
- To use the Amazon S3 API, use the [PutBucketAnalyticsConfiguration](#) REST API, or the equivalent, from the AWS CLI or AWS SDKs.

How do I use storage class analysis?

You use storage class analysis to observe your data access patterns over time to gather information to help you improve the lifecycle management of your STANDARD_IA storage. After you configure a filter, you'll start seeing data analysis based on the filter in the Amazon S3 console in 24 to 48 hours. However, storage class analysis observes the access patterns of a filtered data set for 30 days or longer to gather information for analysis before giving a result. The analysis continues to run after the initial result and updates the result as the access patterns change.

When you first configure a filter, the Amazon S3 console may take a moment to analyze your data.

Storage class analysis observes the access patterns of a filtered object data set for 30 days or longer to gather enough information for the analysis. After storage class analysis has gathered sufficient information, you'll see a message in the Amazon S3 console that analysis is complete.

When performing the analysis for infrequently accessed objects storage class analysis looks at the filtered set of objects grouped together based on age since they were uploaded to Amazon S3. Storage class analysis determines if the age group is infrequently accessed by looking at the following factors for the filtered data set:

- Objects in the STANDARD storage class that are larger than 128 KB.
- How much average total storage you have per age group.
- Average number of bytes transferred out (not frequency) per age group.
- Analytics export data only includes requests with data relevant to storage class analysis. This might cause differences in the number of requests, and the total upload and request bytes compared to what are shown in storage metrics or tracked by your own internal systems.
- Failed GET and PUT requests are not counted for the analysis. However, you will see failed requests in storage metrics.

How Much of My Storage did I Retrieve?

The Amazon S3 console graphs how much of the storage in the filtered data set has been retrieved for the observation period.

What Percentage of My Storage did I Retrieve?

The Amazon S3 console also graphs what percentage of the storage in the filtered data set has been retrieved for the observation period.

As stated earlier in this topic, when you are performing the analysis for infrequently accessed objects, storage class analysis looks at the filtered set of objects grouped together based on the age since they were uploaded to Amazon S3. The storage class analysis uses the following predefined object age groups:

- Amazon S3 Objects less than 15 days old
- Amazon S3 Objects 15-29 days old
- Amazon S3 Objects 30-44 days old
- Amazon S3 Objects 45-59 days old
- Amazon S3 Objects 60-74 days old
- Amazon S3 Objects 75-89 days old
- Amazon S3 Objects 90-119 days old
- Amazon S3 Objects 120-149 days old
- Amazon S3 Objects 150-179 days old
- Amazon S3 Objects 180-364 days old
- Amazon S3 Objects 365-729 days old
- Amazon S3 Objects 730 days and older

Usually it takes about 30 days of observing access patterns to gather enough information for an analysis result. It might take longer than 30 days, depending on the unique access pattern of your data. However, after you configure a filter you'll start seeing data analysis based on the filter in the Amazon S3 console in 24 to 48 hours. You can see analysis on a daily basis of object access broken down by object age group in the Amazon S3 console.

How Much of My Storage is Infrequently Accessed?

The Amazon S3 console shows the access patterns grouped by the predefined object age groups. The **Frequently accessed** or **Infrequently accessed** text shown is meant as a visual aid to help you in the lifecycle creation process.

How can I export storage class analysis data?

You can choose to have storage class analysis export analysis reports to a comma-separated values (CSV) flat file. Reports are updated daily and are based on the object age group filters you configure. When using the Amazon S3 console you can choose the export report option when you create a filter. When selecting data export you specify a destination bucket and optional destination prefix where the file is written. You can export the data to a destination bucket in a different account. The destination bucket must be in the same region as the bucket that you configure to be analyzed.

You must create a bucket policy on the destination bucket to grant permissions to Amazon S3 to verify what AWS account owns the bucket and to write objects to the bucket in the defined location. For an example policy, see [Granting permissions for Amazon S3 Inventory and Amazon S3 analytics \(p. 497\)](#).

After you configure storage class analysis reports, you start getting the exported report daily after 24 hours. After that, Amazon S3 continues monitoring and providing daily exports.

You can open the CSV file in a spreadsheet application or import the file into other applications like [Amazon QuickSight](#). For information on using Amazon S3 files with Amazon QuickSight, see [Create a Data Set Using Amazon S3 Files](#) in the [Amazon QuickSight User Guide](#).

Data in the exported file is sorted by date within object age group as shown in following examples. If the storage class is STANDARD the row also contains data for the columns ObjectAgeForSIATransition and RecommendedObjectAgeForSIATransition.

Date	ConfigId	Filter	StorageClass	ObjectAge	ObjectCount	DataUploaded_MB	Storage_MB	DataRetrieved_MB	GetRequestCount	Cost
8/17/2021	SalesMaterial	SalesMaterial	STANDARD	000-014			0.4313			
9/2/2021	SalesMaterial	SalesMaterial	STANDARD	000-014						
8/22/2021	SalesMaterial	SalesMaterial	STANDARD	000-014			0.4313			
8/25/2021	SalesMaterial	SalesMaterial	STANDARD	000-014			0.4313			
9/6/2021	SalesMaterial	SalesMaterial	STANDARD	000-014						
8/30/2021	SalesMaterial	SalesMaterial	STANDARD	000-014						
8/28/2021	SalesMaterial	SalesMaterial	STANDARD	000-014						
8/21/2021	SalesMaterial	SalesMaterial	STANDARD	000-014			0.4313			
9/5/2021	SalesMaterial	SalesMaterial	STANDARD	000-014						

At the end of the report the object age group is given as ALL. The ALL rows contain cumulative totals, including objects smaller than 128 KB, for all the age groups for that day.

8/24/2021	SalesMaterial	SalesMaterial	STANDARD	ALL	3	0.4599				
9/3/2021	SalesMaterial	SalesMaterial	STANDARD	ALL	3	0.4599				
8/28/2021	SalesMaterial	SalesMaterial	STANDARD	ALL	3	0.4599				
8/17/2021	SalesMaterial	SalesMaterial	STANDARD	ALL	3	0.4599				
8/25/2021	SalesMaterial	SalesMaterial	STANDARD	ALL	3	0.4599				
9/6/2021	SalesMaterial	SalesMaterial	STANDARD	ALL	3	0.4599				
9/4/2021	SalesMaterial	SalesMaterial	STANDARD	ALL	3	0.4599				
8/22/2021	SalesMaterial	SalesMaterial	STANDARD	ALL	3	0.4599				
8/21/2021	SalesMaterial	SalesMaterial	STANDARD	ALL	3	0.4599				
8/30/2021	SalesMaterial	SalesMaterial	STANDARD	ALL	3	0.4599				
8/20/2021	SalesMaterial	SalesMaterial	STANDARD	ALL	3	0.4599				

The next section describes the columns used in the report.

Exported file layout

The following table describe the layout of the exported file.

Configuring storage class analysis

By using the Amazon S3 analytics storage class analysis tool, you can analyze storage access patterns to help you decide when to transition the right data to the right storage class. Storage class analysis observes data access patterns to help you determine when to transition less frequently accessed STANDARD storage to the STANDARD_IA (IA, for infrequent access) storage class. For more information about STANDARD_IA, see the [Amazon S3 FAQ](#) and [Using Amazon S3 storage classes \(p. 688\)](#).

You set up storage class analysis by configuring what object data you want to analyze. You can configure storage class analysis to do the following:

- **Analyze the entire contents of a bucket.**

You'll receive an analysis for all the objects in the bucket.

- **Analyze objects grouped together by prefix and tags.**

You can configure filters that group objects together for analysis by prefix, or by object tags, or by a combination of prefix and tags. You receive a separate analysis for each filter you configure. You can have multiple filter configurations per bucket, up to 1,000.

- **Export analysis data.**

When you configure storage class analysis for a bucket or filter, you can choose to have the analysis data exported to a file each day. The analysis for the day is added to the file to form a historic analysis log for the configured filter. The file is updated daily at the destination of your choice. When selecting

data to export, you specify a destination bucket and optional destination prefix where the file is written.

You can use the Amazon S3 console, the REST API, or the AWS CLI or AWS SDKs to configure storage class analysis.

Important

Storage class analysis does not give recommendations for transitions to the ONEZONE_IA or S3 Glacier Flexible Retrieval storage classes.

If you want to configure storage class analysis to export your findings as a .csv file and the destination bucket uses default bucket encryption with a AWS KMS key, you must update the AWS KMS key policy to grant Amazon S3 permission to encrypt the .csv file. For instructions, see [Granting Amazon S3 permission to use your AWS KMS key for encryption \(p. 743\)](#).

For more information about analytics, see [Amazon S3 analytics – Storage Class Analysis \(p. 1048\)](#).

Using the S3 console

To configure storage class analysis

1. Sign in to the AWS Management Console and open the Amazon S3 console at <https://console.aws.amazon.com/s3/>.
2. In the **Buckets** list, choose the name of the bucket for which you want to configure storage class analysis.
3. Choose the **Metrics** tab.
4. Under **Storage Class Analysis**, choose **Create analytics configuration**.
5. Type a name for the filter. If you want to analyze the whole bucket, leave the **Prefix** field empty.
6. In the **Prefix** field, type text for the prefix for the objects that you want to analyze.
7. To add a tag, choose **Add tag**. Enter a key and value for the tag. You can enter one prefix and multiple tags.
8. Optionally, you can choose **Enable** under **Export CSV** to export analysis reports to a comma-separated values (.csv) flat file. Choose a destination bucket where the file can be stored. You can type a prefix for the destination bucket. The destination bucket must be in the same AWS Region as the bucket for which you are setting up the analysis. The destination bucket can be in a different AWS account.

If the destination bucket for the .csv file uses default bucket encryption with a KMS key, you must update the AWS KMS key policy to grant Amazon S3 permission to encrypt the .csv file. For instructions, see [Granting Amazon S3 permission to use your AWS KMS key for encryption \(p. 743\)](#).

9. Choose **Create Configuration**.

Amazon S3 creates a bucket policy on the destination bucket that grants Amazon S3 write permission. This allows it to write the export data to the bucket.

If an error occurs when you try to create the bucket policy, you'll be given instructions on how to fix it. For example, if you chose a destination bucket in another AWS account and do not have permissions to read and write to the bucket policy, you'll see the following message. You must have the destination bucket owner add the displayed bucket policy to the destination bucket. If the policy is not added to the destination bucket you won't get the export data because Amazon S3 doesn't have permission to write to the destination bucket. If the source bucket is owned by a different account than that of the current user, then the correct account ID of the source bucket must be substituted in the policy.

For information about the exported data and how the filter works, see [Amazon S3 analytics – Storage Class Analysis \(p. 1048\)](#).

Using the REST API

To configure Storage Class Analysis using the REST API, use the [PutBucketAnalyticsConfiguration](#). You can also use the equivalent operation with the AWS CLI or AWS SDKs.

You can use the following REST APIs to work with Storage Class Analysis:

- [DELETE Bucket Analytics configuration](#)
- [GET Bucket Analytics configuration](#)
- [List Bucket Analytics Configuration](#)

Assessing your storage activity and usage with Amazon S3 Storage Lens

Amazon S3 Storage Lens aggregates your usage and activity metrics and displays the information in the account snapshot on the Amazon S3 console home (**Buckets**) page, interactive dashboards, or through a metrics export that you can download in CSV or Parquet format. You can use the dashboard to visualize insights and trends, flag outliers, and receive recommendations for optimizing storage costs and applying data protection best practices. You can use S3 Storage Lens through the AWS Management Console, AWS CLI, AWS SDKs, or REST API.

How S3 Storage Lens works

Amazon S3 Storage Lens provides a single view of usage and activity across your Amazon S3 storage. It has drilldown options to generate insights at the organization, account, bucket, object, or even prefix level. It analyzes storage metrics to deliver contextual recommendations to help you optimize storage costs and apply best practices for protecting your data.

You can use S3 Storage Lens to generate summary insights, such as finding out how much storage you have across your entire organization, or which are the fastest growing buckets and prefixes. Identify outliers in your storage metrics, and then drill down to further investigate the source of the spike in usage or activity.

You can assess your storage based on S3 best practices, such as analyzing the percentage of your buckets that have encryption or S3 Object Lock enabled. And you can identify potential cost savings opportunities, for example, by analyzing your request activity per bucket to find buckets where objects could be transitioned to a lower-cost storage class. For more information about S3 Storage Lens concepts and terminology, see [Understanding Amazon S3 Storage Lens \(p. 1054\)](#).

Default dashboard

On the [S3 console](#), S3 Storage Lens provides an interactive *default dashboard* that is updated daily. Metrics from this dashboard are also summarized in your account snapshot on the S3 console home (**Buckets**) page. You can create other dashboards and scope them by account (for AWS Organizations users), AWS Regions, and S3 buckets to provide [usage metrics](#) for free. For an additional charge, you can upgrade to receive [advanced metrics and recommendations](#). These include usage metrics with prefix-level aggregation, activity metrics aggregated by bucket, contextual recommendations (available only in the Amazon S3 console), and Amazon CloudWatch publishing. For information about working with your S3 Storage Lens dashboard, see [Using Amazon S3 Storage Lens on the console \(p. 1086\)](#).

Metrics export

In addition to viewing the dashboard on the S3 console, you can export metrics in CSV or Parquet format to an S3 bucket of your choice for further analysis. For more information, see [Viewing Amazon S3 Storage Lens metrics using a data export \(p. 1064\)](#).

Amazon CloudWatch publishing option

You can publish S3 Storage Lens usage and activity metrics to Amazon CloudWatch to create a unified view of your operational health in CloudWatch [dashboards](#). You can also use CloudWatch features like alarms and triggered actions, metric math, and anomaly detection to monitor and take action on S3 Storage Lens metrics. In addition, the CloudWatch APIs enable applications, including third-party providers, to access your S3 Storage Lens metrics. The CloudWatch publishing option is available for dashboards upgraded to S3 Storage Lens *advanced metrics and recommendations*. For more information about support for S3 Storage Lens metrics in CloudWatch, see [Monitor S3 Storage Lens metrics in CloudWatch \(p. 1068\)](#).

For more information about using S3 Storage Lens, see [the section called "Working with S3 Storage Lens" \(p. 1086\)](#). For information about S3 Storage Lens pricing, see [Amazon S3 pricing](#).

Topics

- [Understanding Amazon S3 Storage Lens \(p. 1054\)](#)
- [Using Amazon S3 Storage Lens with AWS Organizations \(p. 1059\)](#)
- [Amazon S3 Storage Lens permissions \(p. 1061\)](#)
- [Viewing storage usage and activity metrics with Amazon S3 Storage Lens \(p. 1063\)](#)
- [Using Amazon S3 Storage Lens to optimize your storage costs \(p. 1078\)](#)
- [Amazon S3 Storage Lens metrics glossary \(p. 1081\)](#)
- [Working with Amazon S3 Storage Lens using the console and API \(p. 1086\)](#)

Understanding Amazon S3 Storage Lens

Amazon S3 Storage Lens provides a single view of object storage usage and activity across your entire Amazon S3 storage. It includes drilldown options to generate insights at the organization, account, Region, bucket, or even prefix level.

Amazon S3 Storage Lens aggregates your usage and activity metrics and displays the information in the account snapshot on the Amazon S3 console home (**Buckets**) page, interactive dashboards, or through a metrics export that you can download in CSV or Parquet format. You can use the dashboard to visualize insights and trends, flag outliers, and receive recommendations for optimizing storage costs and applying data protection best practices. You can use S3 Storage Lens through the AWS Management Console, AWS CLI, AWS SDKs, or REST API.

Amazon S3 Storage Lens concepts and terminology

This section contains the terminology and concepts that are essential for understanding and using Amazon S3 Storage Lens successfully.

Topics

- [Configuration \(p. 1055\)](#)
- [Default dashboard \(p. 1055\)](#)
- [Dashboards \(p. 1055\)](#)
- [Account snapshot \(p. 1055\)](#)
- [Metrics export \(p. 1056\)](#)
- [Home Region \(p. 1056\)](#)
- [Retention period \(p. 1057\)](#)
- [Metrics types \(p. 1057\)](#)
- [Recommendations \(p. 1057\)](#)

- [Metrics selection \(p. 1058\)](#)
- [S3 Storage Lens and AWS Organizations \(p. 1059\)](#)

Configuration

Amazon S3 Storage Lens requires a *configuration* that contains the properties that are used to aggregate metrics on your behalf for a single dashboard or export. This includes all or partial sections of your organization account's storage, including filtering by Region, bucket, and prefix-level (available only with advanced metrics) scope. It includes information about whether you chose *free metrics* or *advanced metrics and recommendations*. It also includes whether a metrics export is required, and information about where to place the metrics export if applicable.

Default dashboard

The S3 Storage Lens default dashboard on the console is named **default-account-dashboard**. S3 preconfigures this dashboard to visualize the summarized insights and trends of your entire account's aggregated storage usage and activity metrics, and updates them daily in the Amazon S3 console. You can't modify the configuration scope of the default dashboard, but you can upgrade the metrics selection from **free metrics** to the paid **advanced metrics and recommendations**. You can also configure the optional metrics export, or even disable the dashboard. However, you can't delete the default dashboard.

Note

If you disable your default dashboard, it is no longer updated, and you will no longer receive any new daily metrics in S3 Storage Lens, or in the account snapshot on S3 home (**Buckets**) page.

You can still see historic data in the dashboard until the 14-day period that data is available for queries expires, or 15 months if you are subscribed to *advanced metrics and recommendations* for that dashboard. You can re-enable the dashboard within the expiration period to access this data.

Dashboards

You can also use Amazon S3 Storage Lens to configure dashboards that visualize summarized insights and trends of aggregated storage usage and activity metrics that you can configure, updated daily on the Amazon S3 console. You can create and modify S3 Storage Lens dashboards to express all or partial sections of your organization or account's storage. You can filter by AWS Region, bucket, and prefix (available only with advanced metrics and recommendations). You can also disable or delete dashboards.

Note

- You can use S3 Storage Lens to create up to 50 dashboards per home Region.
- If you disable a dashboard, it is no longer updated, and you will no longer receive any new daily metrics. You can still see historic data until the 14-day expiration period (or 15 months, if you subscribed to advanced metrics and recommendations for that dashboard). You can re-enable the dashboard within the expiration period to access this data.
- If you delete your dashboard, you lose all your dashboard configuration settings. You will no longer receive any new daily metrics, and you also lose access to the historical data associated with that dashboard. If you want to access the historic data for a deleted dashboard, you must create another dashboard with the same name in the same home Region.
- Organization-level dashboards can only be limited to a Regional scope.

Account snapshot

The S3 Storage Lens *account snapshot* displays your total storage, object count, and average object size on the S3 console home (**Buckets**) page by summarizing metrics from your default dashboard. This gives you quick access to insights about your storage without having to leave the **Buckets** page. The account

snapshot also provides one-click access to your S3 Storage Lens page, where you can conduct a deeper analysis of your usage and activity trends by AWS Region, storage class, bucket, or prefix.

You can upgrade the metrics selection in your **default-account-dashboard** from **free metrics** to the paid **advanced metrics and recommendations**. You can then display all requests, bytes uploaded, and bytes downloaded in the S3 Storage Lens account snapshot.

Note

- You can't modify the dashboard scope of the **default dashboard** because it's linked to the **account snapshot**. However, you can upgrade the metrics selection from **free metrics** to the paid **advanced metrics and recommendations**.
- If you disable your default dashboard, your account snapshot is no longer updated. You can re-enable the **default-account-dashboard** to resume displaying metrics in the account snapshot.

Metrics export

An S3 Storage Lens *metrics export* is a file that contains all the metrics identified in your S3 Storage Lens configuration. This information is generated daily in CSV or Parquet format in an S3 bucket of your choice for further analysis. The S3 bucket for your metrics export must be in the same Region as your S3 Storage Lens configuration. You can generate an S3 Storage Lens metrics export from the S3 console by editing your dashboard configuration, or by using the AWS CLI and SDKs.

Home Region

The home Region is the AWS Region where all Amazon S3 Storage Lens metrics for a given dashboard or configuration's are stored. You must choose a home Region when you create your S3 Storage Lens dashboard or configuration. After a home Region is assigned, it can't be changed.

Note

Creating a home Region is supported the following Regions:

- US East (N. Virginia) – us-east-1
- US East (Ohio) – us-east-2
- US West (N. California) – us-west-1
- US West (Oregon) – us-west-2
- Asia Pacific (Mumbai) – ap-south-1
- Asia Pacific (Osaka) – ap-northeast-3
- Asia Pacific (Seoul) – ap-northeast-2
- Asia Pacific (Singapore) – ap-southeast-1
- Asia Pacific (Sydney) – ap-southeast-2
- Asia Pacific (Tokyo) – ap-northeast-1
- Canada (Central) – ca-central-1
- China (Beijing) – cn-north-1
- China (Ningxia) – cn-northwest-1
- Europe (Frankfurt) – eu-central-1
- Europe (Ireland) – eu-west-1
- Europe (London) – eu-west-2
- Europe (Paris) – eu-west-3
- Europe (Stockholm) – eu-north-1
- South America (São Paulo) – sa-east-1

Retention period

Amazon S3 Storage Lens metrics are retained so you can see historical trends and compare differences in your storage usage and activity over time. All S3 Storage Lens metrics are retained for a period of 15 months. A dashboard with only free metrics can display metrics that are up to 14 days old. A dashboard that has advanced metrics and recommendations enabled can display metrics that are up to 15 months old.

You can use Amazon S3 Storage Lens metrics for queries so that you can see historical trends and compare differences in your storage usage and activity over time. Metrics are available for a specific duration. The duration depends on your [metrics selection](#) and cannot be modified. Free metrics are available for queries for a 14-day period, and advanced metrics are available for queries for a 15-month period.

Metrics types

S3 Storage Lens offers two types of storage metrics: *usage* and *activity*.

- **Usage metrics**

S3 Storage Lens collects *usage metrics* for all dashboards and configurations. Usage metrics describe the size, quantity, and characteristics of your storage. This includes the total bytes stored, object count, and average object size. It also includes metrics that describe feature utilization, such as encrypted bytes, or delete market object counts. For more information about the usage metrics aggregated by S3 Storage Lens, see [Metrics glossary](#).

- **Activity metrics**

S3 Storage Lens aggregates *activity metrics* for all dashboards and configurations that have the *advanced metrics and recommendations metrics* type enabled. Activity metrics describe the details of how often your storage is requested. This includes the number of requests by type, upload and download bytes, and errors. For more information about the activity metrics that are aggregated by S3 Storage Lens, see [Metrics glossary](#).

Recommendations

S3 Storage Lens provides automated *recommendations* to help you optimize your storage. Recommendations are placed contextually alongside relevant metrics in the S3 Storage Lens dashboard. Historical data is not eligible for recommendations because recommendations are relevant to what is happening in the most recent period. Recommendations only appear when they are relevant.

S3 Storage Lens recommendations come in the following forms:

- **Suggestions**

Suggestions alert you to trends within your storage usage and activity that might indicate a storage cost optimization opportunity or data protection best practice. You can use the suggested topics in the *Amazon S3 User Guide* and the S3 Storage Lens dashboard to drill down for more details about the specific Regions, buckets, or prefixes to further assist you.

- **Call-outs**

Call-outs are recommendations that alert you to interesting anomalies within your storage usage and activity over a period that might need further attention or monitoring.

- **Outlier call-outs**

S3 Storage Lens provides call-outs for metrics that are *outliers*, based on your recent 30-day trend. The outlier is calculated using a standard score, also known as a *z-score*. In this score, the current

day's metric is subtracted from the average of the last 30 days for that metric, and then divided by the standard deviation for that metric over the last 30 days. The resulting score is usually between -3 and +3. This number represents the number of standard deviations that the current day's metric is from the mean.

S3 Storage Lens considers metrics with a score >2 or <-2 to be outliers because they are higher or lower than 95 percent of normally distributed data.

- **Significant change call-outs**

The *significant change call-out* applies to metrics that are expected to change less frequently. Therefore it is set to a higher sensitivity than the outlier calculation, which is typically in the range of +/- 20 percent versus the prior day, week, or month.

Addressing call-outs in your storage usage and activity – If you receive a significant change call-out, it's not necessarily a problem, and could be the result of an anticipated change in your storage. For example, you might have recently added a large number of new objects, deleted a large number of objects, or made similar planned changes.

If you see a significant change call-out on your dashboard, take note of it and determine whether it can be explained by recent circumstances. If not, use the S3 Storage Lens dashboard to drill down for more details to understand the specific Regions, buckets, or prefixes that are driving the fluctuation.

- **Reminders**

Reminders provide insights into how Amazon S3 works. They can help you learn more about ways to use S3 features to reduce storage costs or apply data protection best practices.

Metrics selection

S3 Storage Lens offers two metrics selections that you can choose for your dashboard and export: *free metrics* and *advanced metrics and recommendations*.

- **Free metrics**

S3 Storage Lens offers free metrics for all dashboards and configurations. Free metrics contain metrics that are relevant to your storage usage. This includes the number of buckets, the objects in your account, and what state they are in. All free metrics are collected daily. Data is available for queries for 14-days. For more information about what usage metrics are aggregated by S3 Storage Lens, see the [Metrics glossary](#).

- **Advanced metrics and recommendations**

S3 Storage Lens offers free metrics for all dashboards and configurations with the option to upgrade to *advanced metrics and recommendations*. Additional charges apply. For more information, see [Amazon S3 pricing](#). Advanced metrics contain all the usage metrics that are included in free metrics. This includes the number of buckets, the objects in your account, and what state they are in.

This metrics selection also provides recommendations to help you optimize your storage. Recommendations are placed contextually alongside relevant metrics in the dashboard.

Advanced metrics and recommendations also include the following features:

- **Activity metrics** - Generate additional metrics aggregated by bucket, such as requests, bytes uploaded/downloaded, and errors. Activity metrics data is relevant to your storage activity. This includes the number of requests, scans, and errors with respect to the configuration scope and what state they are in.
- **Amazon CloudWatch publishing** - Publish S3 Storage Lens usage and activity metrics to CloudWatch to create a unified view of your operational health in CloudWatch [dashboards](#). You can also use CloudWatch APIs and features like alarms and triggered actions, metric math, and anomaly

detection to monitor and take action on S3 Storage Lens metrics. For more information, see [Monitor S3 Storage Lens metrics in CloudWatch \(p. 1068\)](#).

- **Prefix aggregation** - Collect usage metrics at the prefix level. Prefix level metrics are not published to CloudWatch.

All advanced metrics are collected daily. Data is available for queries for 15 months. For more information about the storage metrics aggregated by S3 Storage Lens, see [Amazon S3 Storage Lens metrics glossary \(p. 1081\)](#).

Note

Recommendations are available only when you use the S3 Storage Lens dashboard on the Amazon S3 console, and not via the AWS CLI and SDKs.

S3 Storage Lens and AWS Organizations

AWS Organizations is an AWS service that helps you aggregate all your AWS accounts under one organization hierarchy. Amazon S3 Storage Lens works with AWS Organizations to provide a single view of object storage usage and activity across your Amazon S3 storage.

For more information, see [Using Amazon S3 Storage Lens with AWS Organizations \(p. 1059\)](#).

- **Trusted access**

Using your organization's management account, you must enable *trusted access* for S3 Storage Lens to aggregate storage metrics and usage data for all member accounts in your organization. You can then create dashboards or exports for your organization using your management account or by giving delegated administrator access to other accounts in your organization.

You can disable trusted access for S3 Storage Lens at any time, which stops S3 Storage Lens from aggregating metrics for your organization.

- **Delegated administrator**

You can create dashboards and metrics for S3 Storage Lens for your organization using your AWS Organizations management account, or by giving *delegated administrator* access to other accounts in your organization. You can deregister delegated administrators at any time, which prevents S3 Storage Lens from collecting data on an organization level.

For more information, see [Amazon S3 Storage Lens and AWS Organizations](#) in the *AWS Organizations User Guide*.

Amazon S3 Storage Lens service-linked roles

Along with AWS Organizations trusted access, Amazon S3 Storage Lens uses AWS Identity and Access Management (IAM) service-linked roles. A service-linked role is a unique type of IAM role that is linked directly to S3 Storage Lens. Service-linked roles are predefined by S3 Storage Lens and include all the permissions that it requires to collect daily storage usage and activity metrics from member accounts in your organization.

For more information, see [Using service-linked roles for Amazon S3 Storage Lens](#).

Using Amazon S3 Storage Lens with AWS Organizations

You can use Amazon S3 Storage Lens to collect storage metrics and usage data for all AWS accounts that are part of your AWS Organizations hierarchy. To do this, you must be using AWS Organizations, and you must enable S3 Storage Lens trusted access using your AWS Organizations management account.

After enabling trusted access, you can add delegated administrator access to accounts in your organization. These accounts can then create S3 Storage Lens configurations and dashboards that collect organization-wide storage metrics and user data.

For more information about enabling trusted access, see [Amazon S3 Storage Lens and AWS Organizations](#) in the *AWS Organizations User Guide*.

Topics

- [Enabling trusted access for S3 Storage Lens \(p. 1060\)](#)
- [Disabling trusted access for S3 Storage Lens \(p. 1060\)](#)
- [Registering a delegated administrator for S3 Storage Lens \(p. 1060\)](#)
- [Deregistering a delegated administrator for S3 Storage Lens \(p. 1061\)](#)

Enabling trusted access for S3 Storage Lens

By enabling trusted access, you allow Amazon S3 Storage Lens to have access to your AWS Organizations hierarchy, membership, and structure through the AWS Organizations APIs. S3 Storage Lens then becomes a trusted service for your entire organization's structure.

Whenever a dashboard configuration is created, S3 Storage Lens creates service-linked roles in your organization's management or delegated administrator accounts. The service-linked role grants S3 Storage Lens permissions to describe organizations, list accounts, verify a list of AWS service access for the organizations, and get delegated administrators for the organization. S3 Storage Lens can then ensure it has access to collect the cross-account storage usage and activity metrics for accounts in your organizations. For more information, see [Using service-linked roles for Amazon S3 Storage Lens](#).

After enabling trusted access, you can assign delegate administrator access to accounts in your organization. When an account is marked as a delegate administrator for a service, the account receives authorization to access all read-only organization APIs. This provides visibility to the members and structures of your organization so that they too can create S3 Storage Lens dashboards.

Note

Only the management account can enable trusted access for Amazon S3 Storage Lens.

Disabling trusted access for S3 Storage Lens

By disabling trusted access, you limit S3 Storage Lens to working only on an account level. In addition, each account holder can only see the S3 Storage Lens benefits limited to the scope of their account, and not their entire organization. Any dashboards requiring trusted access are no longer updated, but will retain their historic data per the period that [data is available for queries](#).

Removing an account as a delegated administrator will limit their S3 Storage Lens dashboard metrics access to only work on an account level. Any organizational dashboards that they created are no longer updated, but they will retain their historic data for the period that data is available for queries.

Note

- This action also automatically stops all organization-level dashboards from collecting and aggregating storage metrics.
- Your management and delegated administrator accounts will still be able to see the historic data for your existing organization-level dashboards during the period that data is available for queries.

Registering a delegated administrator for S3 Storage Lens

You can create organization-level dashboards using your organization's management account or delegated administrator accounts. Delegated administrator accounts allow other accounts besides your

management account to create organization-level dashboards. Only the management account of an organization can register and deregister other accounts as delegated administrators for the organization.

To register a delegated administrator using the Amazon S3 console, see [Registering delegated administrators for S3 Storage Lens \(p. 1096\)](#).

You can also register a delegated administrator using the AWS Organizations REST API, AWS CLI, or SDKs from the management account. For more information, see [RegisterDelegatedAdministrator](#) in the [AWS Organizations API Reference](#).

Note

Before you can designate a delegated administrator using the AWS Organizations REST API, AWS CLI, or SDKs, you must call the [EnableAWSOrganizationsAccess](#) operation.

Deregistering a delegated administrator for S3 Storage Lens

You can also de-register a delegated administrator account. Delegated administrator accounts allow other accounts besides your management account to create organization-level dashboards. Only the management account of an organization can de-register accounts as delegated administrators for the organization.

To de-register a delegated admin using the S3 console, see [Deregistering delegated administrators for S3 Storage Lens \(p. 1097\)](#).

You can also de-register a delegated administrator using the AWS Organizations REST API, AWS CLI, or SDKs from the management account. For more information, see [DeregisterDelegatedAdministrator](#) in the [AWS Organizations API Reference](#).

Note

- This action also automatically stops all organization-level dashboards created by that delegated administrator from aggregating new storage metrics.
- The delegate administrator accounts will still be able to see the historic data for those dashboards while data is available for queries.

Amazon S3 Storage Lens permissions

Amazon S3 Storage Lens requires new permissions in AWS Identity and Access Management (IAM) to authorize access to S3 Storage Lens actions. You can attach the policy to IAM users, groups, or roles to grant them permissions to enable or disable S3 Storage Lens, or to access any S3 Storage Lens dashboard or configuration.

The IAM user or role must belong to the account that created or owns the dashboard or configuration, unless your account is a member of AWS Organizations, and you were given access to create organization-level dashboards by your management account as a delegated administrator.

Note

- You can't use your account's root user credentials to view Amazon S3 Storage Lens dashboards. To access S3 Storage Lens dashboards, you must grant the requisite IAM permissions to a new or existing IAM user. Then, sign in with those user credentials to access S3 Storage Lens dashboards. For more information, see [AWS Identity and Access Management best practices](#).
- Using S3 Storage Lens on the Amazon S3 console can require multiple permissions. For example, to edit a dashboard on the console, you need the following permissions:
 - `s3>ListStorageLensConfigurations`

- `s3:GetStorageLensConfiguration`
- `s3:PutStorageLensConfiguration`

Topics

- [Setting account permissions to use S3 Storage Lens \(p. 1062\)](#)
- [Setting permissions to use S3 Storage Lens with AWS Organizations \(p. 1063\)](#)

Setting account permissions to use S3 Storage Lens

Amazon S3 Storage Lens related IAM permissions

Action	IAM permissions
Create or update an S3 Storage Lens dashboard in the Amazon S3 console.	<code>s3>ListStorageLensConfigurations</code> <code>s3:GetStorageLensConfiguration</code> <code>s3:GetStorageLensConfigurationTagging</code> <code>s3:PutStorageLensConfiguration</code> <code>s3:PutStorageLensConfigurationTagging</code>
Get tags of an S3 Storage Lens dashboard on the Amazon S3 console.	<code>s3>ListStorageLensConfigurations</code> <code>s3:GetStorageLensConfigurationTagging</code>
View an S3 Storage Lens dashboard on the Amazon S3 console.	<code>s3>ListStorageLensConfigurations</code> <code>s3:GetStorageLensConfiguration</code> <code>s3:GetStorageLensDashboard</code>
Delete an S3 Storage Lens dashboard on Amazon S3 console.	<code>s3>ListStorageLensConfigurations</code> <code>s3:GetStorageLensConfiguration</code> <code>s3>DeleteStorageLensConfiguration</code>
Create or update an S3 Storage Lens configuration in the AWS CLI or SDK.	<code>s3:PutStorageLensConfiguration</code> <code>s3:PutStorageLensConfigurationTagging</code>
Get tags of an S3 Storage Lens configuration in the AWS CLI or SDK.	<code>s3:GetStorageLensConfigurationTagging</code>
View an S3 Storage Lens configuration in the AWS CLI or SDK.	<code>s3:GetStorageLensConfiguration</code>
Delete an S3 Storage Lens configuration in AWS CLI or SDK.	<code>s3>DeleteStorageLensConfiguration</code>

Note

- You can use resource tags in an IAM policy to manage permissions.
- An IAM user/role with these permissions can see metrics from buckets and prefixes that they might not have direct permission to read or list objects from.

- For S3 Storage Lens configurations with *advanced metrics and recommendations* aggregated at the prefix-level, if the selected prefix matches object keys, it may show the object key as your prefix up to the delimiter and maximum depth selected.
- For metrics exports, which are stored in a bucket in your account, permissions are granted using the existing `s3:GetObject` permission in the IAM policy. Similarly, for an AWS Organizations entity, the organization management or delegated administrator account can use IAM policies to manage access permissions for organization-level dashboard and configurations.

Setting permissions to use S3 Storage Lens with AWS Organizations

You can use Amazon S3 Storage Lens to collect storage metrics and usage data for all accounts that are part of your AWS Organizations hierarchy. The following are the actions and permissions related to using S3 Storage Lens with Organizations.

AWS Organizations related IAM permissions for using S3 Storage Lens

Action	IAM Permissions
Enable trusted access for S3 Storage Lens for your organization.	<code>organizations:EnableAWSServiceAccess</code>
Disable trusted access S3 Storage Lens for your organization.	<code>organizations:DisableAWSServiceAccess</code>
Register a delegated administrator to create S3 Storage Lens dashboards or configurations for your organization.	<code>organizations:RegisterDelegatedAdministrator</code>
De-register a delegated administrator to create S3 Storage Lens dashboards or configurations for your organization.	<code>organizations:DeregisterDelegatedAdministrator</code>
Additional permissions to create S3 Storage Lens organization-wide configurations	<code>organizations:DescribeOrganization</code> <code>organizations>ListAccounts</code> <code>organizations>ListAWSServiceAccessForOrganization</code> <code>organizations>ListDelegatedAdministrators</code> <code>iam>CreateServiceLinkedRole</code>

Viewing storage usage and activity metrics with Amazon S3 Storage Lens

By default, all dashboards are configured with **free metrics**, which include **usage metrics** aggregated down to the bucket level and data is available for queries for 14 days. This means that you can see all the usage metrics that S3 Storage Lens aggregates, and your data will be available for queries for 14 days from the day it was aggregated.

Advanced metrics and recommendations include usage metrics with prefix-level aggregation, activity metrics aggregated by bucket, and contextual recommendations (available only in the dashboard).

[Activity metrics](#) are available for queries for 15 months. There are additional charges for using S3 Storage Lens with advanced metrics. For more information, see [Amazon S3 pricing](#).

Topics

- [Viewing S3 Storage Lens metrics on the dashboards \(p. 1064\)](#)
- [Viewing Amazon S3 Storage Lens metrics using a data export \(p. 1064\)](#)
- [Monitor S3 Storage Lens metrics in CloudWatch \(p. 1068\)](#)

Viewing S3 Storage Lens metrics on the dashboards

S3 Storage Lens provides you with a dashboard containing usage metrics at no additional cost. If you want to receive advanced metrics and recommendations, including usage and activity metrics, prefix aggregations, and contextual recommendations in the dashboard, you must select it from the dashboard configuration page on the Amazon S3 console.

The dashboard provides an interactive visualization for your storage usage and activity metrics. You can view organization-wide trends, or see more granular trends by AWS account, AWS Region, storage class, S3 bucket, or prefix.

If your account is a member of AWS Organizations, you can also see your storage usage and activity for your entire organization across member accounts. This information is available to you provided that S3 Storage Lens has been given trusted access to your organization, and you are an authorized management or delegated administrator account.

Use the interactive dashboard to explore your storage usage and activity trends and insights, and get contextual recommendations for best practices to optimize your storage. For more information, see [Understanding Amazon S3 Storage Lens \(p. 1054\)](#).

Amazon S3 preconfigures the S3 Storage Lens *default dashboard* to help you visualize summarized insights and trends of your entire account's aggregated storage usage and activity metrics(optional upgrade). You can't modify the default dashboard configuration scope, but the metrics selection can be upgraded from *free metrics* to the paid *advanced metrics and recommendations*. You can configure the optional metrics export, or even disable the dashboard. However, the default dashboard cannot be deleted.

In addition to the default dashboard that Amazon S3 creates, you can also create custom dashboards scoped to your own organization's accounts, Regions, buckets, and prefixes(account-level only). These custom dashboards can be edited, deleted, and disabled. Summary information from the **default dashboard** appears in the account snapshot section in the S3 console home (bucket lists) page.

Note

The Amazon S3 Storage Lens dashboard is only available from the Amazon S3 console. For more information, see [Viewing an Amazon S3 Storage Lens dashboard \(p. 1087\)](#).

Viewing Amazon S3 Storage Lens metrics using a data export

Amazon S3 Storage Lens metrics are generated daily in CSV or Apache Parquet-formatted metrics export files and placed in an S3 bucket in your account. From there, you can ingest the metrics export into the analytics tools of your choice, such as Amazon QuickSight and Amazon Athena, where you can analyze storage usage and activity trends.

Topics

- [Using an AWS KMS key to encrypt your metrics exports \(p. 1065\)](#)
- [What is an S3 Storage Lens export manifest? \(p. 1065\)](#)
- [Understanding the Amazon S3 Storage Lens export schema \(p. 1067\)](#)

Using an AWS KMS key to encrypt your metrics exports

To grant Amazon S3 Storage Lens permission to encrypt using a customer managed key, you must use a key policy. To update your key policy so that you can use an KMS key to encrypt your S3 Storage Lens metrics exports, follow these steps.

To grant permissions to encrypt using your KMS key

1. Sign into the AWS Management Console using the AWS account that owns the customer managed key.
2. Open the AWS KMS console at <https://console.aws.amazon.com/kms>.
3. To change the AWS Region, use the **Region selector** in the upper-right corner of the page.
4. In the navigation pane, choose **Customer managed keys**.
5. Under **Customer managed keys**, choose the key that you want to use to encrypt the metrics exports. AWS KMS keys are Region-specific and must be in the same Region as the metrics export destination S3 bucket.
6. Under **Key policy**, choose **Switch to policy view**.
7. To update the key policy, choose **Edit**.
8. Under **Edit key policy**, add the following key policy to the existing key policy.

```
{  
    "Sid": "Allow Amazon S3 Storage Lens use of the KMS key",  
    "Effect": "Allow",  
    "Principal": {  
        "Service": "storage-lens.s3.amazonaws.com"  
    },  
    "Action": [  
        "kms:GenerateDataKey"  
    ],  
    "Resource": "*",  
    "Condition": {  
        "StringEquals": {  
            "aws:SourceArn": "arn:aws:s3:us-east-1:source-account-id:storage-lens/your-  
dashboard-name",  
            "aws:SourceAccount": "source-account-id"  
        }  
    }  
}
```

9. Choose **Save changes**.

For more information about creating customer managed keys and using key policies, see the following topics in the *AWS Key Management Service Developer Guide*:

- [Getting started](#)
- [Using key policies in AWS KMS](#)

You can also use the AWS KMS PUT key policy ([PutKeyPolicy](#)) to copy the key policy to the customer managed keys that you want to use to encrypt the metrics exports using the REST API, AWS CLI, and SDKs.

What is an S3 Storage Lens export manifest?

Given the large amount of data aggregated, an S3 Storage Lens daily metrics export can be split into multiple files. The manifest file `manifest.json` describes where the metrics export files for that day are located. Whenever a new export is delivered, it is accompanied by a new manifest. Each manifest contained in the `manifest.json` file provides metadata and other basic information about the export.

The manifest information includes the following properties:

- `sourceAccountId` – The account ID of the configuration owner.
- `configId` – A unique identifier for the dashboard.
- `destinationBucket` – The destination bucket Amazon Resource Name (ARN) that the metrics export is placed in.
- `reportVersion` – The version of the export.
- `reportDate` – The date of the report.
- `reportFormat` – The format of the report.
- `reportSchema` – The schema of the report.
- `reportFiles` – The actual list of the export report files that are in the destination bucket.

The following is an example of a manifest in a `manifest.json` file for a CSV-formatted export.

```
{  
    "sourceAccountId": "123456789012",  
    "configId": "my-dashboard-configuration-id",  
    "destinationBucket": "arn:aws:s3:::destination-bucket",  
    "reportVersion": "V_1",  
    "reportDate": "2020-11-03",  
    "reportFormat": "CSV",  
  
    "reportSchema": "version_number,configuration_id,report_date,aws_account_number,aws_region,storage_class",  
    "reportFiles": [  
        {  
            "key": "DestinationPrefix/StorageLens/123456789012/my-dashboard-configuration-id/V_1/reports/dt=2020-11-03/a38f6bc4-2e3d-4355-ac8a-e2fdcf3de158.csv",  
            "size": 1603959,  
            "md5Checksum": "2177e775870def72b8d84febe1ad3574"  
        }  
    ]  
}
```

The following is an example of a manifest in a `manifest.json` file for a Parquet-formatted export.

```
{  
    "sourceAccountId": "123456789012",  
    "configId": "my-dashboard-configuration-id",  
    "destinationBucket": "arn:aws:s3:::destination-bucket",  
    "reportVersion": "V_1",  
    "reportDate": "2020-11-03",  
    "reportFormat": "Parquet",  
    "reportSchema": "message s3.storage.lens { required string version_number; required string configuration_id; required string report_date; required string aws_account_number; required string aws_region; required string storage_class; required string record_type; required string record_value; required string bucket_name; required string metric_name; required long metric_value; }",  
    "reportFiles": [  
        {  
            "key": "DestinationPrefix/StorageLens/123456789012/my-dashboard-configuration-id/V_1/reports/dt=2020-11-03/bd23de7c-b46a-4cf4-bcc5-b21aac5be0f5.par",  
            "size": 14714,  
            "md5Checksum": "b5c741ee0251cd99b90b3e8eff50b944"  
        }  
    ]  
}
```

You can configure your metrics export to be generated as part of your dashboard configuration in the Amazon S3 console or by using the Amazon S3 REST API, AWS CLI, and SDKs.

Understanding the Amazon S3 Storage Lens export schema

The following table contains the schema of your S3 Storage Lens metrics export.

Attribute name	Data type	Column name	Description
VersionNumber	String	version_number	The version of the S3 Storage Lens metrics being used.
ConfigurationId	String	configuration_id	The name of the configuration_id of your S3 Storage Lens configuration.
ReportDate	String	report_date	The date the metrics were tracked.
AwsAccountNumber	String	aws_account_number	Your AWS account number.
AwsRegion	String	aws_region	The AWS Region for which the metrics are being tracked.
StorageClass	String	storage_class	The storage class of the bucket in question.
RecordType	ENUM	record_type	The type of artifact that is being reported (ACCOUNT, BUCKET, or PREFIX).
RecordValue	String	record_value	The record value. This field is populated when the record_type is PREFIX. Note The record value is URL-encoded
BucketName	String	bucket_name	The name of the bucket that is being reported.
MetricName	String	metric_name	The name of the metric that is being reported.
MetricValue	Long	metric_value	The value of the metric that is being reported.

Example of an S3 Storage Lens metrics export

The following is an example of an S3 Storage Lens metrics export based on this schema.

version_r	configuration_id	report_date	aws_account_number	aws_region	storage_class	record_type	record_value	bucket_name	metric_name
V_1	sample-cmh-exclude	11/3/2020	546264889236	eu-west-1	STANDARD	ACCOUNT			StorageBytes
V_1	sample-cmh-exclude	11/3/2020	546264889236	eu-west-1	STANDARD	ACCOUNT			ObjectCount
V_1	sample-cmh-exclude	11/3/2020	546264889236	eu-west-1	STANDARD	ACCOUNT			ReplicatedObjects
V_1	sample-cmh-exclude	11/3/2020	546264889236	eu-west-1	STANDARD	ACCOUNT			ReplicatedObjects
V_1	sample-cmh-exclude	11/3/2020	546264889236	eu-west-1	STANDARD	ACCOUNT			EncryptedStorage
V_1	sample-cmh-exclude	11/3/2020	546264889236	eu-west-1	STANDARD	ACCOUNT			EncryptedObjects
V_1	sample-cmh-exclude	11/3/2020	546264889236	eu-west-1	STANDARD	ACCOUNT			DeleteMarkerCount
V_1	sample-cmh-exclude	11/3/2020	546264889236	eu-west-1	STANDARD	ACCOUNT			ObjectLockEnabled
V_1	sample-cmh-exclude	11/3/2020	546264889236	eu-west-1	STANDARD	ACCOUNT			ObjectLockEnabled
V_1	sample-cmh-exclude	11/3/2020	546264889236	eu-west-1	STANDARD	ACCOUNT			CurrentVersion
V_1	sample-cmh-exclude	11/3/2020	546264889236	eu-west-1	STANDARD	ACCOUNT			CurrentVersions
V_1	sample-cmh-exclude	11/3/2020	546264889236	eu-west-1	STANDARD	ACCOUNT			NonCurrent
V_1	sample-cmh-exclude	11/3/2020	546264889236	eu-west-1	STANDARD	ACCOUNT			NonCurrent
V_1	sample-cmh-exclude	11/3/2020	546264889236	eu-west-1	STANDARD	ACCOUNT			IncompleteMultipartUploads
V_1	sample-cmh-exclude	11/3/2020	546264889236	eu-west-1	STANDARD	ACCOUNT			IncompleteMultipartUploads
V_1	sample-cmh-exclude	11/3/2020	546264889236	us-west-1	STANDARD	PREFIX	AWSLogs%2F546264889236%2FCloudTrail%2Fus-cloudtrail-log-sfc	StorageBytes	AWSLogs%2F546264889236%2FCloudTrail%2Fus-cloudtrail-log-sfc
V_1	sample-cmh-exclude	11/3/2020	546264889236	us-west-1	STANDARD	PREFIX	AWSLogs%2F546264889236%2FCloudTrail%2Fus-cloudtrail-log-sfc	ObjectCount	AWSLogs%2F546264889236%2FCloudTrail%2Fus-cloudtrail-log-sfc
V_1	sample-cmh-exclude	11/3/2020	546264889236	us-west-1	STANDARD	PREFIX	AWSLogs%2F546264889236%2FCloudTrail%2Fus-cloudtrail-log-sfc	ReplicatedObjects	AWSLogs%2F546264889236%2FCloudTrail%2Fus-cloudtrail-log-sfc
V_1	sample-cmh-exclude	11/3/2020	546264889236	us-west-1	STANDARD	PREFIX	AWSLogs%2F546264889236%2FCloudTrail%2Fus-cloudtrail-log-sfc	ReplicatedObjects	AWSLogs%2F546264889236%2FCloudTrail%2Fus-cloudtrail-log-sfc
V_1	sample-cmh-exclude	11/3/2020	546264889236	us-west-1	STANDARD	PREFIX	AWSLogs%2F546264889236%2FCloudTrail%2Fus-cloudtrail-log-sfc	EncryptedStorage	AWSLogs%2F546264889236%2FCloudTrail%2Fus-cloudtrail-log-sfc
V_1	sample-cmh-exclude	11/3/2020	546264889236	us-west-1	STANDARD	PREFIX	AWSLogs%2F546264889236%2FCloudTrail%2Fus-cloudtrail-log-sfc	EncryptedObjects	AWSLogs%2F546264889236%2FCloudTrail%2Fus-cloudtrail-log-sfc
V_1	sample-cmh-exclude	11/3/2020	546264889236	us-west-1	STANDARD	PREFIX	AWSLogs%2F546264889236%2FCloudTrail%2Fus-cloudtrail-log-sfc	DeleteMarkerCount	AWSLogs%2F546264889236%2FCloudTrail%2Fus-cloudtrail-log-sfc
V_1	sample-cmh-exclude	11/3/2020	546264889236	us-west-1	STANDARD	PREFIX	AWSLogs%2F546264889236%2FCloudTrail%2Fus-cloudtrail-log-sfc	ObjectLockEnabled	AWSLogs%2F546264889236%2FCloudTrail%2Fus-cloudtrail-log-sfc
V_1	sample-cmh-exclude	11/3/2020	546264889236	us-west-1	STANDARD	PREFIX	AWSLogs%2F546264889236%2FCloudTrail%2Fus-cloudtrail-log-sfc	ObjectLockEnabled	AWSLogs%2F546264889236%2FCloudTrail%2Fus-cloudtrail-log-sfc
V_1	sample-cmh-exclude	11/3/2020	546264889236	us-west-1	STANDARD	PREFIX	AWSLogs%2F546264889236%2FCloudTrail%2Fus-cloudtrail-log-sfc	CurrentVersion	AWSLogs%2F546264889236%2FCloudTrail%2Fus-cloudtrail-log-sfc
V_1	sample-cmh-exclude	11/3/2020	546264889236	us-west-1	STANDARD	PREFIX	AWSLogs%2F546264889236%2FCloudTrail%2Fus-cloudtrail-log-sfc	CurrentVersions	AWSLogs%2F546264889236%2FCloudTrail%2Fus-cloudtrail-log-sfc
V_1	sample-cmh-exclude	11/3/2020	546264889236	us-west-1	STANDARD	PREFIX	AWSLogs%2F546264889236%2FCloudTrail%2Fus-cloudtrail-log-sfc	NonCurrent	AWSLogs%2F546264889236%2FCloudTrail%2Fus-cloudtrail-log-sfc
V_1	sample-cmh-exclude	11/3/2020	546264889236	us-west-1	STANDARD	PREFIX	AWSLogs%2F546264889236%2FCloudTrail%2Fus-cloudtrail-log-sfc	NonCurrent	AWSLogs%2F546264889236%2FCloudTrail%2Fus-cloudtrail-log-sfc
V_1	sample-cmh-exclude	11/3/2020	546264889236	us-west-1	STANDARD	PREFIX	AWSLogs%2F546264889236%2FCloudTrail%2Fus-cloudtrail-log-sfc	IncompleteMultipartUploads	AWSLogs%2F546264889236%2FCloudTrail%2Fus-cloudtrail-log-sfc
V_1	sample-cmh-exclude	11/3/2020	546264889236	us-west-1	STANDARD	PREFIX	AWSLogs%2F546264889236%2FCloudTrail%2Fus-cloudtrail-log-sfc	IncompleteMultipartUploads	AWSLogs%2F546264889236%2FCloudTrail%2Fus-cloudtrail-log-sfc

Monitor S3 Storage Lens metrics in CloudWatch

You can publish S3 Storage Lens usage and activity metrics to Amazon CloudWatch to create a unified view of your operational health in CloudWatch [dashboards](#). You can also use CloudWatch features, like alarms and triggered actions, metric math, and anomaly detection, to monitor and take action on S3 Storage Lens metrics. In addition, the CloudWatch APIs enable applications, including third-party providers, to access your S3 Storage Lens metrics. For more information about CloudWatch features, see the [Amazon CloudWatch User Guide](#).

You can enable the CloudWatch publishing option for new or existing dashboard configurations using the Amazon S3 console, Amazon S3 REST API, AWS CLI, and AWS SDKs. Dashboards that are upgraded to S3 Storage Lens *advanced metrics and recommendations* can use the CloudWatch publishing option. For S3 Storage Lens *advanced metrics and recommendations* pricing, see [Amazon S3 pricing](#). No additional CloudWatch metrics publishing charges apply; however, other CloudWatch charges such as dashboards, alarms, and API are applicable. For more information, see [Amazon CloudWatch Pricing](#).

S3 Storage Lens metrics are published to CloudWatch in the account that owns the S3 Storage Lens configuration. After you enable the CloudWatch publishing option within advanced metrics and recommendations, you can access organization, account, and bucket-level usage and activity metrics in CloudWatch. Prefix-level metrics are not available in CloudWatch.

Note

S3 Storage Lens metrics are daily metrics and are published to CloudWatch once per day. When you query S3 Storage Lens metrics in CloudWatch, the period for the query must be 1 day (86400 seconds). After your daily S3 Storage Lens metrics appear in your S3 Storage Lens dashboard in the Amazon S3 console, it can take a few hours for these same metrics to appear in CloudWatch. When you enable the CloudWatch publishing option for S3 Storage Lens metrics for the first time, it can take up to 24 hours for your metrics to publish to CloudWatch.

After you enable the CloudWatch publishing option, you can use the following CloudWatch features to monitor and analyze your S3 Storage Lens data:

- [Dashboards \(p. 1077\)](#) – Use CloudWatch dashboards to create customized S3 Storage Lens dashboards. Share your CloudWatch dashboard with people who don't have direct access to your AWS account, across teams, with stakeholders, and with people external to your organizations.

- [Alarms and triggered actions \(p. 1077\)](#) – Configure alarms that watch metrics and take action when a threshold is breached. For example, you can configure an alarm that sends an Amazon SNS notification when Incomplete Multipart Upload Bytes exceeds 1 GB for three consecutive days.
- [Anomaly detection \(p. 1077\)](#) – Enable anomaly detection to continuously analyze metrics, determine normal baselines, and surface anomalies. You can create an anomaly detection alarm based on the expected value of a metric. For example, you can monitor anomalies for Object Lock Enabled Bytes to detect unauthorized removal of Object Lock settings.
- [Metric math \(p. 1078\)](#) – You can also use metric math to query multiple S3 Storage Lens metrics and use math expressions to create new time series based on these metrics. For example, you can create a new metric to get the average object size by dividing StorageBytes by ObjectCount.

For more information about the CloudWatch publishing option for S3 Storage Lens metrics, see the following topics.

Topics

- [S3 Storage Lens metrics and dimensions \(p. 1069\)](#)
- [Enabling CloudWatch publishing for S3 Storage Lens \(p. 1071\)](#)
- [Working with S3 Storage Lens metrics in CloudWatch \(p. 1076\)](#)

S3 Storage Lens metrics and dimensions

To send S3 Storage Lens metrics to CloudWatch, you must enable the CloudWatch publishing option within S3 Storage Lens *advanced metrics and recommendations*. Once enabled, you can use CloudWatch **dashboards** to monitor S3 Storage Lens metrics alongside other application metrics and create a unified view of your operational health. You can use dimensions to filter your S3 Storage Lens metrics in CloudWatch by organization, account, bucket, storage class, Region, and metrics configuration ID.

S3 Storage Lens metrics are published to CloudWatch in the account that owns the S3 Storage Lens configuration. After you enable the CloudWatch publishing option within advanced metrics and recommendations, you can access organization, account, and bucket-level usage and activity metrics in CloudWatch. Prefix-level metrics are not available in CloudWatch.

Note

S3 Storage Lens metrics are daily metrics and are published to CloudWatch once per day. When you query S3 Storage Lens metrics in CloudWatch, the period for the query must be 1 day (86400 seconds). After your daily S3 Storage Lens metrics appear in your S3 Storage Lens dashboard in the Amazon S3 console, it can take a few hours for these same metrics to appear in CloudWatch. When you enable the CloudWatch publishing option for S3 Storage Lens metrics for the first time, it can take up to 24 hours for your metrics to publish to CloudWatch.

For more information about S3 Storage Lens metrics and dimensions in CloudWatch, see the following topics.

Topics

- [Metrics \(p. 1069\)](#)
- [Dimensions \(p. 1070\)](#)

Metrics

S3 Storage Lens usage and activity metrics are available as metrics within CloudWatch. S3 Storage Lens metrics are published to the AWS/S3/Storage-Lens namespace. This namespace is only for S3 Storage Lens metrics. Amazon S3 bucket, request, and replication metrics are published to the AWS/S3 namespace.

S3 Storage Lens metrics are published to CloudWatch in the account that owns the S3 Storage Lens configuration. After you enable the CloudWatch publishing option within advanced metrics and recommendations, you can access organization, account, and bucket-level usage and activity metrics in CloudWatch. Prefix-level metrics are not available in CloudWatch.

In S3 Storage Lens, metrics are aggregated and stored only in the designated home Region. S3 Storage Lens metrics are also published to CloudWatch in the home Region that you specify in the S3 Storage Lens configuration.

For a complete list of S3 Storage Lens metrics, including a list of those metrics available in CloudWatch, see [Amazon S3 Storage Lens metrics glossary \(p. 1081\)](#).

Note

The valid statistic for S3 Storage Lens metrics in CloudWatch is Average. For more information about statistics in CloudWatch, see [CloudWatch statistics definitions](#) in the *Amazon CloudWatch User Guide*.

[Granularity of S3 Storage Lens metrics in CloudWatch](#)

S3 Storage Lens offers metrics at organization, account, bucket, and prefix granularity. S3 Storage Lens publishes organization, account, and bucket-level S3 Storage Lens metrics to CloudWatch. Prefix-level S3 Storage Lens metrics are not available in CloudWatch.

For more information about the granularity of S3 Storage Lens metrics available in CloudWatch, see the following list:

- **Organization** – Metrics aggregated across the member accounts in your organization. S3 Storage Lens publishes metrics for member accounts to CloudWatch in the management account.
- **Organization and account** – Metrics for the member accounts in your organization.
- **Organization and bucket** – Metrics for Amazon S3 buckets in the member accounts of your organization.
- **Account (Non-organization level)** – Metrics aggregated across the buckets in your account.
- **Bucket (Non-organization level)** – Metrics for a specific bucket. In CloudWatch, S3 Storage Lens publishes these metrics to the AWS account that created the S3 Storage Lens configuration. S3 Storage Lens publishes these metrics only for non-organization configurations.

[Dimensions](#)

When S3 Storage Lens sends data to CloudWatch, it attaches dimensions to each metric. Dimensions are categories that describe the characteristics of metrics. You can use dimensions to filter the results that CloudWatch returns.

For example, all S3 Storage Lens metrics in CloudWatch have the `configuration_id` dimension. You can use this dimension to differentiate between metrics associated with a specific S3 Storage Lens configuration. The `organization_id` identifies organization-level metrics. For more information about dimensions in CloudWatch, see [Dimensions](#) in the *CloudWatch User Guide*.

Different dimensions are available for S3 Storage Lens metrics depending on the granularity of metrics. For example, you can use the `organization_id` dimension to filter organization-level metrics by the AWS Organizations ID. However, you can't use this dimension for bucket and account-level metrics. For more information, see [Filtering metrics using dimensions \(p. 1077\)](#).

To see which dimensions are available for your S3 Storage Lens configuration, see the following table.

Dimension	Description	Bucket	Organization	Account	BUCKET ORGANIZATION
configuration_id	Dashboard name for S3 Storage Lens configuration reported in the metrics	✓	✓	✓	✓
metrics_version	Version of the S3 Storage Lens metrics. The metrics version has a fixed value of 1.0.	✓	✓	✓	✓
organization_id	AWS Organizations ID for the metrics	✗	✗	✓	✓
aws_account_number	AWS account associated with the metrics	✓	✓	✗	✓
aws_region	AWS Region for the metrics	✓	✓	✓	✓
bucket_name	Name of the S3 bucket reported in the metrics	✓	✗	✗	✓
storage_class	Storage class for bucket reported in the metrics	✓	✓	✓	✓
record_type	Granularity of metrics: ORGANIZATION, ACCOUNT, BUCKET	✓	✓	✓	✓

Enabling CloudWatch publishing for S3 Storage Lens

You can publish S3 Storage Lens usage and activity metrics to Amazon CloudWatch to create a unified view of your operational health in CloudWatch dashboards. You can also use CloudWatch features, like alarms and triggered actions, metric math, and anomaly detection, to monitor and take action on S3 Storage Lens metrics. In addition, the CloudWatch APIs enable applications, including third-party providers, to access your S3 Storage Lens metrics. For more information about CloudWatch features, see the [Amazon CloudWatch User Guide](#).

S3 Storage Lens metrics are published to CloudWatch in the account that owns the S3 Storage Lens configuration. After you enable the CloudWatch publishing option within advanced metrics and recommendations, you can access organization, account, and bucket-level usage and activity metrics in CloudWatch. Prefix-level metrics are not available in CloudWatch.

You can enable CloudWatch support for new or existing dashboard configurations using the S3 console, Amazon S3 REST APIs, AWS CLI, and AWS SDKs. The CloudWatch publishing option is available for dashboards upgraded to S3 Storage Lens *advanced metrics and recommendations*. For S3 Storage Lens *advanced metrics and recommendations* pricing, see [Amazon S3 pricing](#). No additional CloudWatch metrics publishing charges apply; however, other CloudWatch charges such as dashboards, alarms, and API are applicable.

To enable the CloudWatch publishing option for S3 Storage Lens metrics, see the following topics.

Note

S3 Storage Lens metrics are daily metrics and are published to CloudWatch once per day. When you query S3 Storage Lens metrics in CloudWatch, the period for the query must be 1 day (86400 seconds). After your daily S3 Storage Lens metrics appear in your S3 Storage Lens dashboard in the Amazon S3 console, it can take a few hours for these same metrics to appear in CloudWatch. When you enable the CloudWatch publishing option for S3 Storage Lens metrics for the first time, it can take up to 24 hours for your metrics to publish to CloudWatch. Currently, S3 Storage Lens metrics cannot be consumed via CloudWatch streams.

Using the S3 console

When you update a S3 Storage Lens dashboard, you can't change the dashboard name or home Region. You also can't change the scope of the default dashboard, which is scoped to your entire account's storage.

To update S3 Storage Lens dashboard to enable CloudWatch publishing

1. Sign in to the AWS Management Console and open the Amazon S3 console at <https://console.aws.amazon.com/s3/>.
2. In the navigation pane, choose **S3 Storage Lens**.
3. Choose the dashboard that you want to edit, and then choose **Edit**.
4. Under **Metrics selection**, choose **Advanced metrics and recommendations**.

Advanced metrics and recommendations are available for an additional charge. Advanced metrics and recommendations include a 15-month period for data queries, usage metrics aggregated at the prefix level, activity metrics aggregated by bucket, the CloudWatch publishing option, and contextual recommendations that help you optimize storage costs and apply data protection best practices. For more information, see [Amazon S3 pricing](#).

5. Under **Select Advanced metrics and recommendations features**, select **CloudWatch publishing**.

Important

If your configuration enables prefix aggregation for usage metrics, prefix-level metrics will not be published to CloudWatch. Only bucket, account, and organization-level S3 Storage Lens metrics are published to CloudWatch.

6. Choose **Create dashboard**.

To create a new S3 Storage Lens dashboard that enables CloudWatch support

1. Sign in to the AWS Management Console and open the Amazon S3 console at <https://console.aws.amazon.com/s3/>.
2. In the navigation pane, choose **Storage Lens**.
3. Choose **Create dashboard**.
4. Under **General**, define configuration options:

- a. In **Dashboard name**, enter your dashboard name.

Dashboard names must be fewer than 65 characters and must not contain special characters or spaces. You can't change the dashboard name after you create your dashboard.

- b. Choose the **Home Region** for your dashboard.

Metrics for all Regions included in this dashboard scope are stored centrally in the designated home Region. In CloudWatch, S3 Storage Lens metrics are also available in the home Region. You can't change the home Region after you create your dashboard.

5. (Optional) To add tags, choose **Add tag** and enter the tag **Key** and **Value**.

Note

You can add up to 50 tags to your dashboard configuration.

6. Define the scope for your configuration:

- a. If you're creating an organization-level configuration, choose the accounts to include in the configuration: **Include all accounts in your configuration** or **Limit the scope to your signed-in account**.

Note

When you create an organization-level configuration that includes all accounts, you can only include or exclude Regions and not buckets.

- b. Choose Regions and buckets that you want S3 Storage Lens to include in the dashboard configuration:
 - To include all Regions, choose **Include Regions and buckets**.
 - To include specific Regions, clear **Include all Regions**. Under **Choose Regions to include**, choose the Regions that you want S3 Storage Lens to include in the dashboard.
 - To include specific buckets, clear **Include all buckets**. Under **Choose buckets to include**, choose the buckets that you want S3 Storage Lens to include in the dashboard.

Note

You can choose up to 50 buckets.

7. In the **Metrics selection**, choose **Advanced metrics and recommendations**.

For more information about advanced metrics and recommendations pricing, see [Amazon S3 pricing](#).

8. Under **Select advanced metrics and recommendations features**, select the options that apply:

- **Activity metrics**
- **CloudWatch publishing**

Important

If you enable prefix aggregation for your S3 Storage Lens configuration, prefix-level metrics will not be published to CloudWatch. Only bucket, account, and organization-level S3 Storage Lens metrics are published to CloudWatch

- **Prefix aggregation**

9. (Optional) Configure metrics export.

For more information about how to configure a metrics export, see step [Creating an Amazon S3 Storage Lens dashboard \(p. 1089\)](#).

10. Choose **Create dashboard**.

Using the AWS CLI

The following AWS CLI example enables the CloudWatch publishing option using a S3 Storage Lens organization-level *advanced metrics and recommendations* configuration.

```
aws s3control put-storage-lens-configuration --account-id=EXAMPLE-AWS-ACCOUNT-ID --  
config-id=your-configuration-id --region=us-east-1 --storage-lens-configuration=file://./  
config.json  
  
config.json  
{  
    "Id": "SampleS3StorageLensConfiguration", //Use this property to identify S3 S3 Storage  
    //Lens configuration.  
    "AwsOrg": { //Use this property when enabling S3 S3 Storage Lens for AWS Organizations  
        "Arn": "arn:aws:organizations::2222222222:organization/o-abcdefg"  
    },  
    "AccountLevel": {  
        "ActivityMetrics": {  
            "IsEnabled": true  
        },  
        "BucketLevel": {  
            "ActivityMetrics": { //Enables Activity Metrics  
                "IsEnabled": true  
            },  
            "PrefixLevel": { //Enables Prefix Level Metrics
```

```

        "StorageMetrics": {
            "IsEnabled": true,
            "SelectionCriteria": {
                "MaxDepth": 5,
                "MinStorageBytesPercentage": 1.25,
                "Delimiter": "/"
            }
        }
    },
    "Exclude": { //Replace with include if you prefer to include Regions.
        "Regions": [
            "eu-west-1"
        ],
        "Buckets": [ //This attribute is not supported for organization-level
configurations.
            "arn:aws:s3:::source_bucket1"
        ]
    },
    "IsEnabled": true, //Whether the configuration is enabled
    "DataExport": {
        "CloudWatchMetrics": { //Enables publishing Lens metrics to CloudWatch
            "IsEnabled": true
        },
        "S3BucketDestination": { //Enable export of metrics report to na S3 bucket
            "OutputSchemaVersion": "V_1",
            "Format": "CSV", //You can add "Parquet" if you prefer.
            "AccountId": "ExampleAWSAccountNo8",
            "Arn": "arn:aws:s3:::destination-bucket-name", //The destination bucket for
your metrics export must be in the same Region as your S3 S3 Storage Lens configuration.
            "Prefix": "prefix-for-your-export-destination",
            "Encryption": {
                "SSES3": {}
            }
        }
    }
}

```

Using the AWS SDK for Java

```

package aws.example.s3control;

import com.amazonaws.AmazonServiceException;
import com.amazonaws.SdkClientException;
import com.amazonaws.auth.profile.ProfileCredentialsProvider;
import com.amazonaws.services.s3control.AWSS3Control;
import com.amazonaws.services.s3control.AWSS3ControlClient;
import com.amazonaws.services.s3control.model.AccountLevel;
import com.amazonaws.services.s3control.model.ActivityMetrics;
import com.amazonaws.services.s3control.model.BucketLevel;
import com.amazonaws.services.s3control.model.CloudWatchMetrics;
import com.amazonaws.services.s3control.model.Format;
import com.amazonaws.services.s3control.model.Include;
import com.amazonaws.services.s3control.model.OutputSchemaVersion;
import com.amazonaws.services.s3control.model.PrefixLevel;
import com.amazonaws.services.s3control.model.PrefixLevelStorageMetrics;
import com.amazonaws.services.s3control.model.PutStorageLensConfigurationRequest;
import com.amazonaws.services.s3control.model.S3BucketDestination;
import com.amazonaws.services.s3control.model.SSES3;
import com.amazonaws.services.s3control.model.SelectionCriteria;
import com.amazonaws.services.s3control.model.StorageLensAwsOrg;
import com.amazonaws.services.s3control.model.StorageLensConfiguration;
import com.amazonaws.services.s3control.model.StorageLensDataExport;

```

```
import com.amazonaws.services.s3control.model.StorageLensDataExportEncryption;
import com.amazonaws.services.s3control.model.StorageLensTag;

import java.util.Arrays;
import java.util.List;

import static com.amazonaws.regions.Regions.US_WEST_2;

public class CreateAndUpdateDashboard {

    public static void main(String[] args) {
        String configurationId = "ConfigurationId";
        String sourceAccountId = "Source Account ID";
        String exportAccountId = "Destination Account ID";
        String exportBucketArn = "arn:aws:s3:::destBucketName"; // The destination
bucket for your metrics export must be in the same Region as your S3 S3 Storage Lens
configuration.
        String awsOrgARN = "arn:aws:organizations::222222222222:organization/o-abcdefg";
        Format exportFormat = Format.CSV;

        try {
            SelectionCriteria selectionCriteria = new SelectionCriteria()
                .withDelimiter("/")
                .withMaxDepth(5)
                .withMinStorageBytesPercentage(10.0);
            PrefixLevelStorageMetrics prefixStorageMetrics = new
PrefixLevelStorageMetrics()
                .withEnabled(true)
                .withSelectionCriteria(selectionCriteria);
            BucketLevel bucketLevel = new BucketLevel()
                .withActivityMetrics(new ActivityMetrics().withEnabled(true))
                .withPrefixLevel(new
PrefixLevel().withStorageMetrics(prefixStorageMetrics));
            AccountLevel accountLevel = new AccountLevel()
                .withActivityMetrics(new ActivityMetrics().withEnabled(true))
                .withBucketLevel(bucketLevel);

            Include include = new Include()
                .withBuckets(Arrays.asList("arn:aws:s3:::bucketName"))
                .withRegions(Arrays.asList("us-west-2"));

            StorageLensDataExportEncryption exportEncryption = new
StorageLensDataExportEncryption()
                .withSSES3(new SSES3());
            S3BucketDestination s3BucketDestination = new S3BucketDestination()
                .withAccountId(exportAccountId)
                .withArn(exportBucketArn)
                .withEncryption(exportEncryption)
                .withFormat(exportFormat)
                .withOutputSchemaVersion(OutputSchemaVersion.V_1)
                .withPrefix("Prefix");
            CloudWatchMetrics cloudWatchMetrics = new CloudWatchMetrics()
                .withEnabled(true);
            StorageLensDataExport dataExport = new StorageLensDataExport()
                .withCloudWatchMetrics(cloudWatchMetrics)
                .withS3BucketDestination(s3BucketDestination);

            StorageLensAwsOrg awsOrg = new StorageLensAwsOrg()
                .withArn(awsOrgARN);

            StorageLensConfiguration configuration = new StorageLensConfiguration()
                .withId(configurationId)
                .withAccountLevel(accountLevel)
                .withInclude(include)
                .withDataExport(dataExport)
                .withAwsOrg(awsOrg)
        }
    }
}
```

```
        .withIsEnabled(true);

    List<StorageLensTag> tags = Arrays.asList(
        new StorageLensTag().withKey("key-1").withValue("value-1"),
        new StorageLensTag().withKey("key-2").withValue("value-2")
    );

    AWSS3Control s3ControlClient = AWSS3ControlClient.builder()
        .withCredentials(new ProfileCredentialsProvider())
        .withRegion(US_WEST_2)
        .build();

    s3ControlClient.putStorageLensConfiguration(new
PutStorageLensConfigurationRequest()
    .withAccountId(sourceAccountId)
    .withConfigId(configurationId)
    .withStorageLensConfiguration(configuration)
    .withTags(tags)
);
} catch (AmazonServiceException e) {
    // The call was transmitted successfully, but Amazon S3 couldn't process
    // it and returned an error response.
    e.printStackTrace();
} catch (SdkClientException e) {
    // Amazon S3 couldn't be contacted for a response, or the client
    // couldn't parse the response from Amazon S3.
    e.printStackTrace();
}
}
}
```

Using the REST API

To enable the CloudWatch publishing option using the Amazon S3 REST API, you can use [PutStorageLensConfiguration](#).

Next steps

After you enable the CloudWatch publishing option, you can access your S3 Storage Lens metrics in CloudWatch. You also can leverage CloudWatch features to monitor and analyze your S3 Storage Lens data in CloudWatch. For more information, see the following topics:

- [S3 Storage Lens metrics and dimensions \(p. 1069\)](#)
- [Working with S3 Storage Lens metrics in CloudWatch \(p. 1076\)](#)

Working with S3 Storage Lens metrics in CloudWatch

You can publish S3 Storage Lens usage and activity metrics to Amazon CloudWatch to create a unified view of your operational health in CloudWatch [dashboards](#). You can also use CloudWatch features, like alarms and triggered actions, metric math, and anomaly detection, to monitor and take action on S3 Storage Lens metrics. In addition, the CloudWatch APIs enable applications, including third-party providers, to access your S3 Storage Lens metrics. For more information about CloudWatch features, see the [Amazon CloudWatch User Guide](#).

You can enable the CloudWatch publishing option for new or existing dashboard configurations using the Amazon S3 console, Amazon S3 REST APIs, AWS CLI, and AWS SDKs. The CloudWatch publishing option is available for dashboards upgraded to S3 Storage Lens *advanced metrics and recommendations*. For S3 Storage Lens *advanced metrics and recommendations* pricing, see [Amazon S3 pricing](#). No additional CloudWatch metrics publishing charges apply; however, other CloudWatch charges such as dashboards, alarms, and API are applicable. For more information, see [Amazon CloudWatch Pricing](#).

S3 Storage Lens metrics are published to CloudWatch in the account that owns the S3 Storage Lens configuration. After you enable the CloudWatch publishing option within advanced metrics and recommendations, you can access organization, account, and bucket-level usage and activity metrics in CloudWatch. Prefix-level metrics are not available in CloudWatch.

Note

S3 Storage Lens metrics are daily metrics and are published to CloudWatch once per day. When you query S3 Storage Lens metrics in CloudWatch, the period for the query must be 1 day (86400 seconds). After your daily S3 Storage Lens metrics appear in your S3 Storage Lens dashboard in the Amazon S3 console, it can take a few hours for these same metrics to appear in CloudWatch. When you enable the CloudWatch publishing option for S3 Storage Lens metrics for the first time, it can take up to 24 hours for your metrics to publish to CloudWatch. Currently, S3 Storage Lens metrics cannot be consumed via CloudWatch streams.

For more information about working with S3 Storage Lens metrics in CloudWatch, see the following topics.

Topics

- [Working with CloudWatch dashboards \(p. 1077\)](#)
- [Setting alarms, triggering actions, and using anomaly detection \(p. 1077\)](#)
- [Filtering metrics using dimensions \(p. 1077\)](#)
- [Calculating new metrics with metric math \(p. 1078\)](#)
- [Using search expressions in graphs \(p. 1078\)](#)

[Working with CloudWatch dashboards](#)

You can use CloudWatch dashboards to monitor S3 Storage Lens metrics alongside other application metrics and create a unified view of your operational health. Dashboards are customizable home pages in the CloudWatch console that you can use to monitor your resources in a single view.

CloudWatch has broad permissions control that doesn't support limiting access to a specific set of metrics or dimensions. Users in your account or organization who have access to CloudWatch will have access to metrics for all S3 Storage Lens configurations where the CloudWatch support option is enabled. You can't manage permissions for specific dashboards as you can in S3 Storage Lens. For more information about CloudWatch permissions, see [Managing access permissions to your CloudWatch resources](#) in the *Amazon CloudWatch User Guide*.

For more information about using CloudWatch dashboards and configuring permissions, see [Using Amazon CloudWatch dashboards](#) and [Sharing CloudWatch dashboards](#) in the *Amazon CloudWatch User Guide*.

[Setting alarms, triggering actions, and using anomaly detection](#)

You can configure CloudWatch alarms that watch S3 Storage Lens metrics in CloudWatch and take action when a threshold is breached. For example, you can configure an alarm that sends an Amazon SNS notification when Incomplete Multipart Upload Bytes exceeds 1 GB for three consecutive days.

You can also enable anomaly detection to continuously analyze your S3 Storage Lens metrics, determine normal baselines, and surface anomalies. You can create an anomaly detection alarm based on a metric's expected value. For example, you can monitor anomalies for Object Lock Enabled Bytes to detect unauthorized removal of Object Lock settings.

For more information and examples, see [Using Amazon CloudWatch alarms](#) and [Creating an alarm from a metric on a graph](#) in the *Amazon CloudWatch User Guide*.

[Filtering metrics using dimensions](#)

You can use dimensions to filter S3 Storage Lens metrics in the CloudWatch console. For example, you can filter by `configuration_id`, `aws_account_number`, `aws_region`, `bucket_name`, and more.

S3 Storage Lens supports multiple dashboard configurations per account. This means that different configurations can include the same bucket. When these metrics are published to CloudWatch, the bucket will have duplicate metrics within CloudWatch. To view metrics only for a specific S3 Storage Lens configuration in CloudWatch, you can use the `configuration_id` dimension. When you filter by `configuration_ID`, you only see metrics associated with the configuration that you identify.

For more information about filtering by configuration ID, see the following procedure, or see [Searching for available metrics](#) in the *Amazon CloudWatch User Guide*.

Calculating new metrics with metric math

You can use metric math to query multiple S3 Storage Lens metrics and use math expressions to create new time series based on these metrics. For example, you can create a new metric for unencrypted objects by subtracting Encrypted Objects from Object Count. You can also create a metric to get the average object size by dividing StorageBytes by ObjectCount or the number bytes accessed on one day by dividing BytesDownloaded by StorageBytes.

For more information, see [Using metric math](#) in the *Amazon CloudWatch User Guide*.

Using search expressions in graphs

With S3 Storage Lens metrics, you can create a search expression for all metrics called `IncompleteMultipartUploadStorageBytes` and add `SUM` to the expression. With this search expression, you can see your total Incomplete MPU bytes across all dimensions of your storage in a single metric.

This example shows the syntax that you would use to create a search expression for all metrics called `IncompleteMultipartUploadStorageBytes`.

```
SUM(SEARCH(''{AWS/S3/Storage-
Lens,aws_account_number,aws_region,configuration_id,metrics_version,record_type,storage_class}
MetricName="IncompleteMultipartUploadStorageBytes'', 'Average',86400))
```

For more information about this syntax, see [CloudWatch search expression syntax](#) in the *Amazon CloudWatch User Guide*. To create a CloudWatch graph with a search expression, see [Creating a CloudWatch graph with a search expression](#) in the *Amazon CloudWatch User Guide*.

Using Amazon S3 Storage Lens to optimize your storage costs

Amazon S3 Storage Lens aggregates your usage and activity metrics and displays that information in the account snapshot on the Amazon S3 console **Buckets** (home) page. You can use the S3 Storage Lens dashboard to visualize insights and trends, flag outliers, and receive recommendations to optimize storage costs and apply data protection best practices.

The following use cases provide strategies for using the S3 Storage Lens dashboard to optimize your storage more effectively.

Topics

- [Identify your largest S3 buckets \(p. 1079\)](#)
- [Locate incomplete multipart uploads \(p. 1079\)](#)
- [Reduce the number of noncurrent versions retained \(p. 1080\)](#)
- [Uncover cold Amazon S3 buckets \(p. 1080\)](#)

Identify your largest S3 buckets

You pay for storing objects in S3 buckets. The rate you're charged depends on your objects' sizes, how long you store the objects, and their storage classes. With Amazon S3 Storage Lens, you get a centralized view of all the buckets in your account. To see all the buckets in all of your organization's accounts, you can configure an AWS Organizations-level S3 Storage Lens dashboard. From this dashboard view, you can identify your largest buckets.

To identify your largest buckets

1. Sign in to the AWS Management Console and open the Amazon S3 console at <https://console.aws.amazon.com/s3/>.
2. In the navigation pane, choose **Storage Lens, Dashboards**.
3. In the **Dashboards** list, choose the dashboard that you want to view.
4. In the upper-right corner, you see the latest date that S3 Storage Lens has collected storage metrics for. Your dashboard always loads for the latest date for which metrics are available.
 - To adjust the scope of the dashboard data that you are viewing, choose **Filters** to apply temporary filters.
 - To remove all filters, choose **Reset**, and then choose **Apply**.
5. On the **Overview** tab of your dashboard, scroll down to the **Top N overview for date** section to see a ranking of your largest buckets by the **Total storage** metric for a selected date range.

You can toggle the sort order to show the smallest buckets and adjust the metric to rank your buckets by any of the more than 30 metrics available. This view also shows the percentage change from the prior day or week, as well as a spark-line to visualize your 14-day trend (or 30-day trend if you've upgraded to **advanced metrics and recommendations**).

6. For more detailed insights about your buckets, choose the **Buckets** tab of this dashboard. On the **Buckets** tab, you can see details such as the recent growth rate, the average object size, the largest prefixes, and the number of objects.
7. For your largest buckets, you can then navigate to each bucket within the S3 console to understand its objects and associated workload or to identify internal owners of the bucket. From the bucket owners, you can find out whether this growth is expected, or if this growth needs further monitoring and control.

Locate incomplete multipart uploads

You can use the multipart upload feature to upload very large objects (up to 5 TB) as a set of parts for improved throughput and quicker recovery from network issues. In cases where the multipart upload process doesn't finish, the incomplete parts remain in the bucket (in an unusable state) and incur storage costs until the upload process is finished, or until the incomplete parts are removed. For more information, see [Uploading and copying objects using multipart upload \(p. 167\)](#).

With S3 Storage Lens, you can identify the number of incomplete multipart upload bytes in your account or across your entire organization.

To identify incomplete multipart upload bytes

1. Sign in to the AWS Management Console and open the Amazon S3 console at <https://console.aws.amazon.com/s3/>.
2. In the navigation pane, choose **Storage Lens, Dashboards**.
3. In the **Dashboards** list, choose the dashboard that you want to view.
4. At the top of the **Overview** tab in the **Snapshot** section, choose **Cost efficiency** to see the **% incomplete MPU bytes** metric.

You can also select **Incomplete multipart upload bytes** as a metric in any chart in the S3 Storage Lens dashboard. You can then further assess the impact of incomplete multipart upload bytes on your storage, including their contribution to overall growth trends, or you can identify specific buckets that are accumulating incomplete multipart uploads.

To automatically manage incomplete multipart uploads, [create a lifecycle policy to expire incomplete multipart upload bytes](#) from the bucket after a specified number of days.

Reduce the number of noncurrent versions retained

When enabled, the S3 Versioning feature retains multiple versions of the same object that can be used to quickly recover data if an object is accidentally deleted or overwritten. S3 Versioning can have storage cost implications if a large number of previous noncurrent versions have accumulated. For more information, see [Using versioning in S3 buckets \(p. 638\)](#).

To identify the accumulation of your noncurrent versioned objects

1. Sign in to the AWS Management Console and open the Amazon S3 console at <https://console.aws.amazon.com/s3/>.
2. In the navigation pane, choose **Storage Lens, Dashboards**.
3. In the **Dashboards** list, choose the dashboard that you want to view.
4. At the top of the **Overview** tab in the **Snapshot** section, choose **Cost efficiency**. The metric for **% noncurrent version bytes** represents the proportion of your total storage bytes (within the scope of the dashboard) that is attributed to noncurrent versions, for the selected date.

Note

If your **% noncurrent version bytes** is greater than 10 percent of your storage at the account level, it could be an indicator that you're storing too many versions.

5. To identify specific buckets that are accumulating a large number of noncurrent versions, scroll down to the **Top N overview for date** section, and select the **% noncurrent version bytes** metric.

After you've determined which buckets require further investigation, you can navigate to the buckets within the S3 console and enable a lifecycle policy to expire noncurrent versions after a specified number of days. Alternatively, to reduce costs while still retaining noncurrent versions, you can configure a lifecycle policy to transition noncurrent versions to S3 Glacier Flexible Retrieval. For more information, see [Example 6: Specifying a lifecycle rule for a versioning-enabled bucket \(p. 735\)](#).

Uncover cold Amazon S3 buckets

If you have **S3 Storage Lens advanced metrics** enabled, you can use **activity metrics** to understand how cold your S3 buckets are. A "cold" bucket is one whose storage is no longer accessed (or very rarely accessed). This lack of activity typically indicates that the bucket's objects aren't frequently accessed.

Activity metrics, such as **GET Requests** and **Download Bytes**, indicate how often your buckets are accessed each day. To understand the consistency of the access pattern and to spot buckets that are no longer being accessed at all, you can trend this data over several months. The **Retrieval rate** metric, which is computed as **Download bytes / Total storage**, indicates the proportion of storage in a bucket that is accessed daily.

Note

Download bytes are duplicated in cases where the same object is downloaded multiple times during the day.

To see how active your buckets are

1. Sign in to the AWS Management Console and open the Amazon S3 console at <https://console.aws.amazon.com/s3/>.

2. In the navigation pane, choose **Storage Lens, Dashboards**.
3. In the **Dashboards** list, choose the dashboard that you want to view.
4. Choose the **Bucket** tab of the dashboard and scroll down to the **Bubble analysis by buckets for date** charts.
5. In the **Bubble analysis** section, you can plot your buckets on multiple dimensions using any three metrics to represent the **x-axis**, **y-axis**, and **size** of the bubble. Select **% retrieval rate** as one of the metrics.
6. To find buckets that have gone cold, do a bubble analysis using the **Total storage**, **% retrieval rate**, and **Average object size** metrics. Look for any buckets with retrieval rates of zero (or near zero) and a larger relative storage size.

From here, you can identify the bucket owners in your account or organization and find out if that storage is still needed. You can then optimize costs by configuring [lifecycle expiration policies](#) for the buckets or archiving the data in [Amazon S3 Glacier](#).

To avoid the problem of cold buckets going forward, you can [automatically transition your data using S3 Lifecycle policies](#) for your buckets, or you can enable [auto-archiving with S3 Intelligent-Tiering](#).

Conversely, using the preceding example, you can identify hot buckets and see if they have been optimized to serve their requests most effectively by ensuring that the correct [S3 storage class](#) is being used for them.

Amazon S3 Storage Lens metrics glossary

By default, all dashboards are configured with *free metrics*, which include *usage metrics* aggregated down to the bucket level, and data is available for queries for 14 days. This means that you can see all the usage metrics that S3 Storage Lens aggregates, and your metrics are available for queries 14 days from the day the data was aggregated.

Advanced metrics and recommendations include *usage metrics* that can be aggregated by prefix and *activity metrics*. Activity metrics can be aggregated by bucket. Data is available for queries for 15 months. There are additional charges when you use S3 Storage Lens with advanced metrics and recommendations. For more information, see [Amazon S3 pricing](#).

Amazon CloudWatch publishing option

You can publish S3 Storage Lens usage and activity metrics to Amazon CloudWatch to create a unified view of your operational health in CloudWatch [dashboards](#). CloudWatch publishing is an option that you can enable in *Advanced metrics and recommendations*. Storage Lens metrics are published to the AWS/S3/Storage-Lens namespace. For a list of S3 Storage Lens dimensions in CloudWatch, see [Dimensions \(p. 1070\)](#). For a list of S3 Storage Lens metrics in CloudWatch, see the **Metric name in CloudWatch and export** column in the following table.

Derived metrics

Derived metrics are not available for the metrics export and CloudWatch publishing option. However, you can use the metrics formula shown in the **Derived metrics formula** column to compute them.

Interpreting Amazon S3 Storage Lens metrics unit multiples prefix symbols (K, M, G, etc.)

S3 Storage Lens metrics unit multiples are written using prefix symbols that are represented using the International System of Units (SI) symbols that are standardized by the International Bureau of Weights and Measures (BIPM). They are also used in the Unified Code for Units of Measure (UCUM). For more information, see [List of SI prefixes symbols](#).

Metric name	Metric name in CloudWatch and export	Description	From	Type	Category	Derived	Derived metric formula
Total Storage	StorageBytes	The total storage	Y	Usage	Summary	N	
Object Count	ObjectCount	The total object count	Y	Usage	Summary	N	
# Avg Object Size	NA	The average object size	Y	Usage	Summary	Y	$\text{Sum(StorageBytes)}/\text{sum(ObjectCount)}$
# of Active Buckets	NA	The total number of buckets in active usage with storage > 0 bytes	Y	Usage	Summary	Y	$\text{DistinctCount}[\text{Bucketname}]$
# Accounts	NA	The number of accounts whose storage is in scope	Y	Usage	Summary	Y	$\text{DistinctCount}[\text{AccountID}]$
Current Version Storage Bytes	NA	The number of bytes that are a current version	Y	Usage	Data Protection, Cost Efficiency	N	
% Current Version Bytes	NA	The percentage of bytes in scope that are current version	Y	Usage	Data Protection, Cost Efficiency	Y	$\text{Sum(CurrentVersionBytes) } / \text{sum(StorageBytes)}$
Current Version Object Count	CurrentVersionObjectCount	The count of current version objects	Y	Usage	Data Protection, Cost Efficiency	N	
% Current Version Objects	NA	The percentage of objects in scope that are a current version	Y	Usage	Data Protection, Cost Efficiency	Y	$\text{Sum(CurrentVersionObjects)}/\text{sum(ObjectCount)}$
Non-Current Version Storage Bytes	NonCurrentVersionStorageBytes	The bytes of noncurrent versioned bytes	Y	Usage	Data Protection, Cost Efficiency	N	
% Non-Current Version Bytes	NA	The percentage of bytes in scope that are noncurrent version	Y	Usage	Data Protection, Cost Efficiency	Y	$\text{Sum(NonCurrentVersionStorageBytes)}/\text{Sum(StorageBytes)}$
Non-Current Version Object Count	NonCurrentVersionObjectCount	The count of the noncurrent version objects	Y	Usage	Data Protection, Cost Efficiency	N	
% Non-Current Version Objects	NA	The percentage of objects in	Y	Usage	Data Protection,	Y	$\text{Sum(NonCurrentVersionObjects)}/\text{Sum(ObjectCount)}$

Metric name	Metric name in CloudWatch and export	Description	From	Type	Category	Derived	Derived metric formula
		scope that are a noncurrent version			Cost Efficiency		
Delete Marker Object Count	DeleteMarkerObjectCount	The total number of objects with a delete marker	Y	Usage	Cost Efficiency	N	
% Delete Marker Objects	NA	The percentage of objects in scope with a delete marker	Y	Usage	Cost Efficiency	Y	
Encrypted Storage Bytes	EncryptedStorageBytes	The total number of encrypted bytes using Amazon S3 server-side encryption	Y	Usage	Data Protection	N	
% Encrypted Bytes	NA	The percentage of total bytes in scope that are encrypted using Amazon S3 server-side encryption	Y	Usage	Data Protection	Y	$\frac{\text{Sum(EncryptedStorageBytes)}}{\text{Sum(StorageBytes)}}$
Encrypted Object Count	EncryptedObjectCount	The total object counts that are encrypted using Amazon S3 server-side encryption	Y	Usage	Data Protection	N	
% Encrypted Objects	NA	The percentage of objects in scope that are encrypted using Amazon S3 server-side encryption	Y	Usage	Data Protection	Y	$\frac{\text{Sum(EncryptedStorageBytes)}}{\text{Sum(ObjectCount)}}$
Unencrypted Storage Bytes	NA	The number of bytes in scope that are unencrypted	Y	Usage	Data Protection	Y	$\text{Sum(StorageBytes)} - \text{sum(EncryptedStorageBytes)}$
% Unencrypted Bytes	NA	The percentage of bytes in scope that are unencrypted	Y	Usage	Data Protection	Y	$\frac{\text{Sum(UnencryptedStorageBytes)}}{\text{Sum(StorageBytes)}}$
Unencrypted Object Count	NA	The count of the objects that are unencrypted	Y	Usage	Data Protection	Y	$\text{Sum(ObjectCounts)} - \text{sum(EncryptedObjectCount)}$
% Unencrypted Objects	NA	The percentage of unencrypted objects	Y	Usage	Data Protection	Y	$\frac{\text{Sum(UnencryptedStorageBytes)}}{\text{Sum(ObjectCount)}}$

Metric name	Metric name in CloudWatch and export	Description	From	Type	Category	Derived	Derived metric formula
Replicated Storage Bytes	ReplicatedStorageBytes	The total number of bytes in scope that are replicated	Y	Usage	Data Protection	N	
% Replicated Bytes	NA	The percentage of total bytes in scope that are replicated	Y	Usage	Data Protection	Y	$\text{Sum}(\text{ReplicatedStorageBytes}) / \text{Sum}(\text{StorageBytes})$
Replicated Object Count	ReplicatedObjectsCount	The count of replicated objects	Y	Usage	Data Protection	N	
% Replicated Objects	NA	The percentage of total objects that are replicated	Y	Usage	Data Protection	Y	$\text{Sum}(\text{ReplicatedObjects}) / \text{Sum}(\text{ObjectCount})$
Object Lock Enabled Storage Bytes	ObjectLockEnabledBytes	The total bytes in scope that have Object Lock enabled	Y	Usage	Data Protection	N	
% Object Lock Bytes	NA	The percentage of total bytes in scope that have Object Lock enabled	Y	Usage	Data Protection	Y	$\text{Sum}(\text{ObjectLockBytes}) / \text{Sum}(\text{StorageBytes})$
Object Lock Enabled Object Count	ObjectLockEnabledObjectsCount	The total number of objects in scope that have Object Lock enabled	Y	Usage	Data Protection	N	
% Object Lock Objects	NA	The percentage of objects in scope that have Object Lock enabled	Y	Usage	Data Protection	Y	$\text{Sum}(\text{ObjectLockObjects}) / \text{Sum}(\text{ObjectCount})$
Incomplete Multipart Upload Storage Bytes	IncompleteMultipartUploadStorageBytes	The total bytes in scope with incomplete multipart uploads	Y	Usage	Cost Efficiency	N	
% Incomplete MPU Bytes	NA	The percentage of bytes in scope that are results of incomplete multipart uploads	Y	Usage	Cost Efficiency	Y	$\text{Sum}(\text{IncompleteMPUbytes}) / \text{Sum}(\text{StorageBytes})$
Incomplete Multipart Upload Object Count	IncompleteMultipartUploadObjectCount	The total number of objects in scope that are incomplete multipart uploads	Y	Usage	Cost Efficiency	N	

Metric name	Metric name in CloudWatch and export	Description	From	Type	Category	Derived	Derived metric formula
% Incomplete MPU Objects	NA	The percentage of objects in scope that are incomplete multipart uploads	Y	Usage	Cost Efficiency	Y	$\frac{\text{Sum(IncompleteMPUObjects)}}{\text{Sum(ObjectCount)}}$
All Requests	AllRequests	The total number of requests made	N	Activity	Summary, Activity	N	
Get Requests	GetRequests	The total number of GET requests made	N	Activity	Activity	N	
Put Requests	PutRequests	The total number of PUT requests made	N	Activity	Activity	N	
Head Requests	HeadRequests	The total number of head requests made	N	Activity	Activity	N	
Delete Requests	DeleteRequests	The total number of delete requests made	N	Activity	Activity	N	
List Requests	ListRequests	The total number of list requests made	N	Activity	Activity	N	
Post Requests	PostRequests	The total number of post requests made	N	Activity	Activity	N	
Select Requests	SelectRequests	The total number of select requests	N	Activity	Activity	N	
Select Scanned Bytes	SelectBytesScanned	The number of select bytes scanned	N	Activity	Activity	N	
Select Returned Bytes	SelectBytesReturned	The number of select bytes returned	N	Activity	Activity	N	
Bytes Downloaded	BytesDownloaded	The number of bytes in scope that were downloaded	N	Activity	Activity	N	
% Retrieval Rate	NA	The percentage of retrieval rate	N	Activity	Activity, Cost Efficiency	Y	$\frac{\text{Sum(BytesDownloaded)}}{\text{Sum(StorageBytes)}}$
Bytes Uploaded	BytesUploaded	The number of bytes uploaded	N	Activity	Activity	N	

Metric name	Metric name in CloudWatch and export	Description	From	Type	Category	Derived	Derived metric formula
% Ingest Ratio	NA	The number of bytes loaded as a percentage of total storage bytes in scope	N	Activity	Activity, Cost Efficiency	Y	$\text{Sum(BytesUploaded)} / \text{Sum(StorageBytes)}$
4xx Errors	4xxErrors	The total 4xx errors in scope	N	Activity	Activity	N	
5xx Errors	5xxErrors	The total 5xx errors in scope	N	Activity	Activity	N	
Total Errors	NA	The sum of all the (4xx) and (5xx) errors	N	Activity	Activity	Y	$\text{Sum(4xxErrors)} + \text{Sum(5xxErrors)}$
% Error Rate	NA	The total errors as a percent of total requests	N	Activity	Activity	Y	$\text{Sum(TotalErrors)} / \text{Sum(TotalRequests)}$

Working with Amazon S3 Storage Lens using the console and API

Amazon S3 Storage Lens aggregates your usage and activity metrics and displays the information in the account snapshot on the Amazon S3 console home (**Buckets**) page, interactive dashboards, or through a metrics export that you can download in CSV or Parquet format. You can use the dashboard to visualize insights and trends, flag outliers, and receive recommendations for optimizing storage costs and applying data protection best practices. You can use S3 Storage Lens through the AWS Management Console, AWS CLI, AWS SDKs, or REST API.

The following sections contain examples of creating, updating, and viewing S3 Storage Lens configurations and performing operations related to the feature. If you are using S3 Storage Lens with AWS Organizations, these examples also cover those use cases. In the examples, replace any variable values with those that are specific to you.

Topics

- [Using Amazon S3 Storage Lens on the console \(p. 1086\)](#)
- [Amazon S3 Storage Lens examples using the AWS CLI \(p. 1097\)](#)
- [Amazon S3 Storage Lens examples using the SDK for Java \(p. 1102\)](#)

Using Amazon S3 Storage Lens on the console

Amazon S3 Storage Lens aggregates your usage and activity metrics and displays the information in the account snapshot on the Amazon S3 console home (**Buckets**) page, interactive dashboards, or through a metrics export that you can download in CSV or Parquet format. You can use the dashboard to visualize insights and trends, flag outliers, and receive recommendations for optimizing storage costs and applying data protection best practices. You can use S3 Storage Lens through the AWS Management Console, AWS CLI, AWS SDKs, or REST API.

Topics

- [Viewing an Amazon S3 Storage Lens dashboard \(p. 1087\)](#)
- [Creating and updating Amazon S3 Storage Lens dashboards \(p. 1089\)](#)
- [Disabling or deleting Amazon S3 Storage Lens dashboards \(p. 1094\)](#)
- [Working with AWS Organizations to create organization-level dashboards \(p. 1095\)](#)

Viewing an Amazon S3 Storage Lens dashboard

Amazon S3 Storage Lens aggregates your usage and activity metrics and displays the information in the account snapshot on the Amazon S3 console home (**Buckets**) page, interactive dashboards, or through a metrics export that you can download in CSV or Parquet format. You can use the dashboard to visualize insights and trends, flag outliers, and receive recommendations for optimizing storage costs and applying data protection best practices. You can use S3 Storage Lens through the AWS Management Console, AWS CLI, AWS SDKs, or REST API.

The Amazon S3 Storage Lens default dashboard is **default-account-dashboard**. This dashboard is preconfigured by Amazon S3 to help you visualize summarized insights and trends for your entire account's aggregated storage usage and activity metrics on the console. You can't modify its configuration scope, but you can upgrade the metrics selection from the Free Metrics to the paid Advanced Metrics and Recommendations, configure the optional metrics export, or even disable it. The default dashboard cannot be deleted.

You can also create additional S3 Storage Lens dashboards that are focused on specific AWS Regions, S3 buckets, or other accounts in your organizations.

The Amazon S3 dashboard provides a rich resource of information about its storage scope representing more than 30 metrics. These metrics represent trends and other information, including storage summary, cost efficiency, data protection, and activity.

The dashboard always loads for the latest date for which metrics are available.

To view an S3 Storage Lens dashboard

1. Sign in to the AWS Management Console and open the Amazon S3 console at <https://console.aws.amazon.com/s3/>.
2. In the navigation pane, choose **Storage Lens, Dashboards**.
3. In the **Dashboards** list, choose the dashboard that you want to view.

In the upper-right corner, you should see the latest date that S3 Storage Lens has collected storage metrics for. You also have access to temporary filters to further limit the scope of the dashboard data that you are viewing. There is also a reset option that you can use to remove all filters.

Your dashboard always loads for the latest date for which metrics are available.

Note

You can't use your account's root user credentials to view Amazon S3 Storage Lens dashboards. To access S3 Storage Lens dashboards, you must grant the requisite IAM permissions to a new or existing IAM user. Then, sign in with those user credentials to access S3 Storage Lens dashboards. For more information, see [Amazon S3 Storage Lens permissions \(p. 1061\)](#).

Understanding your S3 Storage Lens dashboard

Your S3 Storage Lens dashboard consists of a primary **Overview** tab, and up to five additional tabs that represent each aggregation level:

- **Account** (for organization-level dashboards only)

- **Region**
- **Storage class**
- **Bucket**
- **Prefix** (only if subscribed to advanced metrics and recommendations)

Your dashboard data is aggregated into three different sections.

Snapshot

The first section is the **Snapshot** section, which shows the metrics that S3 Storage Lens has aggregated for the preceding date selected. It shows aggregated data for the following five metrics from your S3 Storage Lens dashboard's configuration scope:

- Total storage bytes
- Total object count
- Average object size
- Accounts – This value is **1** unless you are using AWS Organizations, and your S3 Storage Lens has trusted access with a valid service-linked role. For more information, see [Using service-linked roles for Amazon S3 Storage Lens \(p. 550\)](#).
- Buckets
- Requests – If you chose to use **Advanced metrics and recommendations** for this dashboard.

The **Metrics** section of the **Snapshot** section shows aggregated data of the storage usage and activity metrics grouped into the following categories:

- Summary
- Cost efficiency
- Data protection
- Activity

You can view the relevant properties for these metrics, including **totals**, **% change (day/day, week/week, and month/month) trends**, and **recommendations**.

Trends and distribution

The second section of the **Overview** tab is **Trends and distribution**.

Trends provide two metrics that you can choose to compare over a date range of your choice aggregated by a period of your choice. It helps you see the relationship between the two metrics trends over your dashboard storage scope. You can see the **Storage class** and **Region** distribution between the two trends that you are tracking.

With the three different ways of comparing metrics, you can get further insights about your storage that can help you optimize your usage over time.

Top N overview

The third section of the S3 Storage Lens dashboard is **Top N overview** (sorted in ascending or descending order). This lets you see your select metrics across the top *N* accounts (if you enabled S3 Storage Lens to work with AWS Organizations).

The **Dimension level** tabs provide a detailed view of all values within a particular dimension. For example, the **Region** tab shows metrics for all AWS Regions, and the **Bucket** tab shows metrics for all buckets. Each dimension tab contains an identical layout consisting of four sections:

- A *trend chart* displaying your top N items within the dimension over the last 30 days for the selected metric. By default, this chart displays the top 10 items, but you can increase it to any number that you want.
- A *histogram chart* shows a vertical bar chart for the selected date and metric. You might need to scroll horizontally if you have a large number of items to display in this chart.
- The *bubble analysis chart* plots all items within the dimension by representing the first metric on the x axis, a second metric on the y axis, and a third metric represented by the size of the bubble.
- The *metric grid view* contains each item in the dimension listed in rows. The columns represent each available metric, arranged in metrics category tabs for easier navigation.

Note

To provide a fluid experience in conducting your analysis, the S3 Storage Lens dashboard provides a **drill-down** action menu, which appears when you choose any chart value. Choose any chart value to see the associated metrics values, and choose from two options:

- The **drill-down** action applies the selected value as a filter across all tabs of your dashboard. You can then drill down into that value for deeper analysis.
- The **analyze-by** action takes you to the selected dimension tab in your dashboard and applies that value as a filter. You can then view that value in context of the new dimension for deeper analysis.

The drill-down and analyze-by actions might not appear if the outcome would yield illogical results or would not have any value. Both the drill-down and analyze-by actions result in filters being applied on top of any existing filters across all tabs of the dashboard. If necessary, you can remove the filters, or use the reset option to remove all filters.

Creating and updating Amazon S3 Storage Lens dashboards

Amazon S3 Storage Lens aggregates your usage and activity metrics and displays the information in the account snapshot on the Amazon S3 console home (**Buckets**) page, interactive dashboards, or through a metrics export that you can download in CSV or Parquet format. You can use the dashboard to visualize insights and trends, flag outliers, and receive recommendations for optimizing storage costs and applying data protection best practices. You can use S3 Storage Lens through the AWS Management Console, AWS CLI, AWS SDKs, or REST API.

The Amazon S3 Storage Lens default dashboard is **default-account-dashboard**. This dashboard is preconfigured by Amazon S3 to help you visualize summarized insights and trends for your entire account's aggregated storage usage and activity metrics on the console. You can't modify its configuration scope, but you can upgrade the metrics selection from the Free Metrics to the paid Advanced Metrics and Recommendations, configure the optional metrics export, or even disable it. The default dashboard cannot be deleted.

You can also create additional S3 Storage Lens custom dashboards that can be scoped to cover your AWS Organizations, or to specific Regions or buckets within an account.

Topics

- [Creating an Amazon S3 Storage Lens dashboard \(p. 1089\)](#)
- [Updating an Amazon S3 Storage Lens dashboard \(p. 1091\)](#)

Creating an Amazon S3 Storage Lens dashboard

Use the following steps to create an Amazon S3 Storage Lens dashboard on the Amazon S3 console.

To define the dashboard scope and metrics selection

1. Sign in to the AWS Management Console and open the Amazon S3 console at <https://console.aws.amazon.com/s3/>.
2. In the navigation pane, choose **S3 Storage Lens**.
3. Choose **Create dashboard**.
4. On the **Dashboard** page, in the **General** section, do the following:
 - a. Enter a dashboard name.

Dashboard names must be fewer than 65 characters and must not contain special characters or spaces.

Note

You can't change this dashboard name after the dashboard is created.

- b. Choose the **Home Region** for your dashboard. Your dashboard metrics for all included Regions in this dashboard scope are stored centrally in this designated home Region.
- c. You can optionally choose to add **Tags** to your dashboard. You can use tags to manage permissions for your dashboard and track costs for S3 Storage Lens.

For more information, see [Controlling access using resource tags](#) in the *IAM User Guide* and [AWS-Generated Cost Allocation Tags](#) in the *AWS Billing User Guide*.

Note

You can add up to 50 tags to your dashboard configuration.

5. In the **Dashboard scope** section, do the following:
 - a. Choose the Regions and buckets that you want S3 Storage Lens to include or exclude in the dashboard.
 - b. Choose the buckets in your selected Regions that you want S3 Storage Lens to include or exclude. You can either include or exclude buckets, but not both. This option is not available when you create organization-level dashboards.

Note

- You can either include or exclude Regions and buckets. This option is limited to Regions only when creating organization-level dashboards across member accounts in your organization.
- You can choose up to 50 buckets to include or exclude.

6. In the **Metrics selection** section, choose the type of metrics that you want to aggregate for this dashboard.
 - Choose **Free Metrics** to include usage metrics aggregated at the bucket level and available for queries for 14 days.
 - For an additional charge, choose **Advanced Metrics and Recommendations**. With Advanced Metrics and Recommendations, you get contextual recommendations that help you further optimize storage costs and apply data protection best practices, and data is available for queries for 15 months. Advanced Metrics and Recommendations also includes usage metrics aggregated at the prefix-level, activity metrics aggregated by bucket, and Amazon CloudWatch publishing. For more information, see [Amazon S3 pricing](#).
7. If you enable Advanced Metrics and Recommendations, you can choose **Advanced metrics and recommendations features**:
 - a. Choose **Activity metrics** to track requests and errors for objects in your dashboard scope.
 - b. Choose **CloudWatch publishing** to publish your S3 Storage Lens metrics to CloudWatch.

For more information about the CloudWatch publishing option, see [Monitor S3 Storage Lens metrics in CloudWatch \(p. 1068\)](#).

- c. Choose **Prefix aggregation** to aggregate your usage metrics at the prefix level so that you can receive detailed insights for your top prefixes in each bucket.

Note

At this time, you can only receive prefix aggregation for usage metrics. Prefix-level metrics do not publish to CloudWatch.

8. If you chose to enable prefix aggregation, configure the following:

- a. Choose the minimum prefix threshold size that S3 Storage Lens will collect for this dashboard. For example, a prefix threshold of 5 percent indicates that prefixes that make up 5 percent or greater in size of the storage of the bucket will be aggregated.
- b. Choose the prefix depth. This setting indicates the maximum number of levels up to which the prefixes are evaluated. The prefix depth must be less than 10.
- c. Enter a prefix delimiter character. This is the value used to identify each prefix level. The default value in Amazon S3 is the / character, but your storage structure might use other delimiter characters.

To export metrics for the dashboard

1. In the **Metrics Export** section, choose **Enable** to create a metrics export that will be placed daily in a destination bucket of your choice.

The metrics export is in CSV or Apache Parquet format. It represents the same scope of data as your S3 Storage Lens dashboard data without the recommendations.

2. If enabled, choose the output format of your daily metrics export. You can choose between **CSV** or **Apache Parquet**. Parquet is an open source file format for Hadoop that stores nested data in a flat columnar format.
3. Choose the destination S3 bucket for your metrics export. You can choose a bucket in the current account of the S3 Storage Lens dashboard. Or you can choose another AWS account if you have the destination bucket permissions and the destination bucket owner account ID.
4. Choose the destination (format: s3://bucket/prefix) of the destination S3 bucket. The bucket address must be in S3 format in the home Region of your S3 Storage Lens dashboard.

Note

- Amazon S3 will update the permissions policy on the destination bucket to allow S3 to place data in that bucket.
 - The S3 console will show you the explicit destination bucket permission that will be added by Amazon S3 to the destination bucket policy in the destination bucket permission box.
 - If your metrics export destination S3 bucket has server-side encryption already enabled, all export files that are placed there must also have server-side encryption enabled.
5. If you choose to enable server-side encryption for your dashboard, you must choose an encryption key type. You can choose between an [Amazon S3 key \(SSE-S3\)](#) and an [AWS Key Management Service \(AWS KMS\)](#) key (SSE-KMS).
 6. If you chose an AWS KMS key, you must choose from your KMS keys or enter a key Amazon Resource Name (ARN).
 7. Choose **Create dashboard**.

Updating an Amazon S3 Storage Lens dashboard

Use the following steps to update an Amazon S3 Storage Lens dashboard on the Amazon S3 console.

To update the dashboard scope and metrics selection

1. Sign in to the AWS Management Console and open the Amazon S3 console at <https://console.aws.amazon.com/s3/>.
2. In the navigation pane, choose **S3 Storage Lens**.
3. Choose the dashboard that you want to edit, and then choose **Edit** at the top of the list.

Note

You can't change the following:

- The dashboard name
- The home Region
- The dashboard scope of the default dashboard, which is scoped to your entire account's storage.

4. On the dashboard configuration page, in the **General** section, you can update and add tags to your dashboard.

You can use tags to manage permissions for your dashboard and to track costs for S3 Storage Lens. For more information, see [Controlling access using resource tags](#) in the *IAM User Guide* and [AWS-Generated Cost Allocation Tags](#) in the *AWS Billing User Guide*.

Note

You can add up to 50 tags to your dashboard configuration.

5. In the **Dashboard scope** section, do the following:

- Update the Regions and buckets that you want S3 Storage Lens to include or exclude in the dashboard.

Note

- You can either include or exclude Regions and buckets. This option is limited to Regions only when creating organization-level dashboards across member accounts in your organization.
- You can choose up to 50 buckets to include or exclude.

Update the buckets in your selected Regions that you want S3 Storage Lens to include or exclude. You can either include or exclude buckets, but not both. This option is not present when creating organization-level dashboards.

6. In the **Metrics selection** section, choose the type of metrics that you want to aggregate for this dashboard.

- Choose **Free Metrics** to include usage metrics aggregated at the bucket level and 14-day data retention.
- For an additional charge, choose **Advanced Metrics and Recommendations**. With Advanced Metrics and Recommendations, you get contextual recommendations that help you optimize storage costs and apply data protection best practices, and data is available for queries for 15 months. Advanced Metrics and Recommendations also includes usage metrics aggregated at the prefix-level, activity metrics aggregated by bucket, and Amazon CloudWatch publishing. For more information, see [Amazon S3 pricing](#).

7. If you enable Advanced Metrics and Recommendations, you can choose **Advanced metrics and recommendations features**:

- a. Choose **Activity metrics** to track requests and errors for objects in your dashboard scope.
- b. Choose **CloudWatch publishing** to publish your S3 Storage Lens metrics to CloudWatch.

For more information about the CloudWatch publishing option, see [Monitor S3 Storage Lens metrics in CloudWatch \(p. 1068\)](#).

- c. Choose **Prefix aggregation** to aggregate your usage metrics at the prefix level so that you can receive detailed insights for your top prefixes in each bucket.

Note

At this time, you can only receive prefix aggregation for usage metrics. Prefix-level metrics are not publishing to CloudWatch

8. If you chose to enable prefix aggregation, configure the following:
 - a. Choose the minimum prefix threshold size that S3 Storage Lens will collect for this dashboard. For example, a prefix threshold of 5 percent indicates that prefixes that make up 5 percent or more in size of the storage of the bucket will be aggregated.
 - b. Choose the prefix depth. This setting indicates the maximum number of levels up to which the prefixes are evaluated. The prefix depth must be less than 10.
 - c. Enter a prefix delimiter character. This is the value used to identify each prefix level. The default value in Amazon S3 is the / character, but your storage structure might use other delimiter characters.

To configure metrics export

1. Under **Metrics Export**, choose **Enable** if you want to create a metrics export that will be placed daily in a destination bucket of your choice. The metrics export is in CSV or Apache Parquet format and represents the same scope of data as your S3 Storage Lens dashboard data, without the recommendations.
2. If enabled, choose the output format of your daily metrics export. You can choose between **CSV** or **Apache Parquet**. Parquet is an open source file format for Hadoop that stores nested data in a flat columnar format.
3. Update the destination S3 bucket of your metrics export. You can choose between a bucket in the current account for the S3 Storage Lens dashboard, or choose another AWS account if you have the destination bucket permissions and the destination bucket owner account ID.
4. Update the destination (format: s3://bucket/prefix) of the destination S3 bucket. The bucket address must be in S3 format in the home Region of your S3 Storage Lens dashboard.

Note

- Amazon S3 will update the permissions policy on the destination bucket to allow S3 to place data in that bucket.
 - The S3 console will show you the explicit destination bucket permission that will be added by Amazon S3 to the destination bucket policy in the destination bucket permission box.
 - If your metrics export destination S3 bucket has server-side encryption already enabled, all export files placed there must also have server-side encryption enabled.
5. If you chose to enable server-side encryption for your dashboard, you must choose an encryption key type. You can choose between an [Amazon S3 key \(SSE-S3\)](#) and an [AWS Key Management Service \(AWS KMS\)](#) key (SSE-KMS).
 6. If you chose an AWS KMS key, you must choose from your KMS keys or enter a key Amazon Resource Name (ARN).
 7. Choose **Save changes**.

You can then view the metrics included for this dashboard.

Disabling or deleting Amazon S3 Storage Lens dashboards

Amazon S3 Storage Lens aggregates your usage and activity metrics and displays the information in the account snapshot on the Amazon S3 console home (**Buckets**) page, interactive dashboards, or through a metrics export that you can download in CSV or Parquet format. You can use the dashboard to visualize insights and trends, flag outliers, and receive recommendations for optimizing storage costs and applying data protection best practices. You can use S3 Storage Lens through the AWS Management Console, AWS CLI, AWS SDKs, or REST API.

The Amazon S3 Storage Lens default dashboard is **default-account-dashboard**. This dashboard is preconfigured by Amazon S3 to help you visualize summarized insights and trends for your entire account's aggregated storage usage and activity metrics on the console. You can't modify its configuration scope, but you can upgrade the metrics selection from the Free Metrics to the paid Advanced Metrics and Recommendations, configure the optional metrics export, or even disable it. The default dashboard cannot be deleted.

You can delete or disable an Amazon S3 Storage Lens dashboard from the Amazon S3 console. Disabling or deleting a dashboard prevents it from generating metrics in the future. A disabled dashboard still retains its configuration information, so it can be easily resumed when re-enabled. A disabled dashboard retains its historical data until it's no longer available for queries.

Data for Free Metrics selections is available for queries for 14 days, and data for Advanced Metrics and Recommendations selections is available for queries for 15 months.

Topics

- [Disabling an Amazon S3 Storage Lens dashboard \(p. 1094\)](#)
- [Deleting an Amazon S3 Storage Lens dashboard \(p. 1094\)](#)

Disabling an Amazon S3 Storage Lens dashboard

To disable an S3 Storage Lens dashboard

1. Sign in to the AWS Management Console and open the Amazon S3 console at <https://console.aws.amazon.com/s3/>.
2. In the navigation pane, choose **Storage Lens, Dashboards**.
3. In the **Dashboards** list, choose the dashboard that you want to disable, and then choose **Disable** at the top of the list.
4. On the confirmation page, confirm that you want to disable the dashboard by entering the name of dashboard into the text field, and then choose **Confirm**.

Deleting an Amazon S3 Storage Lens dashboard

Note

Before deleting a dashboard, consider the following:

- As an alternative to deleting a dashboard, you can *disable* the dashboard so that it is available to be re-enabled in the future. For more information, see [Disabling an Amazon S3 Storage Lens dashboard \(p. 1094\)](#).
- Deleting the dashboard will delete all the configuration settings that are associated with it.
- Deleting a dashboard will make all the historic metrics data unavailable. This historical data is still retained for 15 months. If you want to access this data again, create a dashboard with the same name in the same home Region as the one that was deleted.

To delete an S3 Storage Lens dashboard

1. Sign in to the AWS Management Console and open the Amazon S3 console at <https://console.aws.amazon.com/s3/>.
2. In the navigation pane, choose **Storage Lens, Dashboards**.
3. In the **Dashboards** list, choose the dashboard that you want to delete, and then choose **Delete** at the top of the list.
4. On the **Delete dashboards** page, confirm that you want to delete the dashboard by entering the name of dashboard into the text field. Then choose **Confirm**.

Working with AWS Organizations to create organization-level dashboards

Amazon S3 Storage Lens aggregates your usage and activity metrics and displays the information in the account snapshot on the Amazon S3 console home (**Buckets**) page, interactive dashboards, or through a metrics export that you can download in CSV or Parquet format. You can use the dashboard to visualize insights and trends, flag outliers, and receive recommendations for optimizing storage costs and applying data protection best practices. You can use S3 Storage Lens through the AWS Management Console, AWS CLI, AWS SDKs, or REST API.

The Amazon S3 Storage Lens default dashboard is **default-account-dashboard**. This dashboard is preconfigured by Amazon S3 to help you visualize summarized insights and trends for your entire account's aggregated storage usage and activity metrics on the console. You can't modify its configuration scope, but you can upgrade the metrics selection from the Free Metrics to the paid Advanced Metrics and Recommendations, configure the optional metrics export, or even disable it. The default dashboard cannot be deleted.

You can also create additional S3 Storage Lens dashboards that are focused on specific AWS Regions, S3 buckets, or other AWS accounts in your organization.

The Amazon S3 dashboard provides a rich resource of information about its storage scope representing more than 30 metrics that represent trends and information, including storage summary, cost efficiency, data protection, and activity.

Amazon S3 Storage Lens can be used to collect storage metrics and usage data for all accounts that are part of your AWS Organizations hierarchy. To do this, you must be using AWS Organizations, and you must enable S3 Storage Lens trusted access using your AWS Organizations management account.

When trusted access is enabled, you can add delegate administrator access to accounts in your organization. These accounts can then create organization-wide dashboards and configurations for S3 Storage Lens. For more information about enabling trusted access, see [Amazon S3 Lens and AWS Organizations](#) in the *AWS Organizations User Guide*.

The following console controls are only available to the AWS Organizations management accounts.

Topics

- [Enabling trusted access for S3 Storage Lens in your organization \(p. 1095\)](#)
- [Disabling S3 Storage Lens trusted access in your organization \(p. 1096\)](#)
- [Registering delegated administrators for S3 Storage Lens \(p. 1096\)](#)
- [Deregistering delegated administrators for S3 Storage Lens \(p. 1097\)](#)

[Enabling trusted access for S3 Storage Lens in your organization](#)

Enabling trusted access allows Amazon S3 Storage Lens to access your AWS Organizations hierarchy, membership, and structure through AWS Organization APIs. S3 Storage Lens becomes a trusted service for your entire organization's structure. It can create service-linked roles in your organization's management or delegated administrator accounts whenever a dashboard configuration is created.

The service-linked role grants S3 Storage Lens permissions to describe organizations, list accounts, verify a list of service access for the organizations, and get delegated administrators for the organization. This allows S3 Storage Lens to collect cross-account storage usage and activity metrics for dashboards within accounts in your organizations.

For more information, see [Using service-linked roles for Amazon S3 Storage Lens \(p. 550\)](#).

Note

- Trusted access can only be enabled by the *management account*.
- Only the management account and delegated administrators can create S3 Storage Lens dashboards or configurations for your organization.

To enable S3 Storage Lens to have trusted access

1. Sign in to the AWS Management Console and open the Amazon S3 console at <https://console.aws.amazon.com/s3/>.
2. In the navigation pane, choose **Storage Lens, Organization settings**.
3. In **Organizations access**, choose **Edit**.

The **Organization access** page opens. Here you can **Enable trusted access** for S3 Storage Lens. This allows you and any other account holders that you add as delegated administrators to create dashboards for all accounts and storage in your organization.

[Disabling S3 Storage Lens trusted access in your organization](#)

Disabling trusted access will limit S3 Storage Lens to only work on an account level. Each account holder will only be able to see the benefits of S3 Storage Lens limited to the scope of their account, and not their organization. Any dashboards requiring trusted access will no longer be updated, but they will be able to query their historic data per the [period that data is available for queries](#).

Removing an account as a delegated administrator limits their S3 Storage Lens dashboard metrics access to only work on an account level. Any organizational dashboards that they created will no longer be updated, but they will be able to query their historic data per the [period that it is available for queries](#).

Note

- Disabling trusted access also automatically disables all organization-level dashboards because S3 Storage Lens will no longer have trusted access to the organization accounts to collect and aggregate storage metrics.
- The management and delegate administrator accounts can still see the historic data for these disabled dashboards and can query this data while it is available.

To disable trusted access for S3 Storage Lens

1. Sign in to the AWS Management Console and open the Amazon S3 console at <https://console.aws.amazon.com/s3/>.
2. In the navigation pane, choose **Storage Lens, Organization settings**.
3. In **Organizations access**, choose **Edit**.

The **Organization access** page opens. Here you can **Disable trusted access** for S3 Storage Lens.

[Registering delegated administrators for S3 Storage Lens](#)

After enabling trusted access, you can register delegate administrator access to accounts in your organization. When an account is registered as a delegate administrator, the account receives

authorization to access all read-only AWS Organizations APIs. This provides visibility to the members and structures of your organization so that they can create S3 Storage Lens dashboards on your behalf.

To register delegated administrators for S3 Storage Lens

1. Sign in to the AWS Management Console and open the Amazon S3 console at <https://console.aws.amazon.com/s3/>.
2. In the navigation pane, choose **Storage Lens, Organization settings**.
3. In the **Delegated access** section, for **Accounts**, choose **Add account**.

The **Delegated admin access** page opens. Here you can add an AWS account ID as a delegated administrator to create organization-level dashboards for all accounts and storage in your organization.

Deregistering delegated administrators for S3 Storage Lens

You can deregister delegate administrator access to accounts in your organization. When an account is deregistered as a delegated administrator, the account loses authorization to access all read-only AWS Organizations APIs that provide visibility to the members and structures of your organization.

Note

- Deregistering a delegated administrator also automatically disables all organization-level dashboards created by the delegated administrator.
- The delegate administrator accounts can still see the historic data for these disabled dashboards according to the respective period that data is available for queries.

To deregister accounts for delegated administrator access

1. Sign in to the AWS Management Console and open the Amazon S3 console at <https://console.aws.amazon.com/s3/>.
2. In the navigation pane, choose **Storage Lens, Organization settings**.
3. In the **Accounts with delegated access** section, choose the account ID you want to deregister, and then choose **Remove**.

Amazon S3 Storage Lens examples using the AWS CLI

Amazon S3 Storage Lens aggregates your usage and activity metrics and displays the information in the account snapshot on the Amazon S3 console home (**Buckets**) page, interactive dashboards, or through a metrics export that you can download in CSV or Parquet format. You can use the dashboard to visualize insights and trends, flag outliers, and receive recommendations for optimizing storage costs and applying data protection best practices. You can use S3 Storage Lens through the AWS Management Console, AWS CLI, AWS SDKs, or REST API.. For more information, see [Assessing storage activity and usage with Amazon S3 Storage Lens](#).

The following examples show how you can use S3 Storage Lens with the AWS Command Line Interface.

Topics

- [Helper files for using Amazon S3 Storage Lens \(p. 1098\)](#)
- [Using Amazon S3 Storage Lens configurations with the AWS CLI \(p. 1100\)](#)
- [Using Amazon S3 Storage Lens with AWS Organizations using the AWS CLI \(p. 1101\)](#)

Helper files for using Amazon S3 Storage Lens

Use the following JSON files for key inputs for your examples.

S3 Storage Lens sample configuration JSON

Example config.json

Contains details of a S3 Storage Lens Organizations-level *Advanced Metrics and Recommendations* configuration.

Note

Additional charges apply for Advanced Metrics and Recommendations. For more information, see [Advanced Metrics and Recommendations](#).

```
{
    "Id": "SampleS3StorageLensConfiguration", //Use this property to identify S3 Storage
    //Lens configuration.
    "AwsOrg": { //Use this property when enabling S3 Storage Lens for AWS Organizations
        "Arn": "arn:aws:organizations::222222222222:organization/o-abcdefg"
    },
    "AccountLevel": {
        "ActivityMetrics": {
            "IsEnabled":true
        },
        "BucketLevel": {
            "ActivityMetrics": {
                "IsEnabled":true //Mark this as false if you only want Free Metrics metrics.
            },
            "PrefixLevel": {
                "StorageMetrics": {
                    "IsEnabled":true, //Mark this as false if you only want Free Metrics
                    //metrics.
                    "SelectionCriteria": {
                        "MaxDepth":5,
                        "MinStorageBytesPercentage":1.25,
                        "Delimiter":"/"
                    }
                }
            }
        }
    },
    "Exclude": { //Replace with include if you prefer to include regions.
        "Regions": [
            "eu-west-1"
        ],
        "Buckets": [ //This attribute is not supported for Organizations-level
        //configurations.
            "arn:aws:s3:::source_bucket1"
        ]
    },
    "IsEnabled": true, //Whether the configuration is enabled
    "DataExport": { //Details about the metrics export
        "S3BucketDestination": {
            "OutputSchemaVersion": "V_1",
            "Format": "CSV", //You can add "Parquet" if you prefer.
            "AccountId": "ExampleAWSAccountNo8",
            "Arn": "arn:aws:s3:::destination-bucket-name", // The destination bucket for
            //your metrics export must be in the same Region as your S3 Storage Lens configuration.
            "Prefix": "prefix-for-your-export-destination",
            "Encryption": {
                "SSESS3": {}
            }
        }
    }
}
```

```
        }
    }
}
```

S3 Storage Lens sample configuration tags JSON

Example tags.json

```
[  
  {  
    "Key": "key1",  
    "Value": "value1"  
  },  
  {  
    "Key": "key2",  
    "Value": "value2"  
  }  
]
```

S3 Storage Lens sample configuration IAM permissions

Example permissions.json - Specific dashboard name

This example policy shows S3 Storage Lens IAM permissions with a specific dashboard name specified. Replace *your-dashboard-name* and *example-account-id* with your own values.

```
{  
  "Version": "2012-10-17",  
  "Statement": [  
    {  
      "Effect": "Allow",  
      "Action": [  
        "s3:GetStorageLensConfiguration",  
        "s3>DeleteStorageLensConfiguration",  
        "s3:PutStorageLensConfiguration"  
      ],  
      "Condition": {  
        "StringEquals": {  
          "aws:ResourceTag/key1": "value1"  
        }  
      },  
      "Resource": "arn:aws:s3:us-east-1:example-account-id:storage-lens/your-  
      dashboard-name"  
    }  
  ]  
}
```

Example permissions.json - No specific dashboard name

This example policy shows S3 Storage Lens IAM permissions without a specific dashboard name specified. Replace *example-account-id* with your AWS account ID.

```
{  
  "Version": "2012-10-17",  
  "Statement": [  
    {  
      "Effect": "Allow",  
      "Action": [  
        "s3:GetStorageLensConfiguration",  
        "s3>DeleteStorageLensConfiguration",  
        "s3:PutStorageLensConfiguration"  
      ]  
    }  
  ]  
}
```

```
        "s3:GetStorageLensConfiguration",
        "s3>DeleteStorageLensConfiguration",
        "s3:PutStorageLensConfiguration"
    ],
    "Condition": {
        "StringEquals": {
            "aws:ResourceTag/key1": "value1"
        }
    },
    "Resource": "arn:aws:s3:us-east-1:example-account-id:storage-lens/*"
}
]
```

Using Amazon S3 Storage Lens configurations with the AWS CLI

You can use the AWS CLI to list, create, get and update your S3 Storage Lens configurations. The following examples use the helper JSON files for key inputs.

Topics

- [Put an S3 Storage Lens configuration \(p. 1100\)](#)
- [Put an S3 Storage Lens configuration without tags \(p. 1100\)](#)
- [Get an S3 Storage Lens configuration \(p. 1100\)](#)
- [List S3 Storage Lens configurations without next token \(p. 1101\)](#)
- [List S3 Storage Lens configurations \(p. 1101\)](#)
- [Delete an S3 Storage Lens configuration \(p. 1101\)](#)
- [Put tags to an S3 Storage Lens configuration \(p. 1101\)](#)
- [Get tags for an S3 Storage Lens configuration \(p. 1101\)](#)
- [Delete tags for an S3 Storage Lens configuration \(p. 1101\)](#)

Put an S3 Storage Lens configuration

Example Puts an S3 Storage Lens configuration

```
aws s3control put-storage-lens-configuration --account-id=222222222222 --config-id=your-configuration-id --region=us-east-1 --storage-lens-configuration=file://./config.json --tags=file://./tags.json
```

Put an S3 Storage Lens configuration without tags

Example Put an S3 Storage Lens configuration

```
aws s3control put-storage-lens-configuration --account-id=222222222222 --config-id=your-configuration-id --region=us-east-1 --storage-lens-configuration=file://./config.json
```

Get an S3 Storage Lens configuration

Example Get an S3 Storage Lens configuration

```
aws s3control get-storage-lens-configuration --account-id=222222222222 --config-id=your-configuration-id --region=us-east-1
```

List S3 Storage Lens configurations without next token

Example List S3 Storage Lens configurations without next token

```
aws s3control list-storage-lens-configurations --account-id=222222222222 --region=us-east-1
```

List S3 Storage Lens configurations

Example List S3 Storage Lens configurations

```
aws s3control list-storage-lens-configurations --account-id=222222222222 --region=us-east-1  
--next-token=abcdefgij1234
```

Delete an S3 Storage Lens configuration

Example Delete an S3 Storage Lens configuration

```
aws s3control delete-storage-lens-configuration --account-id=222222222222 --region=us-  
east-1 --config-id=your-configuration-id
```

Put tags to an S3 Storage Lens configuration

Example Put tags to an S3 Storage Lens configuration

```
aws s3control put-storage-lens-configuration-tagging --account-id=222222222222 --region=us-  
east-1 --config-id=your-configuration-id --tags=file://./tags.json
```

Get tags for an S3 Storage Lens configuration

Example Get tags for an S3 Storage Lens configuration

```
aws s3control get-storage-lens-configuration-tagging --account-id=222222222222 --region=us-  
east-1 --config-id=your-configuration-id
```

Delete tags for an S3 Storage Lens configuration

Example Delete tags for an S3 Storage Lens configuration

```
aws s3control delete-storage-lens-configuration-tagging --account-id=222222222222 --  
region=us-east-1 --config-id=your-configuration-id
```

Using Amazon S3 Storage Lens with AWS Organizations using the AWS CLI

Use Amazon S3 Storage Lens to collect storage metrics and usage data for all accounts that are part of your AWS Organizations hierarchy. For more information, see [Using Amazon S3 Storage Lens with AWS Organizations](#).

Topics

- [Enable Organizations trusted access for S3 Storage Lens \(p. 1102\)](#)
- [Disable Organizations trusted access for S3 Storage Lens \(p. 1102\)](#)
- [Register Organizations delegated administrators for S3 Storage Lens \(p. 1102\)](#)
- [Deregister Organizations delegated administrators for S3 Storage Lens \(p. 1102\)](#)

[Enable Organizations trusted access for S3 Storage Lens](#)

Example Enable Organizations trusted access for S3 Storage Lens

```
aws organizations enable-aws-service-access --service-principal storage-lens.s3.amazonaws.com
```

[Disable Organizations trusted access for S3 Storage Lens](#)

Example Disable Organizations trusted access for S3 Storage Lens

```
aws organizations disable-aws-service-access --service-principal storage-lens.s3.amazonaws.com
```

[Register Organizations delegated administrators for S3 Storage Lens](#)

Example Register Organizations delegated administrators for S3 Storage Lens

```
aws organizations register-delegated-administrator --service-principal storage-lens.s3.amazonaws.com --account-id 123456789012
```

[Deregister Organizations delegated administrators for S3 Storage Lens](#)

Example Deregister Organizations delegated administrators for S3 Storage Lens

```
aws organizations deregister-delegated-administrator --service-principal storage-lens.s3.amazonaws.com --account-id 123456789012
```

[Amazon S3 Storage Lens examples using the SDK for Java](#)

Amazon S3 Storage Lens aggregates your usage and activity metrics and displays the information in the account snapshot on the Amazon S3 console home (**Buckets**) page, interactive dashboards, or through a metrics export that you can download in CSV or Parquet format. You can use the dashboard to visualize insights and trends, flag outliers, and receive recommendations for optimizing storage costs and applying data protection best practices. You can use S3 Storage Lens through the AWS Management Console, AWS CLI, AWS SDKs, or REST API.. For more information, see [Assessing storage activity and usage with Amazon S3 Storage Lens](#).

The following examples show how you can use S3 Storage Lens with the AWS SDK for Java.

Topics

- [Using Amazon S3 Storage Lens configurations using the SDK for Java \(p. 1103\)](#)

- [Using Amazon S3 Storage Lens with your AWS Organizations using the SDK for Java \(p. 1111\)](#)

Using Amazon S3 Storage Lens configurations using the SDK for Java

You can use the SDK for Java to list, create, get and update your S3 Storage Lens configurations. The following examples use the helper json files for key inputs.

Topics

- [Create and update an S3 Storage Lens configuration \(p. 1103\)](#)
- [Delete an S3 Storage Lens configuration \(p. 1105\)](#)
- [Gets an S3 Storage Lens configuration \(p. 1105\)](#)
- [Lists S3 Storage Lens configurations \(p. 1106\)](#)
- [Put tags to an S3 Storage Lens configuration \(p. 1107\)](#)
- [Get tags for an S3 Storage Lens configuration \(p. 1108\)](#)
- [Delete tags for an S3 Storage Lens configuration \(p. 1109\)](#)
- [Update default S3 Storage Lens configuration with Advanced Metrics and Recommendations \(p. 1109\)](#)

Create and update an S3 Storage Lens configuration

Example Create and update an S3 Storage Lens configuration

```
package aws.example.s3control;

import com.amazonaws.AmazonServiceException;
import com.amazonaws.SdkClientException;
import com.amazonaws.auth.profile.ProfileCredentialsProvider;
import com.amazonaws.services.s3control.AWSS3Control;
import com.amazonaws.services.s3control.AWSS3ControlClient;
import com.amazonaws.services.s3control.model.AccountLevel;
import com.amazonaws.services.s3control.model.ActivityMetrics;
import com.amazonaws.services.s3control.model.BucketLevel;
import com.amazonaws.services.s3control.model.Format;
import com.amazonaws.services.s3control.model.Include;
import com.amazonaws.services.s3control.model.OutputSchemaVersion;
import com.amazonaws.services.s3control.model.PrefixLevel;
import com.amazonaws.services.s3control.model.PrefixLevelStorageMetrics;
import com.amazonaws.services.s3control.model.PutStorageLensConfigurationRequest;
import com.amazonaws.services.s3control.model.S3BucketDestination;
import com.amazonaws.services.s3control.model.SSSE3;
import com.amazonaws.services.s3control.model.SelectionCriteria;
import com.amazonaws.services.s3control.model.StorageLensAwsOrg;
import com.amazonaws.services.s3control.model.StorageLensConfiguration;
import com.amazonaws.services.s3control.model.StorageLensDataExport;
import com.amazonaws.services.s3control.model.StorageLensDataExportEncryption;
import com.amazonaws.services.s3control.model.StorageLensTag;

import java.util.Arrays;
import java.util.List;

import static com.amazonaws.regions.Regions.US_WEST_2;

public class CreateAndUpdateDashboard {

    public static void main(String[] args) {
        String configurationId = "ConfigurationId";
        String sourceAccountId = "Source Account ID";
        String exportAccountId = "Destination Account ID";
    }
}
```

```

String exportBucketArn = "arn:aws:s3:::destBucketName"; // The destination bucket
for your metrics export must be in the same Region as your S3 Storage Lens configuration.
String awsOrgARN = "arn:aws:organizations::2222222222:organization/o-abcdefg";
Format exportFormat = Format.CSV;

try {
    SelectionCriteria selectionCriteria = new SelectionCriteria()
        .withDelimiter("/")
        .withMaxDepth(5)
        .withMinStorageBytesPercentage(10.0);
    PrefixLevelStorageMetrics prefixStorageMetrics = new
PrefixLevelStorageMetrics()
        .withIsEnabled(true)
        .withSelectionCriteria(selectionCriteria);
    BucketLevel bucketLevel = new BucketLevel()
        .withActivityMetrics(new ActivityMetrics().withIsEnabled(true))
        .withPrefixLevel(new
PrefixLevel().withStorageMetrics(prefixStorageMetrics));
    AccountLevel accountLevel = new AccountLevel()
        .withActivityMetrics(new ActivityMetrics().withIsEnabled(true))
        .withBucketLevel(bucketLevel);

    Include include = new Include()
        .withBuckets(Arrays.asList("arn:aws:s3:::bucketName"))
        .withRegions(Arrays.asList("us-west-2"));

    StorageLensDataExportEncryption exportEncryption = new
StorageLensDataExportEncryption()
        .withSSES3(new SSES3());
    S3BucketDestination s3BucketDestination = new S3BucketDestination()
        .withAccountId(exportAccountId)
        .withArn(exportBucketArn)
        .withEncryption(exportEncryption)
        .withFormat(exportFormat)
        .withOutputSchemaVersion(OutputSchemaVersion.V_1)
        .withPrefix("Prefix");
    StorageLensDataExport dataExport = new StorageLensDataExport()
        .withS3BucketDestination(s3BucketDestination);

    StorageLensAwsOrg awsOrg = new StorageLensAwsOrg()
        .withArn(awsOrgARN);

    StorageLensConfiguration configuration = new StorageLensConfiguration()
        .withId(configurationId)
        .withAccountLevel(accountLevel)
        .withInclude(include)
        .withDataExport(dataExport)
        .withAwsOrg(awsOrg)
        .withIsEnabled(true);

    List<StorageLensTag> tags = Arrays.asList(
        new StorageLensTag().withKey("key-1").withValue("value-1"),
        new StorageLensTag().withKey("key-2").withValue("value-2")
    );
}

AWSS3Control s3ControlClient = AWSS3ControlClient.builder()
    .withCredentials(new ProfileCredentialsProvider())
    .withRegion(US_WEST_2)
    .build();

s3ControlClient.putStorageLensConfiguration(new
PutStorageLensConfigurationRequest()
    .withAccountId(sourceAccountId)
    .withConfigId(configurationId)
    .withStorageLensConfiguration(configuration)
    .withTags(tags))

```

```
        );
    } catch (AmazonServiceException e) {
        // The call was transmitted successfully, but Amazon S3 couldn't process
        // it and returned an error response.
        e.printStackTrace();
    } catch (SdkClientException e) {
        // Amazon S3 couldn't be contacted for a response, or the client
        // couldn't parse the response from Amazon S3.
        e.printStackTrace();
    }
}
```

Delete an S3 Storage Lens configuration

Example Delete an S3 Storage Lens configuration.

```
package aws.example.s3control;

import com.amazonaws.AmazonServiceException;
import com.amazonaws.SdkClientException;
import com.amazonaws.auth.profile.ProfileCredentialsProvider;
import com.amazonaws.services.s3control.AWSS3Control;
import com.amazonaws.services.s3control.AWSS3ControlClient;
import com.amazonaws.services.s3control.model.DeleteStorageLensConfigurationRequest;

import static com.amazonaws.regions.Regions.US_WEST_2;

public class DeleteDashboard {

    public static void main(String[] args) {
        String configurationId = "ConfigurationId";
        String sourceAccountId = "Source Account ID";
        try {
            AWSS3Control s3ControlClient = AWSS3ControlClient.builder()
                .withCredentials(new ProfileCredentialsProvider())
                .withRegion(US_WEST_2)
                .build();

            s3ControlClient.deleteStorageLensConfiguration(new
DeleteStorageLensConfigurationRequest()
                .withAccountId(sourceAccountId)
                .withConfigId(configurationId)
            );
        } catch (AmazonServiceException e) {
            // The call was transmitted successfully, but Amazon S3 couldn't process
            // it and returned an error response.
            e.printStackTrace();
        } catch (SdkClientException e) {
            // Amazon S3 couldn't be contacted for a response, or the client
            // couldn't parse the response from Amazon S3.
            e.printStackTrace();
        }
    }
}
```

Gets an S3 Storage Lens configuration

Example Get an S3 Storage Lens configuration

```
package aws.example.s3control;
```

```

import com.amazonaws.AmazonServiceException;
import com.amazonaws.SdkClientException;
import com.amazonaws.auth.profile.ProfileCredentialsProvider;
import com.amazonaws.services.s3control.AWSS3Control;
import com.amazonaws.services.s3control.AWSS3ControlClient;
import com.amazonaws.services.s3control.model.GetStorageLensConfigurationRequest;
import com.amazonaws.services.s3control.model.GetStorageLensConfigurationResult;
import com.amazonaws.services.s3control.model.StorageLensConfiguration;

import static com.amazonaws.regions.Regions.US_WEST_2;

public class GetDashboard {

    public static void main(String[] args) {
        String configurationId = "ConfigurationId";
        String sourceAccountId = "Source Account ID";

        try {
            AWSS3Control s3ControlClient = AWSS3ControlClient.builder()
                .withCredentials(new ProfileCredentialsProvider())
                .withRegion(US_WEST_2)
                .build();

            final StorageLensConfiguration configuration =
                s3ControlClient.getStorageLensConfiguration(new
                    GetStorageLensConfigurationRequest()
                        .withAccountId(sourceAccountId)
                        .withConfigId(configurationId)
                ).getStorageLensConfiguration();

            System.out.println(configuration.toString());
        } catch (AmazonServiceException e) {
            // The call was transmitted successfully, but Amazon S3 couldn't process
            // it and returned an error response.
            e.printStackTrace();
        } catch (SdkClientException e) {
            // Amazon S3 couldn't be contacted for a response, or the client
            // couldn't parse the response from Amazon S3.
            e.printStackTrace();
        }
    }
}

```

Lists S3 Storage Lens configurations

Example Lists S3 Storage Lens configurations

```

package aws.example.s3control;

import com.amazonaws.AmazonServiceException;
import com.amazonaws.SdkClientException;
import com.amazonaws.auth.profile.ProfileCredentialsProvider;
import com.amazonaws.services.s3control.AWSS3Control;
import com.amazonaws.services.s3control.AWSS3ControlClient;
import com.amazonaws.services.s3control.model.ListStorageLensConfigurationEntry;
import com.amazonaws.services.s3control.model.ListStorageConfigurationsRequest;

import java.util.List;

import static com.amazonaws.regions.Regions.US_WEST_2;

public class ListDashboard {

```

```
public static void main(String[] args) {
    String sourceAccountId = "Source Account ID";
    String nextToken = "nextToken";

    try {
        AWSS3Control s3ControlClient = AWSS3ControlClient.builder()
            .withCredentials(new ProfileCredentialsProvider())
            .withRegion(US_WEST_2)
            .build();

        final List<ListStorageLensConfigurationEntry> configurations =
            s3ControlClient.listStorageLensConfigurations(new
ListStorageLensConfigurationsRequest()
                .withAccountId(sourceAccountId)
                .withNextToken(nextToken)
            ).getStorageLensConfigurationList();

        System.out.println(configurations.toString());
    } catch (AmazonServiceException e) {
        // The call was transmitted successfully, but Amazon S3 couldn't process
        // it and returned an error response.
        e.printStackTrace();
    } catch (SdkClientException e) {
        // Amazon S3 couldn't be contacted for a response, or the client
        // couldn't parse the response from Amazon S3.
        e.printStackTrace();
    }
}
}
```

Put tags to an S3 Storage Lens configuration

Example Put tags to an S3 Storage Lens configuration

```
package aws.example.s3control;

import com.amazonaws.AmazonServiceException;
import com.amazonaws.SdkClientException;
import com.amazonaws.auth.profile.ProfileCredentialsProvider;
import com.amazonaws.services.s3control.AWSS3Control;
import com.amazonaws.services.s3control.AWSS3ControlClient;
import com.amazonaws.services.s3control.model.PutStorageLensConfigurationTaggingRequest;
import com.amazonaws.services.s3control.model.StorageLensTag;

import java.util.Arrays;
import java.util.List;

import static com.amazonaws.regions.Regions.US_WEST_2;

public class PutDashboardTagging {

    public static void main(String[] args) {
        String configurationId = "ConfigurationId";
        String sourceAccountId = "Source Account ID";

        try {
            List<StorageLensTag> tags = Arrays.asList(
                new StorageLensTag().withKey("key-1").withValue("value-1"),
                new StorageLensTag().withKey("key-2").withValue("value-2")
            );

            AWSS3Control s3ControlClient = AWSS3ControlClient.builder()
                .withCredentials(new ProfileCredentialsProvider())
                .withRegion(US_WEST_2)
        }
    }
}
```

```
        .build();

        s3ControlClient.putStorageLensConfigurationTagging(new
PutStorageLensConfigurationTaggingRequest()
        .withAccountId(sourceAccountId)
        .withConfigId(configurationId)
        .withTags(tags)
    );
} catch (AmazonServiceException e) {
    // The call was transmitted successfully, but Amazon S3 couldn't process
    // it and returned an error response.
    e.printStackTrace();
} catch (SdkClientException e) {
    // Amazon S3 couldn't be contacted for a response, or the client
    // couldn't parse the response from Amazon S3.
    e.printStackTrace();
}
}
```

Get tags for an S3 Storage Lens configuration

Example Get tags for an S3 Storage Lens configuration

```
package aws.example.s3control;

import com.amazonaws.AmazonServiceException;
import com.amazonaws.SdkClientException;
import com.amazonaws.auth.profile.ProfileCredentialsProvider;
import com.amazonaws.services.s3control.AWS3Control;
import com.amazonaws.services.s3control.AWS3ControlClient;
import com.amazonaws.services.s3control.model.DeleteStorageLensConfigurationRequest;
import com.amazonaws.services.s3control.model.GetStorageLensConfigurationTaggingRequest;
import com.amazonaws.services.s3control.model.StorageLensTag;

import java.util.List;

import static com.amazonaws.regions.Regions.US_WEST_2;

public class GetDashboardTagging {

    public static void main(String[] args) {
        String configurationId = "ConfigurationId";
        String sourceAccountId = "Source Account ID";
        try {
            AWS3Control s3ControlClient = AWS3ControlClient.builder()
                .withCredentials(new ProfileCredentialsProvider())
                .withRegion(US_WEST_2)
                .build();

            final List<StorageLensTag> s3Tags = s3ControlClient
                .getStorageLensConfigurationTagging(new
GetStorageLensConfigurationTaggingRequest()
                .withAccountId(sourceAccountId)
                .withConfigId(configurationId)
            ).getTags();

            System.out.println(s3Tags.toString());
        } catch (AmazonServiceException e) {
            // The call was transmitted successfully, but Amazon S3 couldn't process
            // it and returned an error response.
            e.printStackTrace();
        } catch (SdkClientException e) {
            // Amazon S3 couldn't be contacted for a response, or the client
```

```
        // couldn't parse the response from Amazon S3.
        e.printStackTrace();
    }
}
```

Delete tags for an S3 Storage Lens configuration

Example Delete tags for an S3 Storage Lens configuration

```
package aws.example.s3control;

import com.amazonaws.AmazonServiceException;
import com.amazonaws.SdkClientException;
import com.amazonaws.auth.profile.ProfileCredentialsProvider;
import com.amazonaws.services.s3control.AWS3Control;
import com.amazonaws.services.s3control.AWSS3ControlClient;
import com.amazonaws.services.s3control.model.DeleteStorageLensConfigurationTaggingRequest;

import static com.amazonaws.regions.Regions.US_WEST_2;

public class DeleteDashboardTagging {

    public static void main(String[] args) {
        String configurationId = "ConfigurationId";
        String sourceAccountId = "Source Account ID";
        try {
            AWSS3Control s3ControlClient = AWSS3ControlClient.builder()
                .withCredentials(new ProfileCredentialsProvider())
                .withRegion(US_WEST_2)
                .build();

            s3ControlClient.deleteStorageLensConfigurationTagging(new
DeleteStorageLensConfigurationTaggingRequest()
                .withAccountId(sourceAccountId)
                .withConfigId(configurationId)
            );
        } catch (AmazonServiceException e) {
            // The call was transmitted successfully, but Amazon S3 couldn't process
            // it and returned an error response.
            e.printStackTrace();
        } catch (SdkClientException e) {
            // Amazon S3 couldn't be contacted for a response, or the client
            // couldn't parse the response from Amazon S3.
            e.printStackTrace();
        }
    }
}
```

Update default S3 Storage Lens configuration with Advanced Metrics and Recommendations

Example Update default S3 Storage Lens configuration with Advanced Metrics and Recommendations

```
package aws.example.s3control;

import com.amazonaws.AmazonServiceException;
import com.amazonaws.SdkClientException;
import com.amazonaws.auth.profile.ProfileCredentialsProvider;
import com.amazonaws.services.s3control.AWS3Control;
import com.amazonaws.services.s3control.AWSS3ControlClient;
import com.amazonaws.services.s3control.model.AccountLevel;
```

```

import com.amazonaws.services.s3control.model.ActivityMetrics;
import com.amazonaws.services.s3control.model.BucketLevel;
import com.amazonaws.services.s3control.model.Format;
import com.amazonaws.services.s3control.model.Include;
import com.amazonaws.services.s3control.model.OutputSchemaVersion;
import com.amazonaws.services.s3control.model.PrefixLevel;
import com.amazonaws.services.s3control.model.PrefixLevelStorageMetrics;
import com.amazonaws.services.s3control.model.PutStorageLensConfigurationRequest;
import com.amazonaws.services.s3control.model.S3BucketDestination;
import com.amazonaws.services.s3control.model.SSSE3;
import com.amazonaws.services.s3control.model.SelectionCriteria;
import com.amazonaws.services.s3control.model.StorageLensAwsOrg;
import com.amazonaws.services.s3control.model.StorageLensConfiguration;
import com.amazonaws.services.s3control.model.StorageLensDataExport;
import com.amazonaws.services.s3control.model.StorageLensDataExportEncryption;
import com.amazonaws.services.s3control.model.StorageLensTag;

import java.util.Arrays;
import java.util.List;

import static com.amazonaws.regions.Regions.US_WEST_2;

public class UpdateDefaultConfigWithPaidFeatures {

    public static void main(String[] args) {
        String configurationId = "default-account-dashboard"; // This configuration ID
        cannot be modified
        String sourceAccountId = "Source Account ID";

        try {
            SelectionCriteria selectionCriteria = new SelectionCriteria()
                .withDelimiter("/")
                .withMaxDepth(5)
                .withMinStorageBytesPercentage(10.0);
            PrefixLevelStorageMetrics prefixStorageMetrics = new
            PrefixLevelStorageMetrics()
                .with.IsEnabled(true)
                .withSelectionCriteria(selectionCriteria);
            BucketLevel bucketLevel = new BucketLevel()
                .withActivityMetrics(new ActivityMetrics().with.IsEnabled(true))
                .withPrefixLevel(new
            PrefixLevel().withStorageMetrics(prefixStorageMetrics));
            AccountLevel accountLevel = new AccountLevel()
                .withActivityMetrics(new ActivityMetrics().with.IsEnabled(true))
                .withBucketLevel(bucketLevel);

            StorageLensConfiguration configuration = new StorageLensConfiguration()
                .withId(configurationId)
                .withAccountLevel(accountLevel)
                .with.IsEnabled(true);

            AWSS3Control s3ControlClient = AWSS3ControlClient.builder()
                .withCredentials(new ProfileCredentialsProvider())
                .withRegion(US_WEST_2)
                .build();

            s3ControlClient.putStorageLensConfiguration(new
            PutStorageLensConfigurationRequest()
                .withAccountId(sourceAccountId)
                .withConfigId(configurationId)
                .withStorageLensConfiguration(configuration)
            );
        } catch (AmazonServiceException e) {
            // The call was transmitted successfully, but Amazon S3 couldn't process
            // it and returned an error response.
        }
    }
}

```

```
        e.printStackTrace();
    } catch (SdkClientException e) {
        // Amazon S3 couldn't be contacted for a response, or the client
        // couldn't parse the response from Amazon S3.
        e.printStackTrace();
    }
}
```

Note

Additional charges apply for Advanced Metrics and Recommendations. For more information, see [Advanced Metrics and Recommendations](#).

Using Amazon S3 Storage Lens with your AWS Organizations using the SDK for Java

Use Amazon S3 Storage Lens to collect storage metrics and usage data for all accounts that are part of your AWS Organizations hierarchy. For more information, see [Using Amazon S3 Storage Lens with AWS Organizations](#).

Topics

- [Enable Organizations trusted access for S3 Storage Lens \(p. 1111\)](#)
- [Disable Organizations trusted access for S3 Storage Lens \(p. 1112\)](#)
- [Register Organizations delegated administrators for S3 Storage Lens \(p. 1112\)](#)
- [Deregister Organizations delegated administrators for S3 Storage Lens \(p. 1113\)](#)

[Enable Organizations trusted access for S3 Storage Lens](#)

Example Enable Organizations trusted access for S3 Storage Lens

```
import com.amazonaws.AmazonServiceException;
import com.amazonaws.SdkClientException;
import com.amazonaws.auth.profile.ProfileCredentialsProvider;
import com.amazonaws.regions.Regions;
import com.amazonaws.services.organizations.AWSOrganizations;
import com.amazonaws.services.organizations.AWSOrganizationsClient;
import com.amazonaws.services.organizations.model.EnableAWSServiceAccessRequest;

public class EnableOrganizationsTrustedAccess {
    private static final String S3_STORAGE_LENS_SERVICE_PRINCIPAL = "storage-
lens.s3.amazonaws.com";

    public static void main(String[] args) {
        try {
            AWSOrganizations organizationsClient = AWSOrganizationsClient.builder()
                .withCredentials(new ProfileCredentialsProvider())
                .withRegion(Regions.US_EAST_1)
                .build();

            organizationsClient.enableAWSServiceAccess(new EnableAWSServiceAccessRequest()
                .withServicePrincipal(S3_STORAGE_LENS_SERVICE_PRINCIPAL));
        } catch (AmazonServiceException e) {
            // The call was transmitted successfully, but AWS Organizations couldn't
process
            // it and returned an error response.
            e.printStackTrace();
        } catch (SdkClientException e) {
            // AWS Organizations couldn't be contacted for a response, or the client
        }
    }
}
```

```
// couldn't parse the response from AWS Organizations.  
e.printStackTrace();  
}  
}  
}
```

Disable Organizations trusted access for S3 Storage Lens

Example Disable Organizations trusted access for S3 Storage Lens

```
import com.amazonaws.AmazonServiceException;  
import com.amazonaws.SdkClientException;  
import com.amazonaws.auth.profile.ProfileCredentialsProvider;  
import com.amazonaws.regions.Regions;  
import com.amazonaws.services.organizations.AWSOrganizations;  
import com.amazonaws.services.organizations.AWSOrganizationsClient;  
import com.amazonaws.services.organizations.model.DisableAWSServiceAccessRequest;  
  
public class DisableOrganizationsTrustedAccess {  
    private static final String S3_STORAGE_LENS_SERVICE_PRINCIPAL = "storage-  
lens.s3.amazonaws.com";  
  
    public static void main(String[] args) {  
        try {  
            AWSOrganizations organizationsClient = AWSOrganizationsClient.builder()  
                .withCredentials(new ProfileCredentialsProvider())  
                .withRegion(Regions.US_EAST_1)  
                .build();  
  
            // Make sure to remove any existing delegated administrator for S3 Storage Lens  
            // before disabling access, otherwise the request will fail.  
            organizationsClient.disableAWSServiceAccess(new  
                DisableAWSServiceAccessRequest()  
                    .withServicePrincipal(S3_STORAGE_LENS_SERVICE_PRINCIPAL));  
        } catch (AmazonServiceException e) {  
            // The call was transmitted successfully, but AWS Organizations couldn't  
            // process  
            // it and returned an error response.  
            e.printStackTrace();  
        } catch (SdkClientException e) {  
            // AWS Organizations couldn't be contacted for a response, or the client  
            // couldn't parse the response from AWS Organizations.  
            e.printStackTrace();  
        }  
    }  
}
```

Register Organizations delegated administrators for S3 Storage Lens

Example Register Organizations delegated administrators for S3 Storage Lens

```
import com.amazonaws.AmazonServiceException;  
import com.amazonaws.SdkClientException;  
import com.amazonaws.auth.profile.ProfileCredentialsProvider;  
import com.amazonaws.regions.Regions;  
import com.amazonaws.services.organizations.AWSOrganizations;  
import com.amazonaws.services.organizations.AWSOrganizationsClient;  
import com.amazonaws.services.organizations.model.RegisterDelegatedAdministratorRequest;  
  
public class RegisterOrganizationsDelegatedAdministrator {  
    private static final String S3_STORAGE_LENS_SERVICE_PRINCIPAL = "storage-  
lens.s3.amazonaws.com";
```

```
public static void main(String[] args) {
    try {
        String delegatedAdminAccountId = "253880222538"; // Account ID for the
delegated administrator
        AWSOrganizations organizationsClient = AWSOrganizationsClient.builder()
            .withCredentials(new ProfileCredentialsProvider())
            .withRegion(Regions.US_EAST_1)
            .build();

        organizationsClient.registerDelegatedAdministrator(new
RegisterDelegatedAdministratorRequest()
            .withAccountId(delegatedAdminAccountId)
            .withServicePrincipal(S3_STORAGE_LENS_SERVICE_PRINCIPAL));
    } catch (AmazonServiceException e) {
        // The call was transmitted successfully, but AWS Organizations couldn't
process
        // it and returned an error response.
        e.printStackTrace();
    } catch (SdkClientException e) {
        // AWS Organizations couldn't be contacted for a response, or the client
        // couldn't parse the response from AWS Organizations.
        e.printStackTrace();
    }
}
}
```

Deregister Organizations delegated administrators for S3 Storage Lens

Example Deregister Organizations delegated administrators for S3 Storage Lens

```
import com.amazonaws.AmazonServiceException;
import com.amazonaws.SdkClientException;
import com.amazonaws.auth.profile.ProfileCredentialsProvider;
import com.amazonaws.regions.Regions;
import com.amazonaws.services.organizations.AWSOrganizations;
import com.amazonaws.services.organizations.AWSOrganizationsClient;
import com.amazonaws.services.organizations.model.DeregisterDelegatedAdministratorRequest;

public class DeregisterOrganizationsDelegatedAdministrator {
    private static final String S3_STORAGE_LENS_SERVICE_PRINCIPAL = "storage-
lens.s3.amazonaws.com";

    public static void main(String[] args) {
        try {
            String delegatedAdminAccountId = "Account ID"; // Account ID for the delegated
administrator
            AWSOrganizations organizationsClient = AWSOrganizationsClient.builder()
                .withCredentials(new ProfileCredentialsProvider())
                .withRegion(Regions.US_EAST_1)
                .build();

            organizationsClient.deregisterDelegatedAdministrator(new
DeregisterDelegatedAdministratorRequest()
                .withAccountId(delegatedAdminAccountId)
                .withServicePrincipal(S3_STORAGE_LENS_SERVICE_PRINCIPAL));
        } catch (AmazonServiceException e) {
            // The call was transmitted successfully, but AWS Organizations couldn't
process
            // it and returned an error response.
            e.printStackTrace();
        } catch (SdkClientException e) {
            // AWS Organizations couldn't be contacted for a response, or the client
            // couldn't parse the response from AWS Organizations.
        }
    }
}
```

```
        e.printStackTrace();
    }
}
```

Tracing Amazon S3 requests using AWS X-Ray

AWS X-Ray collects data about requests that your application serves. You can then view and filter the data to identify and troubleshoot performance issues and errors in your distributed applications and micro-services architecture. For any traced request to your application, it shows you detailed information about the request, the response, and the calls that your application makes to downstream AWS resources, micro-services, databases, and HTTP web APIs.

For more information, see [What is AWS X-Ray?](#) in the *AWS X-Ray Developer Guide*.

Topics

- [How X-Ray works with Amazon S3 \(p. 1114\)](#)
- [Available Regions \(p. 1115\)](#)

How X-Ray works with Amazon S3

AWS X-Ray supports trace context propagation for Amazon S3, so you can view end-to-end requests as they travel through your entire application. X-Ray aggregates the data that is generated by the individual services such as Amazon S3, AWS Lambda, and Amazon EC2, and the many resources that make up your application. It provides you with an overall view of how your application is performing.

Amazon S3 integrates with X-Ray to propagate [trace context](#) and give you one request chain with [upstream and downstream](#) nodes. If an upstream service includes a valid-formatted trace header with its S3 request, Amazon S3 passes the trace header when delivering event notifications to downstream services such as Lambda, Amazon SQS, and Amazon SNS. If you have all these services actively integrated with X-Ray, they are linked in one request chain to give you the complete details of your Amazon S3 requests.

To send X-Ray trace headers through Amazon S3, you must include a [formatted X-Amzn-Trace-Id](#) in your requests. You can also instrument the Amazon S3 client using the AWS X-Ray SDKs. For a list of the supported SDKs, see the [AWS X-Ray documentation](#).

Service maps

X-Ray *service maps* show you the relationships between Amazon S3 and other AWS services and resources in your application in near-real time. To see the end-to-end requests using the X-Ray service maps, you can use the X-Ray console to view a map of the connections between Amazon S3 and other services that your application uses. You can easily detect where high latency is occurring, visualize node distribution for these services, and then drill down into the specific services and paths impacting application performance.

X-Ray Analytics

You can also use the [X-Ray Analytics](#) console to analyze traces, view metrics such as latency and failure rates, and [generate insights](#) to help you identify and troubleshoot issues. This console also shows you metrics such as average latency and failure rates. For more information, see [AWS X-Ray console](#) in the *AWS X-Ray Developer Guide*.

Available Regions

AWS X-Ray support for Amazon S3 is available in all [AWS X-Ray Regions](#). For more information, see [Amazon S3 and AWS X-Ray](#) in the [AWS X-Ray Developer Guide](#).

Hosting a static website using Amazon S3

You can use Amazon S3 to host a static website. On a *static* website, individual webpages include static content. They might also contain client-side scripts.

By contrast, a *dynamic* website relies on server-side processing, including server-side scripts, such as PHP, JSP, or ASP.NET. Amazon S3 does not support server-side scripting, but AWS has other resources for hosting dynamic websites. To learn more about website hosting on AWS, see [Web Hosting](#).

Note

You can use the AWS Amplify Console to host a single-page web app. The AWS Amplify Console supports single-page apps built with single-page app frameworks (for example, React JS, Vue JS, Angular JS, and Nuxt) and static site generators (for example, Gatsby JS, React-static, Jekyll, and Hugo). For more information, see [Getting Started](#) in the *AWS Amplify Console User Guide*. Amazon S3 website endpoints do not support HTTPS. If you want to use HTTPS, you can use Amazon CloudFront to serve a static website hosted on Amazon S3. For more information, see [How do I use CloudFront to serve HTTPS requests for my Amazon S3 bucket?](#) To use HTTPS with a custom domain, see [Configuring a static website using a custom domain registered with Route 53](#).

For more information about hosting a static website on Amazon S3, including instructions and step-by-step walkthroughs, see the following topics.

Topics

- [Website endpoints \(p. 1116\)](#)
- [Enabling website hosting \(p. 1118\)](#)
- [Configuring an index document \(p. 1122\)](#)
- [Configuring a custom error document \(p. 1124\)](#)
- [Setting permissions for website access \(p. 1126\)](#)
- [\(Optional\) Logging web traffic \(p. 1130\)](#)
- [\(Optional\) Configuring a webpage redirect \(p. 1130\)](#)

Website endpoints

When you configure your bucket as a static website, the website is available at the AWS Region-specific website endpoint of the bucket. Website endpoints are different from the endpoints where you send REST API requests. For more information about the differences between the endpoints, see [Key differences between a website endpoint and a REST API endpoint \(p. 1118\)](#).

Depending on your Region, your Amazon S3 website endpoint follows one of these two formats.

- **s3-website dash (-) Region** - `http://bucket-name.s3-website-Region.amazonaws.com`
- **s3-website dot (.) Region** - `http://bucket-name.s3-website.Region.amazonaws.com`

These URLs return the default index document that you configure for the website. For a complete list of Amazon S3 website endpoints, see [Amazon S3 Website Endpoints](#).

If you want your website to be public, you must make all your content publicly readable for your customers to be able to access it at the website endpoint. For more information, see [Setting permissions for website access \(p. 1126\)](#).

Important

Amazon S3 website endpoints do not support HTTPS or access points. If you want to use HTTPS, you can use Amazon CloudFront to serve a static website hosted on Amazon S3. For more information, see [How do I use CloudFront to serve HTTPS requests for my Amazon S3 bucket?](#) To use HTTPS with a custom domain, see [Configuring a static website using a custom domain registered with Route 53](#).

Requester Pays buckets do not allow access through a website endpoint. Any request to such a bucket receives a 403 Access Denied response. For more information, see [Using Requester Pays buckets for storage transfers and usage \(p. 144\)](#).

Topics

- [Website endpoint examples \(p. 1117\)](#)
- [Adding a DNS CNAME \(p. 1117\)](#)
- [Using a custom domain with Route 53 \(p. 1118\)](#)
- [Key differences between a website endpoint and a REST API endpoint \(p. 1118\)](#)

Website endpoint examples

The following examples show how you can access an Amazon S3 bucket that is configured as a static website.

Example — Requesting an object at the root level

To request a specific object that is stored at the root level in the bucket, use the following URL structure.

```
http://bucket-name.s3-website.Region.amazonaws.com/object-name
```

For example, the following URL requests the `photo.jpg` object that is stored at the root level in the bucket.

```
http://example-bucket.s3-website.us-west-2.amazonaws.com/photo.jpg
```

Example — Requesting an object in a prefix

To request an object that is stored in a folder in your bucket, use this URL structure.

```
http://bucket-name.s3-website.Region.amazonaws.com/folder-name/object-name
```

The following URL requests the `docs/doc1.html` object in your bucket.

```
http://example-bucket.s3-website.us-west-2.amazonaws.com/docs/doc1.html
```

Adding a DNS CNAME

If you have a registered domain, you can add a DNS CNAME entry to point to the Amazon S3 website endpoint. For example, if you registered the domain `www.example-bucket.com`, you could create a bucket `www.example-bucket.com`, and add a DNS CNAME record that points to `www.example-`

bucket.com.s3-website.Region.amazonaws.com. All requests to `http://www.example-bucket.com` are routed to `www.example-bucket.com.s3-website.Region.amazonaws.com`.

For more information, see [Customizing Amazon S3 URLs with CNAME records \(p. 1178\)](#).

Using a custom domain with Route 53

Instead of accessing the website using an Amazon S3 website endpoint, you can use your own domain registered with Amazon Route 53 to serve your content—for example, `example.com`. You can use Amazon S3 with Route 53 to host a website at the root domain. For example, if you have the root domain `example.com` and you host your website on Amazon S3, your website visitors can access the site from their browser by entering either `http://www.example.com` or `http://example.com`.

For an example walkthrough, see [Configuring a static website using a custom domain registered with Route 53 \(p. 97\)](#).

Key differences between a website endpoint and a REST API endpoint

An Amazon S3 website endpoint is optimized for access from a web browser. The following table summarizes the key differences between a REST API endpoint and a website endpoint.

Key difference	REST API endpoint	Website endpoint
Access control	Supports both public and private content	Supports only publicly readable content
Error message handling	Returns an XML-formatted error response	Returns an HTML document
Redirection support	Not applicable	Supports both object-level and bucket-level redirects
Requests supported	Supports all bucket and object operations	Supports only GET and HEAD requests on objects
Responses to GET and HEAD requests at the root of a bucket	Returns a list of the object keys in the bucket	Returns the index document that is specified in the website configuration
Secure Sockets Layer (SSL) support	Supports SSL connections	Does not support SSL connections

For a complete list of Amazon S3 endpoints, see [Amazon S3 endpoints and quotas in the AWS General Reference](#).

Enabling website hosting

When you configure a bucket as a static website, you must enable static website hosting, configure an index document, and set permissions.

You can enable static website hosting using the Amazon S3 console, REST API, the AWS SDKs, the AWS CLI, or AWS CloudFormation.

To configure your website with a custom domain, see [Configuring a static website using a custom domain registered with Route 53 \(p. 97\)](#).

Using the S3 console

To enable static website hosting

1. Sign in to the AWS Management Console and open the Amazon S3 console at <https://console.aws.amazon.com/s3/>.
2. In the **Buckets** list, choose the name of the bucket that you want to enable static website hosting for.
3. Choose **Properties**.
4. Under **Static website hosting**, choose **Edit**.
5. Choose **Use this bucket to host a website**.
6. Under **Static website hosting**, choose **Enable**.
7. In **Index document**, enter the file name of the index document, typically `index.html`.

The index document name is case sensitive and must exactly match the file name of the HTML index document that you plan to upload to your S3 bucket. When you configure a bucket for website hosting, you must specify an index document. Amazon S3 returns this index document when requests are made to the root domain or any of the subfolders. For more information, see [Configuring an index document \(p. 1122\)](#).

8. To provide your own custom error document for 4XX class errors, in **Error document**, enter the custom error document file name.

The error document name is case sensitive and must exactly match the file name of the HTML error document that you plan to upload to your S3 bucket. If you don't specify a custom error document and an error occurs, Amazon S3 returns a default HTML error document. For more information, see [Configuring a custom error document \(p. 1124\)](#).

9. (Optional) If you want to specify advanced redirection rules, in **Redirection rules**, enter JSON to describe the rules.

For example, you can conditionally route requests according to specific object key names or prefixes in the request. For more information, see [Configure redirection rules to use advanced conditional redirects \(p. 1131\)](#).

10. Choose **Save changes**.

Amazon S3 enables static website hosting for your bucket. At the bottom of the page, under **Static website hosting**, you see the website endpoint for your bucket.

11. Under **Static website hosting**, note the **Endpoint**.

The **Endpoint** is the Amazon S3 website endpoint for your bucket. After you finish configuring your bucket as a static website, you can use this endpoint to test your website.

Using the REST API

For more information about sending REST requests directly to enable static website hosting, see the following sections in the Amazon Simple Storage Service API Reference:

- [PUT Bucket website](#)
- [GET Bucket website](#)

- [DELETE Bucket website](#)

Using the AWS SDKs

To host a static website on Amazon S3, you configure an Amazon S3 bucket for website hosting and then upload your website content to the bucket. You can also use the AWS SDKs to create, update, and delete the website configuration programmatically. The SDKs provide wrapper classes around the Amazon S3 REST API. If your application requires it, you can send REST API requests directly from your application.

.NET

The following example shows how to use the AWS SDK for .NET to manage website configuration for a bucket. To add a website configuration to a bucket, you provide a bucket name and a website configuration. The website configuration must include an index document and can contain an optional error document. These documents must be stored in the bucket. For more information, see [PUT Bucket website](#). For more information about the Amazon S3 website feature, see [Hosting a static website using Amazon S3 \(p. 1116\)](#).

The following C# code example adds a website configuration to the specified bucket. The configuration specifies both the index document and the error document names. For instructions on how to create and test a working sample, see [Running the Amazon S3 .NET Code Examples \(p. 1193\)](#).

```
using Amazon;
using Amazon.S3;
using Amazon.S3.Model;
using System;
using System.Threading.Tasks;

namespace Amazon.DocSamples.S3
{
    class WebsiteConfigTest
    {
        private const string bucketName = "*** bucket name ***";
        private const string indexDocumentSuffix = "*** index object key ***"; // For
example, index.html.
        private const string errorDocument = "*** error object key ***"; // For
example, error.html.
        // Specify your bucket region (an example region is shown).
        private static readonly RegionEndpoint bucketRegion = RegionEndpoint.USWest2;
        private static IAmazonS3 client;
        public static void Main()
        {
            client = new AmazonS3Client(bucketRegion);
            AddWebsiteConfigurationAsync(bucketName, indexDocumentSuffix,
errorDocument).Wait();
        }

        static async Task AddWebsiteConfigurationAsync(string bucketName,
                                                       string indexDocumentSuffix,
                                                       string errorDocument)
        {
            try
            {
                // 1. Put the website configuration.
                PutBucketWebsiteRequest putRequest = new PutBucketWebsiteRequest()
                {
                    BucketName = bucketName,
                    WebsiteConfiguration = new WebsiteConfiguration()
                    {
                        IndexDocumentSuffix = indexDocumentSuffix,
                        ErrorDocumentKey = errorDocument
                    }
                };
                await client.PutBucketWebsiteAsync(putRequest);
            }
            catch (AmazonS3Exception e)
            {
                Console.WriteLine("Caught exception: " + e.Message);
            }
        }
    }
}
```

```
        ErrorDocument = errorDocument
    }
}
PutBucketWebsiteResponse response = await
client.PutBucketWebsiteAsync(putRequest);

// 2. Get the website configuration.
GetBucketWebsiteRequest getRequest = new GetBucketWebsiteRequest()
{
    BucketName = bucketName
};
GetBucketWebsiteResponse getResponse = await
client.GetBucketWebsiteAsync(getRequest);
Console.WriteLine("Index document: {0}",
getResponse.WebsiteConfiguration.IndexDocumentSuffix);
Console.WriteLine("Error document: {0}",
getResponse.WebsiteConfiguration.ErrorDocument);
}
catch (AmazonS3Exception e)
{
    Console.WriteLine("Error encountered on server. Message:{0}' when
writing an object", e.Message);
}
catch (Exception e)
{
    Console.WriteLine("Unknown encountered on server. Message:{0}' when
writing an object", e.Message);
}
}
}
```

PHP

The following PHP example adds a website configuration to the specified bucket. The `create_website_config` method explicitly provides the index document and error document names. The example also retrieves the website configuration and prints the response. For more information about the Amazon S3 website feature, see [Hosting a static website using Amazon S3 \(p. 1116\)](#).

For instructions on creating and testing a working sample, see [Using the AWS SDK for PHP and Running PHP Examples \(p. 1193\)](#).

```
require 'vendor/autoload.php';

use Aws\S3\S3Client;

$bucket = '*** Your Bucket Name ***';

$s3 = new S3Client([
    'version' => 'latest',
    'region'  => 'us-east-1'
]);


// Add the website configuration.
$s3->putBucketWebsite([
    'Bucket'           => $bucket,
    'WebsiteConfiguration' => [
        'IndexDocument' => ['Suffix' => 'index.html'],
        'ErrorDocument' => ['Key' => 'error.html']
    ]
]);
```

```
// Retrieve the website configuration.  
$result = $s3->getBucketWebsite([  
    'Bucket' => $bucket  
]);  
echo $result->getPath('IndexDocument/Suffix');  
  
// Delete the website configuration.  
$s3->deleteBucketWebsite([  
    'Bucket' => $bucket  
]);
```

Using the AWS CLI

For more information about using the AWS CLI to configure an S3 bucket as a static website, see [website in the AWS CLI Command Reference](#).

Next, you must configure your index document and set permissions. For information, see [Configuring an index document \(p. 1122\)](#) and [Setting permissions for website access \(p. 1126\)](#).

You can also optionally configure an [error document \(p. 1124\)](#), [web traffic logging \(p. 1130\)](#), or a [redirect \(p. 1130\)](#).

Configuring an index document

When you enable website hosting, you must also configure and upload an index document. An *index document* is a webpage that Amazon S3 returns when a request is made to the root of a website or any subfolder. For example, if a user enters `http://www.example.com` in the browser, the user is not requesting any specific page. In that case, Amazon S3 serves up the index document, which is sometimes referred to as the *default page*.

When you enable static website hosting for your bucket, you enter the name of the index document (for example, `index.html`). After you enable static website hosting for your bucket, you upload an HTML file with the index document name to your bucket.

The trailing slash at the root-level URL is optional. For example, if you configure your website with `index.html` as the index document, either of the following URLs returns `index.html`.

```
http://example-bucket.s3-website.Region.amazonaws.com/  
http://example-bucket.s3-website.Region.amazonaws.com
```

For more information about Amazon S3 website endpoints, see [Website endpoints \(p. 1116\)](#).

Index document and folders

In Amazon S3, a bucket is a flat container of objects. It does not provide any hierarchical organization as the file system on your computer does. However, you can create a logical hierarchy by using object key names that imply a folder structure.

For example, consider a bucket with three objects that have the following key names. Although these are stored with no physical hierarchical organization, you can infer the following logical folder structure from the key names:

- `sample1.jpg` — Object is at the root of the bucket.
- `photos/2006/Jan/sample2.jpg` — Object is in the `photos/2006/Jan` subfolder.
- `photos/2006/Feb/sample3.jpg` — Object is in the `photos/2006/Feb` subfolder.

In the Amazon S3 console, you can also create a folder in a bucket. For example, you can create a folder named photos. You can upload objects to the bucket or to the photos folder within the bucket. If you add the object sample.jpg to the bucket, the key name is sample.jpg. If you upload the object to the photos folder, the object key name is photos/sample.jpg.

If you create a folder structure in your bucket, you must have an index document at each level. In each folder, the index document must have the same name, for example, index.html. When a user specifies a URL that resembles a folder lookup, the presence or absence of a trailing slash determines the behavior of the website. For example, the following URL, with a trailing slash, returns the photos/index.html index document.

```
http://bucket-name.s3-website.Region.amazonaws.com/photos/
```

However, if you exclude the trailing slash from the preceding URL, Amazon S3 first looks for an object photos in the bucket. If the photos object is not found, it searches for an index document, photos/index.html. If that document is found, Amazon S3 returns a 302 Found message and points to the photos/ key. For subsequent requests to photos/, Amazon S3 returns photos/index.html. If the index document is not found, Amazon S3 returns an error.

Configure an index document

To configure an index document using the S3 console, use the following procedure. You can also configure an index document using the REST API, the AWS SDKs, the AWS CLI, or AWS CloudFormation.

When you enable static website hosting for your bucket, you enter the name of the index document (for example, **index.html**). After you enable static website hosting for the bucket, you upload an HTML file with this index document name to your bucket.

To configure the index document

1. Create an **index.html** file.

If you don't have an **index.html** file, you can use the following HTML to create one:

```
<html xmlns="http://www.w3.org/1999/xhtml" >
<head>
    <title>My Website Home Page</title>
</head>
<body>
    <h1>Welcome to my website</h1>
    <p>Now hosted on Amazon S3!</p>
</body>
</html>
```

2. Save the index file locally.

The index document file name must exactly match the index document name that you enter in the **Static website hosting** dialog box. The index document name is case sensitive. For example, if you enter **index.html** for the **Index document** name in the **Static website hosting** dialog box, your index document file name must also be **index.html** and not **Index.html**.

3. Sign in to the AWS Management Console and open the Amazon S3 console at <https://console.aws.amazon.com/s3/>.
4. In the **Buckets** list, choose the name of the bucket that you want to use to host a static website.
5. Enable static website hosting for your bucket, and enter the exact name of your index document (for example, **index.html**). For more information, see [Enabling website hosting \(p. 1118\)](#).

After enabling static website hosting, proceed to step 6.

6. To upload the index document to your bucket, do one of the following:
 - Drag and drop the index file into the console bucket listing.
 - Choose **Upload**, and follow the prompts to choose and upload the index file.

For step-by-step instructions, see [Uploading objects \(p. 158\)](#).

7. (Optional) Upload other website content to your bucket.

Next, you must set permissions for website access. For information, see [Setting permissions for website access \(p. 1126\)](#).

You can also optionally configure an [error document \(p. 1124\)](#), [web traffic logging \(p. 1130\)](#), or a [redirect \(p. 1130\)](#).

Configuring a custom error document

After you configure your bucket as a static website, when an error occurs, Amazon S3 returns an HTML error document. You can optionally configure your bucket with a custom error document so that Amazon S3 returns that document when an error occurs.

Note

Some browsers display their own error message when an error occurs, ignoring the error document that Amazon S3 returns. For example, when an HTTP 404 Not Found error occurs, Google Chrome might ignore the error document that Amazon S3 returns and display its own error.

Topics

- [Amazon S3 HTTP response codes \(p. 1124\)](#)
- [Configuring a custom error document \(p. 1125\)](#)

Amazon S3 HTTP response codes

The following table lists the subset of HTTP response codes that Amazon S3 returns when an error occurs.

HTTP error code	Description
301 Moved Permanently	When a user sends a request directly to the Amazon S3 website endpoint (<code>http://s3-website.<i>Region</i>.amazonaws.com/</code>), Amazon S3 returns a 301 Moved Permanently response and redirects those requests to <code>https://aws.amazon.com/s3/</code> .
302 Found	When Amazon S3 receives a request for a key <code>x</code> , <code>http://bucket-name.s3-website.<i>Region</i>.amazonaws.com/x</code> , without a trailing slash, it first looks for the object with the key name <code>x</code> . If the object is not found, Amazon S3 determines that the request is for subfolder <code>x</code> and redirects the request by adding a slash at the end, and returns 302 Found .
304 Not Modified	Amazon S3 uses request headers <code>If-Modified-Since</code> , <code>If-Unmodified-Since</code> , <code>If-Match</code> and/or <code>If-None-Match</code> to determine whether the requested object is same as the cached copy held by the client. If the object is the same, the website endpoint returns a 304 Not Modified response.

HTTP error code	Description
400 Malformed Request	The website endpoint responds with a 400 Malformed Request when a user attempts to access a bucket through the incorrect regional endpoint.
403 Forbidden	The website endpoint responds with a 403 Forbidden when a user request translates to an object that is not publicly readable. The object owner must make the object publicly readable using a bucket policy or an ACL.
404 Not Found	<p>The website endpoint responds with 404 Not Found for the following reasons:</p> <ul style="list-style-type: none"> Amazon S3 determines that the URL of the website refers to an object key that does not exist. Amazon S3 infers that the request is for an index document that does not exist. A bucket specified in the URL does not exist. A bucket specified in the URL exists, but isn't configured as a website. <p>You can create a custom document that is returned for 404 Not Found. Make sure that the document is uploaded to the bucket configured as a website, and that the website hosting configuration is set to use the document.</p> <p>For information on how Amazon S3 interprets the URL as a request for an object or an index document, see Configuring an index document (p. 1122).</p>
500 Service Error	The website endpoint responds with a 500 Service Error when an internal server error occurs.
503 Service Unavailable	The website endpoint responds with a 503 Service Unavailable when Amazon S3 determines that you need to reduce your request rate.

For each of these errors, Amazon S3 returns a predefined HTML message. The following is an example HTML message that is returned for a **403 Forbidden** response.

403 Forbidden

- Code: AccessDenied
- Message: Access Denied
- RequestId: 873CA367A51F7EC7
- HostId: DdQezl9vkuw5huD5HKsFaTDm9KH4PZzCPRkW3igimILbTu1DiYlvXjgyd7pVxq32

An Error Occurred While Attempting to Retrieve a Custom Error Document

- Code: AccessDenied
- Message: Access Denied

Configuring a custom error document

When you configure your bucket as a static website, you can provide a custom error document that contains a user-friendly error message and additional help. Amazon S3 returns your custom error document for only the HTTP 4XX class of error codes.

To configure a custom error document using the S3 console, follow the steps below. You can also configure an error document using the REST API, the AWS SDKs, the AWS CLI, or AWS CloudFormation. For more information, see the following:

- [PutBucketWebsite](#) in the *Amazon Simple Storage Service API Reference*
- [AWS::S3::Bucket WebsiteConfiguration](#) in the *AWS CloudFormation User Guide*
- [put-bucket-website](#) in the *AWS CLI Command Reference*

When you enable static website hosting for your bucket, you enter the name of the error document (for example, `404.html`). After you enable static website hosting for the bucket, you upload an HTML file with this error document name to your bucket.

To configure an error document

1. Create an error document, for example `404.html`.
2. Save the error document file locally.

The error document name is case sensitive and must exactly match the name that you enter when you enable static website hosting. For example, if you enter `404.html` for the **Error document** name in the **Static website hosting** dialog box, your error document file name must also be `404.html`.

3. Sign in to the AWS Management Console and open the Amazon S3 console at <https://console.aws.amazon.com/s3/>.
4. In the **Buckets** list, choose the name of the bucket that you want to use to host a static website.
5. Enable static website hosting for your bucket, and enter the exact name of your error document (for example, `404.html`). For more information, see [Enabling website hosting \(p. 1118\)](#).

After enabling static website hosting, proceed to step 6.

6. To upload the error document to your bucket, do one of the following:
 - Drag and drop the error document file into the console bucket listing.
 - Choose **Upload**, and follow the prompts to choose and upload the index file.

For step-by-step instructions, see [Uploading objects \(p. 158\)](#).

Setting permissions for website access

When you configure a bucket as a static website, if you want your website to be public, you can grant public read access. To make your bucket publicly readable, you must disable block public access settings for the bucket and write a bucket policy that grants public read access. If your bucket contains objects that are not owned by the bucket owner, you might also need to add an object access control list (ACL) that grants everyone read access.

If you don't want to disable block public access settings for your bucket but you still want your website to be public, you can create a Amazon CloudFront distribution to serve your static website. For more information, see [Use an Amazon CloudFront distribution to serve a static website](#) in the *Amazon Route 53 Developer Guide*.

Note

On the website endpoint, if a user requests an object that doesn't exist, Amazon S3 returns HTTP response code `404 (Not Found)`. If the object exists but you haven't granted read permission on it, the website endpoint returns HTTP response code `403 (Access Denied)`. The user can use the response code to infer whether a specific object exists. If you don't want this behavior, you should not enable website support for your bucket.

Topics

- [Step 1: Edit S3 Block Public Access settings \(p. 1127\)](#)
- [Step 2: Add a bucket policy \(p. 1128\)](#)
- [Object access control lists \(p. 1129\)](#)

Step 1: Edit S3 Block Public Access settings

If you want to configure an existing bucket as a static website that has public access, you must edit Block Public Access settings for that bucket. You might also have to edit your account-level Block Public Access settings. Amazon S3 applies the most restrictive combination of the bucket-level and account-level block public access settings.

For example, if you allow public access for a bucket but block all public access at the account level, Amazon S3 will continue to block public access to the bucket. In this scenario, you would have to edit your bucket-level and account-level Block Public Access settings. For more information, see [Blocking public access to your Amazon S3 storage \(p. 584\)](#).

By default, Amazon S3 blocks public access to your account and buckets. If you want to use a bucket to host a static website, you can use these steps to edit your block public access settings.

Warning

Before you complete this step, review [Blocking public access to your Amazon S3 storage \(p. 584\)](#) to ensure that you understand and accept the risks involved with allowing public access. When you turn off block public access settings to make your bucket public, anyone on the internet can access your bucket. We recommend that you block all public access to your buckets.

1. Open the Amazon S3 console at <https://console.aws.amazon.com/s3/>.
2. Choose the name of the bucket that you have configured as a static website.
3. Choose **Permissions**.
4. Under **Block public access (bucket settings)**, choose **Edit**.
5. Clear **Block all public access**, and choose **Save changes**.

Warning

Before you complete this step, review [Blocking public access to your Amazon S3 storage \(p. 584\)](#) to ensure you understand and accept the risks involved with allowing public access. When you turn off block public access settings to make your bucket public, anyone on the internet can access your bucket. We recommend that you block all public access to your buckets.

Block public access (bucket settings)

Public access is granted to buckets and objects through access control lists (ACLs), bucket policies, access point policies, or all. In order to ensure that public access to all your S3 buckets and objects is blocked, turn on Block all public access. These settings apply only to this bucket and its access points. AWS recommends that you turn on Block all public access, but before applying any of these settings, ensure that your applications will work correctly without public access. If you require some level of public access to your buckets or objects within, you can customize the individual settings below to suit your specific storage use cases. [Learn more](#)



Account settings for Block Public Access are currently turned on

[Account settings for Block Public Access](#) that are enabled apply even if they are disabled for this bucket.

Block all public access

Turning this setting on is the same as turning on all four settings below. Each of the following settings are independent of one another.

Block public access to buckets and objects granted through new access control lists (ACLs)

S3 will block public access permissions applied to newly added buckets or objects, and prevent the creation of new public access ACLs for existing buckets and objects. This setting doesn't change any existing permissions that allow public access to S3 resources using ACLs.

Block public access to buckets and objects granted through any access control lists (ACLs)

S3 will ignore all ACLs that grant public access to buckets and objects.

Block public access to buckets and objects granted through new public bucket or access point policies

S3 will block new bucket and access point policies that grant public access to buckets and objects. This setting doesn't change any existing policies that allow public access to S3 resources.

Block public and cross-account access to buckets and objects through any public bucket or access point policies

S3 will ignore public and cross-account access for buckets or access points with policies that grant public access to buckets and objects.

Amazon S3 turns off Block Public Access settings for your bucket. To create a public, static website, you might also have to [edit the Block Public Access settings](#) for your account before adding a bucket policy. If account settings for Block Public Access are currently turned on, you see a note under **Block public access (bucket settings)**.

Step 2: Add a bucket policy

To make the objects in your bucket publicly readable, you must write a bucket policy that grants everyone `s3:GetObject` permission.

After you edit S3 Block Public Access settings, you can add a bucket policy to grant public read access to your bucket. When you grant public read access, anyone on the internet can access your bucket.

Important

The following policy is an example only and allows full access to the contents of your bucket. Before you proceed with this step, review [How can I secure the files in my Amazon S3 bucket?](#) to ensure that you understand the best practices for securing the files in your S3 bucket and risks involved in granting public access.

1. Under **Buckets**, choose the name of your bucket.
2. Choose **Permissions**.
3. Under **Bucket Policy**, choose **Edit**.
4. To grant public read access for your website, copy the following bucket policy, and paste it in the **Bucket policy editor**.

```
{  
    "Version": "2012-10-17",  
    "Statement": [  
        {
```

```
        "Sid": "PublicReadGetObject",
        "Effect": "Allow",
        "Principal": "*",
        "Action": [
            "s3:GetObject"
        ],
        "Resource": [
            "arn:aws:s3:::Bucket-Name/*"
        ]
    }
}
```

5. Update the Resource to your bucket name.

In the preceding example bucket policy, *Bucket-Name* is a placeholder for the bucket name. To use this bucket policy with your own bucket, you must update this name to match your bucket name.

6. Choose **Save changes**.

A message appears indicating that the bucket policy has been successfully added.

If you see an error that says `Policy has invalid resource`, confirm that the bucket name in the bucket policy matches your bucket name. For information about adding a bucket policy, see [How do I add an S3 bucket policy?](#)

If you get an error message and cannot save the bucket policy, check your account and bucket Block Public Access settings to confirm that you allow public access to the bucket.

Object access control lists

You can use a bucket policy to grant public read permission to your objects. However, the bucket policy applies only to objects that are owned by the bucket owner. If your bucket contains objects that aren't owned by the bucket owner, the bucket owner should use the object access control list (ACL) to grant public READ permission on those objects.

By default, when another AWS account uploads an object to your S3 bucket, that account (the object writer) owns the object, has access to it, and can grant other users access to it through ACLs. You can use Object Ownership to change this default behavior so that ACLs are disabled and you, as the bucket owner, automatically own every object in your bucket. As a result, access control for your data is based on policies, such as IAM policies, S3 bucket policies, virtual private cloud (VPC) endpoint policies, and AWS Organizations service control policies (SCPs).

A majority of modern use cases in Amazon S3 no longer require the use of ACLs, and we recommend that you disable ACLs except in unusual circumstances where you need to control access for each object individually. With Object Ownership, you can disable ACLs and rely on policies for access control. When you disable ACLs, you can easily maintain a bucket with objects uploaded by different AWS accounts. You, as the bucket owner, own all the objects in the bucket and can manage access to them using policies. For more information, see [Controlling ownership of objects and disabling ACLs for your bucket \(p. 601\)](#).

Important

If your bucket uses the bucket owner enforced setting for S3 Object Ownership, you must use policies to grant access to your bucket and the objects in it. Requests to set ACLs or update ACLs fail and return the `AccessControlListNotSupported` error code. Requests to read ACLs are still supported.

To make an object publicly readable using an ACL, grant READ permission to the `AllUsers` group, as shown in the following grant element. Add this grant element to the object ACL. For information about managing ACLs, see [Access control list \(ACL\) overview \(p. 554\)](#).

```
<Grant>
<Grantee xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:type="Group">
    <URI>http://acs.amazonaws.com/groups/global/AllUsers</URI>
</Grantee>
<Permission>READ</Permission>
</Grant>
```

(Optional) Logging web traffic

You can optionally enable Amazon S3 server access logging for a bucket that is configured as a static website. Server access logging provides detailed records for the requests that are made to your bucket. For more information, see [Logging requests using server access logging \(p. 978\)](#). If you plan to use Amazon CloudFront to [speed up your website \(p. 109\)](#), you can also use CloudFront logging. For more information, see [Configuring and Using Access Logs in the Amazon CloudFront Developer Guide](#).

To enable server access logging for your static website bucket

1. Open the Amazon S3 console at <https://console.aws.amazon.com/s3/>.
2. In the same Region where you created the bucket that is configured as a static website, create a bucket for logging, for example logs.example.com.
3. Create a folder for the server access logging log files (for example, logs).

When you group your log data files in a folder, they are easier to locate.

4. (Optional) If you want to use CloudFront to improve your website performance, create a folder for the CloudFront log files (for example, cdn).

For more information, see [Speeding up your website with Amazon CloudFront \(p. 109\)](#).

5. In the **Buckets** list, choose your bucket.
6. Choose **Properties**.
7. Under **Server access logging**, choose **Edit**.
8. Choose **Enable**.
9. Under the **Target bucket**, choose the bucket and folder destination for the server access logs:
 - Browse to the folder and bucket location:
 1. Choose **Browse S3**.
 2. Choose the bucket name, and then choose the logs folder.
 3. Choose **Choose path**.
 - Enter the S3 bucket path, for example, `s3://logs.example.com/logs/`.
10. Choose **Save changes**.

In your log bucket, you can now access your logs. Amazon S3 writes website access logs to your log bucket every 2 hours.

(Optional) Configuring a webpage redirect

If your Amazon S3 bucket is configured for static website hosting, you can configure redirects for your bucket or the objects in it. You have the following options for configuring redirects.

Topics

- [Redirect requests for your bucket's website endpoint to another bucket or domain \(p. 1131\)](#)
- [Configure redirection rules to use advanced conditional redirects \(p. 1131\)](#)
- [Redirect requests for an object \(p. 1136\)](#)

Redirect requests for your bucket's website endpoint to another bucket or domain

You can redirect all requests to a website endpoint for a bucket to another bucket or domain. If you redirect all requests, any request made to the website endpoint is redirected to the specified bucket or domain.

For example, if your root domain is `example.com`, and you want to serve requests for both `http://example.com` and `http://www.example.com`, you must create two buckets named `example.com` and `www.example.com`. Then, maintain the content in the `example.com` bucket, and configure the other `www.example.com` bucket to redirect all requests to the `example.com` bucket. For more information, see [Configuring a Static Website Using a Custom Domain Name](#).

To redirect requests for a bucket website endpoint

1. Open the Amazon S3 console at <https://console.aws.amazon.com/s3/>.
2. Under **Buckets**, choose the name of the bucket that you want to redirect requests from (for example, `www.example.com`).
3. Choose **Properties**.
4. Under **Static website hosting**, choose **Edit**.
5. Choose **Redirect requests for an object**.
6. In the **Host name** box, enter the website endpoint for your bucket or your custom domain.

For example, if you are redirecting to a root domain address, you would enter `example.com`.
7. For **Protocol**, choose the protocol for the redirected requests (`none`, `http`, or `https`).

If you do not specify a protocol, the default option is `none`.
8. Choose **Save changes**.

Configure redirection rules to use advanced conditional redirects

Using advanced redirection rules, you can route requests conditionally according to specific object key names, prefixes in the request, or response codes. For example, suppose that you delete or rename an object in your bucket. You can add a routing rule that redirects the request to another object. If you want to make a folder unavailable, you can add a routing rule to redirect the request to another webpage. You can also add a routing rule to handle error conditions by routing requests that return the error to another domain when the error is processed.

When enabling static website hosting for your bucket, you can optionally specify advanced redirection rules. Amazon S3 has a limitation of 50 routing rules per website configuration. If you require more than 50 routing rules, you can use object redirect. For more information, see [Using the S3 console \(p. 1136\)](#).

For more information about configuring routing rules using the REST API, see [PutBucketWebsite](#) in the *Amazon Simple Storage Service API Reference*.

Important

To create redirection rules in the new Amazon S3 console, you must use JSON. For JSON examples, see [Redirection rules examples \(p. 1134\)](#).

To configure redirection rules for a static website

To add redirection rules for a bucket that already has static website hosting enabled, follow these steps.

1. Open the Amazon S3 console at <https://console.aws.amazon.com/s3/>.
2. In the **Buckets** list, choose the name of a bucket that you have configured as a static website.
3. Choose **Properties**.
4. Under **Static website hosting**, choose **Edit**.
5. In **Redirection rules** box, enter your redirection rules in JSON.

In the S3 console you describe the rules using JSON. For JSON examples, see [Redirection rules examples \(p. 1134\)](#). Amazon S3 has a limitation of 50 routing rules per website configuration.

6. Choose **Save changes**.

Routing rule elements

The following is general syntax for defining the routing rules in a website configuration in JSON and XML. To configure redirection rules in the new S3 console, you must use JSON. For JSON examples, see [Redirection rules examples \(p. 1134\)](#).

JSON

```
[  
  {  
    "Condition": {  
      "HttpErrorCodeReturnedEquals": "string",  
      "KeyPrefixEquals": "string"  
    },  
    "Redirect": {  
      "HostName": "string",  
      "HttpRedirectCode": "string",  
      "Protocol": "http"|"https",  
      "ReplaceKeyPrefixWith": "string",  
      "ReplaceKeyWith": "string"  
    }  
  }  
]
```

Note: Redirect must each have at least one child element. You can have either ReplaceKeyPrefixWith or ReplaceKeyWith but not both.

XML

```
<RoutingRules> =  
  <RoutingRules>  
    <RoutingRule>...</RoutingRule>  
    [<RoutingRule>...</RoutingRule>  
     ...]  
  </RoutingRules>  
  
<RoutingRule> =  
  <RoutingRule>  
    [<Condition>...</Condition> ]
```

```

        <Redirect>...</Redirect>
    </RoutingRule>

<Condition> =
    <Condition>
        [ <KeyPrefixEquals>...</KeyPrefixEquals> ]
        [ <HttpErrorCodeReturnedEquals>...</HttpErrorCodeReturnedEquals> ]
    </Condition>
    Note: <Condition> must have at least one child element.

<Redirect> =
    <Redirect>
        [ <HostName>...</HostName> ]
        [ <Protocol>...</Protocol> ]
        [ <ReplaceKeyPrefixWith>...</ReplaceKeyPrefixWith> ]
        [ <ReplaceKeyWith>...</ReplaceKeyWith> ]
        [ <HttpRedirectCode>...</HttpRedirectCode> ]
    </Redirect>

    Note: <Redirect> must have at least one child element. You can have either
          ReplaceKeyPrefix with or ReplaceKeyWith but not both.

```

The following table describes the elements in the routing rule.

Name	Description
RoutingRules	Container for a collection of RoutingRule elements.
RoutingRule	A rule that identifies a condition and the redirect that is applied when the condition is met. Condition: <ul style="list-style-type: none"> • A RoutingRules container must contain at least one routing rule.
Condition	Container for describing a condition that must be met for the specified redirect to be applied. If the routing rule does not include a condition, the rule is applied to all requests.
KeyPrefixEquals	The prefix of the object key name from which requests are redirected. KeyPrefixEquals is required if HttpErrorCodeReturnedEquals is not specified. If both KeyPrefixEquals and HttpErrorCodeReturnedEquals are specified, both must be true for the condition to be met.
HttpErrorCodeReturnedEquals	The HTTP error code that must match for the redirect to apply. If an error occurs, and if the error code meets this value, then the specified redirect applies. HttpErrorCodeReturnedEquals is required if KeyPrefixEquals is not specified. If both KeyPrefixEquals and HttpErrorCodeReturnedEquals are specified, both must be true for the condition to be met.
Redirect	Container element that provides instructions for redirecting the request. You can redirect requests to another host or another page, or you can specify another protocol to use. A RoutingRule must have a Redirect element. A Redirect element must contain at least one of the following sibling elements: Protocol,

Name	Description
	HostName, ReplaceKeyPrefixWith, ReplaceKeyWith, or HttpRedirectCode.
Protocol	The protocol, http or https, to be used in the Location header that is returned in the response. If one of its siblings is supplied, Protocol is not required.
HostName	The hostname to be used in the Location header that is returned in the response. If one of its siblings is supplied, HostName is not required.
ReplaceKeyPrefixWith	The prefix of the object key name that replaces the value of KeyPrefixEquals in the redirect request. If one of its siblings is supplied, ReplaceKeyPrefixWith is not required. It can be supplied only if ReplaceKeyWith is not supplied.
ReplaceKeyWith	The object key to be used in the Location header that is returned in the response. If one of its siblings is supplied, ReplaceKeyWith is not required. It can be supplied only if ReplaceKeyPrefixWith is not supplied.
HttpRedirectCode	The HTTP redirect code to be used in the Location header that is returned in the response. If one of its siblings is supplied, HttpRedirectCode is not required.

Redirection rules examples

The following examples explain common redirection tasks:

Important

To create redirection rules in the new Amazon S3 console, you must use JSON.

Example 1: Redirect after renaming a key prefix

Suppose that your bucket contains the following objects:

- index.html
- docs/article1.html
- docs/article2.html

You decide to rename the folder from docs/ to documents/. After you make this change, you need to redirect requests for prefix docs/ to documents/. For example, request for docs/article1.html will be redirected to documents/article1.html.

In this case, you add the following routing rule to the website configuration.

JSON

```
[  
  {  
    "Condition": {
```

```
        "KeyPrefixEquals": "docs/"
    },
    "Redirect": {
        "ReplaceKeyPrefixWith": "documents/"
    }
}
]
```

XML

```
<RoutingRules>
  <RoutingRule>
    <Condition>
      <KeyPrefixEquals>docs/</KeyPrefixEquals>
    </Condition>
    <Redirect>
      <ReplaceKeyPrefixWith>documents/</ReplaceKeyPrefixWith>
    </Redirect>
  </RoutingRule>
</RoutingRules>
```

Example 2: Redirect requests for a deleted folder to a page

Suppose that you delete the `images/` folder (that is, you delete all objects with the key prefix `images/`). You can add a routing rule that redirects requests for any object with the key prefix `images/` to a page named `folderdeleted.html`.

JSON

```
[
  {
    "Condition": {
      "KeyPrefixEquals": "images/"
    },
    "Redirect": {
      "ReplaceKeyWith": "folderdeleted.html"
    }
}
]
```

XML

```
<RoutingRules>
  <RoutingRule>
    <Condition>
      <KeyPrefixEquals>images/</KeyPrefixEquals>
    </Condition>
    <Redirect>
      <ReplaceKeyWith>folderdeleted.html</ReplaceKeyWith>
    </Redirect>
  </RoutingRule>
</RoutingRules>
```

Example 3: Redirect for an HTTP error

Suppose that when a requested object is not found, you want to redirect requests to an Amazon Elastic Compute Cloud (Amazon EC2) instance. Add a redirection rule so that when an HTTP status code 404 (Not Found) is returned, the site visitor is redirected to an Amazon EC2 instance that handles the request.

The following example also inserts the object key prefix `report-404/` in the redirect. For example, if you request a page `ExamplePage.html` and it results in an HTTP 404 error, the request is redirected to a page `report-404/ExamplePage.html` on the specified Amazon EC2 instance. If there is no routing rule and the HTTP error 404 occurs, the error document that is specified in the configuration is returned.

JSON

```
[  
  {  
    "Condition": {  
      "HttpErrorCodeReturnedEquals": "404"  
    },  
    "Redirect": {  
      "HostName": "ec2-11-22-33-44.compute-1.amazonaws.com",  
      "ReplaceKeyPrefixWith": "report-404/"  
    }  
  }  
]
```

XML

```
<RoutingRules>  
  <RoutingRule>  
    <Condition>  
      <HttpErrorCodeReturnedEquals>404</HttpErrorCodeReturnedEquals>  
    </Condition>  
    <Redirect>  
      <HostName>ec2-11-22-33-44.compute-1.amazonaws.com</HostName>  
      <ReplaceKeyPrefixWith>report-404/</ReplaceKeyPrefixWith>  
    </Redirect>  
  </RoutingRule>  
</RoutingRules>
```

Redirect requests for an object

You can redirect requests for an object to another object or URL by setting the website redirect location in the metadata of the object. You set the redirect by adding the `x-amz-website-redirect-location` property to the object metadata. On the Amazon S3 console, you set the **Website Redirect Location** in the metadata of the object. If you use the [Amazon S3 API \(p. 1137\)](#), you set `x-amz-website-redirect-location`. The website then interprets the object as a 301 redirect.

To redirect a request to another object, you set the redirect location to the key of the target object. To redirect a request to an external URL, you set the redirect location to the URL that you want. For more information about object metadata, see [System-defined object metadata \(p. 154\)](#).

When you set a page redirect, you can either keep or delete the source object content. For example, if you have a `page1.html` object in your bucket, you can redirect any requests for this page to another object, `page2.html`. You have two options:

- Keep the content of the `page1.html` object and redirect page requests.
- Delete the content of `page1.html` and upload a zero-byte object named `page1.html` to replace the existing object and redirect page requests.

Using the S3 console

1. Open the Amazon S3 console at <https://console.aws.amazon.com/s3/>.

2. In the **Buckets** list, choose the name of the bucket that you have configured as a static website (for example, `example.com`).
 3. Under **Objects**, select your object.
 4. Choose **Actions**, and choose **Edit metadata**.
 5. Choose **Metadata**.
 6. Choose **Add Metadata**.
 7. Under **Type**, choose **System Defined**.
 8. In **Key**, choose **x-amz-website-redirect-location**.
 9. In **Value**, enter the key name of the object that you want to redirect to, for example, `/page2.html`.
- For another object in the same bucket, the `/` prefix in the value is required. You can also set the value to an external URL, for example, `http://www.example.com`.
10. Choose **Edit metadata**.

Using the REST API

The following Amazon S3 API actions support the `x-amz-website-redirect-location` header in the request. Amazon S3 stores the header value in the object metadata as `x-amz-website-redirect-location`.

- [PUT Object](#)
- [Initiate Multipart Upload](#)
- [POST Object](#)
- [PUT Object - Copy](#)

A bucket configured for website hosting has both the website endpoint and the REST endpoint. A request for a page that is configured as a 301 redirect has the following possible outcomes, depending on the endpoint of the request:

- **Region-specific website endpoint** – Amazon S3 redirects the page request according to the value of the `x-amz-website-redirect-location` property.
- **REST endpoint** – Amazon S3 doesn't redirect the page request. It returns the requested object.

For more information about the endpoints, see [Key differences between a website endpoint and a REST API endpoint \(p. 1118\)](#).

When setting a page redirect, you can either keep or delete the object content. For example, suppose that you have a `page1.html` object in your bucket.

- To keep the content of `page1.html` and only redirect page requests, you can submit a [PUT Object - Copy](#) request to create a new `page1.html` object that uses the existing `page1.html` object as the source. In your request, you set the `x-amz-website-redirect-location` header. When the request is complete, you have the original page with its content unchanged, but Amazon S3 redirects any requests for the page to the redirect location that you specify.
- To delete the content of the `page1.html` object and redirect requests for the page, you can send a PUT Object request to upload a zero-byte object that has the same object key: `page1.html`. In the PUT request, you set `x-amz-website-redirect-location` for `page1.html` to the new object. When the request is complete, `page1.html` has no content, and requests are redirected to the location that is specified by `x-amz-website-redirect-location`.

When you retrieve the object using the [GET Object](#) action, along with other object metadata, Amazon S3 returns the `x-amz-website-redirect-location` header in the response.

Developing with Amazon S3

This section covers developer-related topics for using Amazon S3. For more information, review the topics below.

Topics

- [Making requests \(p. 1138\)](#)
- [Developing with Amazon S3 using the AWS CLI \(p. 1183\)](#)
- [Developing with Amazon S3 using the AWS SDKs, and explorers \(p. 1184\)](#)
- [Developing with Amazon S3 using the REST API \(p. 1197\)](#)
- [Handling REST and SOAP errors \(p. 1201\)](#)
- [Developer reference \(p. 1204\)](#)

Making requests

Amazon S3 is a REST service. You can send requests to Amazon S3 using the REST API or the AWS SDK (see [Sample Code and Libraries](#)) wrapper libraries that wrap the underlying Amazon S3 REST API, simplifying your programming tasks.

Every interaction with Amazon S3 is either authenticated or anonymous. Authentication is a process of verifying the identity of the requester trying to access an Amazon Web Services (AWS) product. Authenticated requests must include a signature value that authenticates the request sender. The signature value is, in part, generated from the requester's AWS access keys (access key ID and secret access key). For more information about getting access keys, see [How Do I Get Security Credentials?](#) in the [AWS General Reference](#).

If you are using the AWS SDK, the libraries compute the signature from the keys you provide. However, if you make direct REST API calls in your application, you must write the code to compute the signature and add it to the request.

Topics

- [About access keys \(p. 1138\)](#)
- [Request endpoints \(p. 1140\)](#)
- [Making requests to Amazon S3 over IPv6 \(p. 1140\)](#)
- [Making requests using the AWS SDKs \(p. 1147\)](#)
- [Making requests using the REST API \(p. 1174\)](#)

About access keys

The following sections review the types of access keys that you can use to make authenticated requests.

AWS account access keys

The account access keys provide full access to the AWS resources owned by the account. The following are examples of access keys:

- Access key ID (a 20-character, alphanumeric string). For example: AKIAIOSFODNN7EXAMPLE
- Secret access key (a 40-character string). For example: wJalrXUtnFEMI/K7MDENG/bPxRfiCYEXAMPLEKEY

The access key ID uniquely identifies an AWS account. You can use these access keys to send authenticated requests to Amazon S3.

IAM user access keys

You can create one AWS account for your company; however, there may be several employees in the organization who need access to your organization's AWS resources. Sharing your AWS account access keys reduces security, and creating individual AWS accounts for each employee might not be practical. Also, you cannot easily share resources such as buckets and objects because they are owned by different accounts. To share resources, you must grant permissions, which is additional work.

In such scenarios, you can use AWS Identity and Access Management (IAM) to create users under your AWS account with their own access keys and attach IAM user policies granting appropriate resource access permissions to them. To better manage these users, IAM enables you to create groups of users and grant group-level permissions that apply to all users in that group.

These users are referred to as IAM users that you create and manage within AWS. The parent account controls a user's ability to access AWS. Any resources an IAM user creates are under the control of and paid for by the parent AWS account. These IAM users can send authenticated requests to Amazon S3 using their own security credentials. For more information about creating and managing users under your AWS account, go to the [AWS Identity and Access Management product details page](#).

Temporary security credentials

In addition to creating IAM users with their own access keys, IAM also enables you to grant temporary security credentials (temporary access keys and a security token) to any IAM user to enable them to access your AWS services and resources. You can also manage users in your system outside AWS. These are referred to as federated users. Additionally, users can be applications that you create to access your AWS resources.

IAM provides the AWS Security Token Service API for you to request temporary security credentials. You can use either the AWS STS API or the AWS SDK to request these credentials. The API returns temporary security credentials (access key ID and secret access key), and a security token. These credentials are valid only for the duration you specify when you request them. You use the access key ID and secret key the same way you use them when sending requests using your AWS account or IAM user access keys. In addition, you must include the token in each request you send to Amazon S3.

An IAM user can request these temporary security credentials for their own use or hand them out to federated users or applications. When requesting temporary security credentials for federated users, you must provide a user name and an IAM policy defining the permissions you want to associate with these temporary security credentials. The federated user cannot get more permissions than the parent IAM user who requested the temporary credentials.

You can use these temporary security credentials in making requests to Amazon S3. The API libraries compute the necessary signature value using those credentials to authenticate your request. If you send requests using expired credentials, Amazon S3 denies the request.

For information on signing requests using temporary security credentials in your REST API requests, see [Signing and authenticating REST requests \(p. 1210\)](#). For information about sending requests using AWS SDKs, see [Making requests using the AWS SDKs \(p. 1147\)](#).

For more information about IAM support for temporary security credentials, see [Temporary Security Credentials](#) in the *IAM User Guide*.

For added security, you can require multifactor authentication (MFA) when accessing your Amazon S3 resources by configuring a bucket policy. For information, see [Adding a bucket policy to require MFA \(p. 495\)](#). After you require MFA to access your Amazon S3 resources, the only way you can access these resources is by providing temporary credentials that are created with an MFA key. For more information, see the [AWS Multi-Factor Authentication](#) detail page and [Configuring MFA-Protected API Access](#) in the *IAM User Guide*.

Request endpoints

You send REST requests to the service's predefined endpoint. For a list of all AWS services and their corresponding endpoints, go to [Regions and Endpoints](#) in the *AWS General Reference*.

Making requests to Amazon S3 over IPv6

Amazon Simple Storage Service (Amazon S3) supports the ability to access S3 buckets using the Internet Protocol version 6 (IPv6), in addition to the IPv4 protocol. Amazon S3 dual-stack endpoints support requests to S3 buckets over IPv6 and IPv4. There are no additional charges for accessing Amazon S3 over IPv6. For more information about pricing, see [Amazon S3 Pricing](#).

Topics

- [Getting started making requests over IPv6 \(p. 1140\)](#)
- [Using IPv6 addresses in IAM policies \(p. 1141\)](#)
- [Testing IP address compatibility \(p. 1142\)](#)
- [Using Amazon S3 dual-stack endpoints \(p. 1142\)](#)

Getting started making requests over IPv6

To make a request to an S3 bucket over IPv6, you need to use a dual-stack endpoint. The next section describes how to make requests over IPv6 by using dual-stack endpoints.

The following are some things you should know before trying to access a bucket over IPv6:

- The client and the network accessing the bucket must be enabled to use IPv6.
- Both virtual hosted-style and path style requests are supported for IPv6 access. For more information, see [Amazon S3 dual-stack endpoints \(p. 1142\)](#).
- If you use source IP address filtering in your AWS Identity and Access Management (IAM) user or bucket policies, you need to update the policies to include IPv6 address ranges. For more information, see [Using IPv6 addresses in IAM policies \(p. 1141\)](#).
- When using IPv6, server access log files output IP addresses in an IPv6 format. You need to update existing tools, scripts, and software that you use to parse Amazon S3 log files so that they can parse the IPv6 formatted Remote IP addresses. For more information, see [Amazon S3 server access log format \(p. 989\)](#) and [Logging requests using server access logging \(p. 978\)](#).

Note

If you experience issues related to the presence of IPv6 addresses in log files, contact [AWS Support](#).

Making requests over IPv6 by using dual-stack endpoints

You make requests with Amazon S3 API calls over IPv6 by using dual-stack endpoints. The Amazon S3 API operations work the same way whether you're accessing Amazon S3 over IPv6 or over IPv4. Performance should be the same too.

When using the REST API, you access a dual-stack endpoint directly. For more information, see [Dual-stack endpoints \(p. 1142\)](#).

When using the AWS Command Line Interface (AWS CLI) and AWS SDKs, you can use a parameter or flag to change to a dual-stack endpoint. You can also specify the dual-stack endpoint directly as an override of the Amazon S3 endpoint in the config file.

You can use a dual-stack endpoint to access a bucket over IPv6 from any of the following:

- The AWS CLI, see [Using dual-stack endpoints from the AWS CLI \(p. 1143\)](#).
- The AWS SDKs, see [Using dual-stack endpoints from the AWS SDKs \(p. 1144\)](#).
- The REST API, see [Making requests to dual-stack endpoints by using the REST API \(p. 1175\)](#).

Features not available over IPv6

The following feature is currently not supported when accessing an S3 bucket over IPv6: Static website hosting from an S3 bucket.

Using IPv6 addresses in IAM policies

Before trying to access a bucket using IPv6, you must ensure that any IAM user or S3 bucket policies that are used for IP address filtering are updated to include IPv6 address ranges. IP address filtering policies that are not updated to handle IPv6 addresses may result in clients incorrectly losing or gaining access to the bucket when they start using IPv6. For more information about managing access permissions with IAM, see [Identity and access management in Amazon S3 \(p. 394\)](#).

IAM policies that filter IP addresses use [IP Address Condition Operators](#). The following bucket policy identifies the 54.240.143.* range of allowed IPv4 addresses by using IP address condition operators. Any IP addresses outside of this range will be denied access to the bucket (`examplebucket`). Since all IPv6 addresses are outside of the allowed range, this policy prevents IPv6 addresses from being able to access `examplebucket`.

```
{  
    "Version": "2012-10-17",  
    "Statement": [  
        {  
            "Sid": "IPAllow",  
            "Effect": "Allow",  
            "Principal": "*",  
            "Action": "s3:*",  
            "Resource": "arn:aws:s3:::examplebucket/*",  
            "Condition": {  
                "IpAddress": {"aws:SourceIp": "54.240.143.0/24"}  
            }  
        }  
    ]  
}
```

You can modify the bucket policy's `Condition` element to allow both IPv4 (54.240.143.0/24) and IPv6 (2001:DB8:1234:5678::/64) address ranges as shown in the following example. You can use the same type of `Condition` block shown in the example to update both your IAM user and bucket policies.

```
"Condition": {  
    "IpAddress": {  
        "aws:SourceIp": [  
            "54.240.143.0/24",  
            "2001:DB8:1234:5678::/64"  
        ]  
    }  
}
```

```
}
```

Before using IPv6 you must update all relevant IAM user and bucket policies that use IP address filtering to allow IPv6 address ranges. We recommend that you update your IAM policies with your organization's IPv6 address ranges in addition to your existing IPv4 address ranges. For an example of a bucket policy that allows access over both IPv6 and IPv4, see [Limiting access to specific IP addresses \(p. 493\)](#).

You can review your IAM user policies using the IAM console at <https://console.aws.amazon.com/iam/>. For more information about IAM, see the [IAM User Guide](#). For information about editing S3 bucket policies, see [Adding a bucket policy using the Amazon S3 console \(p. 488\)](#).

Testing IP address compatibility

If you are using Linux/Unix or Mac OS X, you can test whether you can access a dual-stack endpoint over IPv6 by using the curl command as shown in the following example:

Example

```
curl -v http://s3.dualstack.us-west-2.amazonaws.com/
```

You get back information similar to the following example. If you are connected over IPv6 the connected IP address will be an IPv6 address.

```
* About to connect() to s3-us-west-2.amazonaws.com port 80 (#0)
*   Trying IPv6 address... connected
* Connected to s3.dualstack.us-west-2.amazonaws.com (IPv6 address) port 80 (#0)
> GET / HTTP/1.1
> User-Agent: curl/7.18.1 (x86_64-unknown-linux-gnu) libcurl/7.18.1 OpenSSL/1.0.1t
  zlib/1.2.3
> Host: s3.dualstack.us-west-2.amazonaws.com
```

If you are using Microsoft Windows 7 or Windows 10, you can test whether you can access a dual-stack endpoint over IPv6 or IPv4 by using the ping command as shown in the following example.

```
ping ipv6.s3.dualstack.us-west-2.amazonaws.com
```

Using Amazon S3 dual-stack endpoints

Amazon S3 dual-stack endpoints support requests to S3 buckets over IPv6 and IPv4. This section describes how to use dual-stack endpoints.

Topics

- [Amazon S3 dual-stack endpoints \(p. 1142\)](#)
- [Using dual-stack endpoints from the AWS CLI \(p. 1143\)](#)
- [Using dual-stack endpoints from the AWS SDKs \(p. 1144\)](#)
- [Using dual-stack endpoints from the REST API \(p. 1146\)](#)

Amazon S3 dual-stack endpoints

When you make a request to a dual-stack endpoint, the bucket URL resolves to an IPv6 or an IPv4 address. For more information about accessing a bucket over IPv6, see [Making requests to Amazon S3 over IPv6 \(p. 1140\)](#).

When using the REST API, you directly access an Amazon S3 endpoint by using the endpoint name (URI). You can access an S3 bucket through a dual-stack endpoint by using a virtual hosted-style or a path-style endpoint name. Amazon S3 supports only regional dual-stack endpoint names, which means that you must specify the region as part of the name.

Use the following naming conventions for the dual-stack virtual hosted-style and path-style endpoint names:

- Virtual hosted-style dual-stack endpoint:

`bucketname.s3.dualstack.aws-region.amazonaws.com`

- Path-style dual-stack endpoint:

`s3.dualstack.aws-region.amazonaws.com/bucketname`

For more information about endpoint name style, see [Methods for accessing a bucket \(p. 125\)](#). For a list of Amazon S3 endpoints, see [Regions and Endpoints](#) in the *AWS General Reference*.

Important

You can use transfer acceleration with dual-stack endpoints. For more information, see [Getting started with Amazon S3 Transfer Acceleration \(p. 137\)](#).

When using the AWS Command Line Interface (AWS CLI) and AWS SDKs, you can use a parameter or flag to change to a dual-stack endpoint. You can also specify the dual-stack endpoint directly as an override of the Amazon S3 endpoint in the config file. The following sections describe how to use dual-stack endpoints from the AWS CLI and the AWS SDKs.

Using dual-stack endpoints from the AWS CLI

This section provides examples of AWS CLI commands used to make requests to a dual-stack endpoint. For instructions on setting up the AWS CLI, see [Developing with Amazon S3 using the AWS CLI \(p. 1183\)](#).

You set the configuration value `use_dualstack_endpoint` to `true` in a profile in your AWS Config file to direct all Amazon S3 requests made by the `s3` and `s3api` AWS CLI commands to the dual-stack endpoint for the specified region. You specify the region in the config file or in a command using the `--region` option.

When using dual-stack endpoints with the AWS CLI, both path and virtual addressing styles are supported. The addressing style, set in the config file, controls if the bucket name is in the hostname or part of the URL. By default, the CLI will attempt to use virtual style where possible, but will fall back to path style if necessary. For more information, see [AWS CLI Amazon S3 Configuration](#).

You can also make configuration changes by using a command, as shown in the following example, which sets `use_dualstack_endpoint` to `true` and `addressing_style` to `virtual` in the default profile.

```
$ aws configure set default.s3.use_dualstack_endpoint true
$ aws configure set default.s3.addressing_style virtual
```

If you want to use a dual-stack endpoint for specified AWS CLI commands only (not all commands), you can use either of the following methods:

- You can use the dual-stack endpoint per command by setting the `--endpoint-url` parameter to `https://s3.dualstack.aws-region.amazonaws.com` or `http://s3.dualstack.aws-region.amazonaws.com` for any `s3` or `s3api` command.

```
$ aws s3api list-objects --bucket bucketname --endpoint-url https://s3.dualstack.aws-region.amazonaws.com
```

- You can set up separate profiles in your AWS Config file. For example, create one profile that sets `use_dualstack_endpoint` to `true` and a profile that does not set `use_dualstack_endpoint`. When you run a command, specify which profile you want to use, depending upon whether or not you want to use the dual-stack endpoint.

Note

When using the AWS CLI you currently cannot use transfer acceleration with dual-stack endpoints. However, support for the AWS CLI is coming soon. For more information, see [Using the AWS CLI \(p. 139\)](#).

Using dual-stack endpoints from the AWS SDKs

This section provides examples of how to access a dual-stack endpoint by using the AWS SDKs.

AWS SDK for Java dual-stack endpoint example

The following example shows how to enable dual-stack endpoints when creating an Amazon S3 client using the AWS SDK for Java.

For instructions on creating and testing a working Java sample, see [Testing the Amazon S3 Java Code Examples \(p. 1192\)](#).

```
import com.amazonaws.AmazonServiceException;
import com.amazonaws.SdkClientException;
import com.amazonaws.auth.profile.ProfileCredentialsProvider;
import com.amazonaws.regions.Regions;
import com.amazonaws.services.s3.AmazonS3;
import com.amazonaws.services.s3.AmazonS3ClientBuilder;

public class DualStackEndpoints {

    public static void main(String[] args) {
        Regions clientRegion = Regions.DEFAULT_REGION;
        String bucketName = "*** Bucket name ***";

        try {
            // Create an Amazon S3 client with dual-stack endpoints enabled.
            AmazonS3 s3Client = AmazonS3ClientBuilder.standard()
                .withCredentials(new ProfileCredentialsProvider())
                .withRegion(clientRegion)
                .withDualstackEnabled(true)
                .build();

            s3Client.listObjects(bucketName);
        } catch (AmazonServiceException e) {
            // The call was transmitted successfully, but Amazon S3 couldn't process
            // it, so it returned an error response.
            e.printStackTrace();
        } catch (SdkClientException e) {
            // Amazon S3 couldn't be contacted for a response, or the client
            // couldn't parse the response from Amazon S3.
            e.printStackTrace();
        }
    }
}
```

If you are using the AWS SDK for Java on Windows, you might have to set the following Java virtual machine (JVM) property:

```
java.net.preferIPv6Addresses=true
```

AWS .NET SDK dual-stack endpoint example

When using the AWS SDK for .NET you use the `AmazonS3Config` class to enable the use of a dual-stack endpoint as shown in the following example.

```
using Amazon;
using Amazon.S3;
using Amazon.S3.Model;
using System;
using System.Threading.Tasks;

namespace Amazon.DocSamples.S3
{
    class DualStackEndpointTest
    {
        private const string bucketName = "*** bucket name ***";
        // Specify your bucket region (an example region is shown).
        private static readonly RegionEndpoint bucketRegion = RegionEndpoint.USWest2;
        private static IAmazonS3 client;

        public static void Main()
        {
            var config = new AmazonS3Config
            {
                UseDualstackEndpoint = true,
                RegionEndpoint = bucketRegion
            };
            client = new AmazonS3Client(config);
            Console.WriteLine("Listing objects stored in a bucket");
            ListingObjectsAsync().Wait();
        }

        private static async Task ListingObjectsAsync()
        {
            try
            {
                var request = new ListObjectsV2Request
                {
                    BucketName = bucketName,
                    MaxKeys = 10
                };
                ListObjectsV2Response response;
                do
                {
                    response = await client.ListObjectsV2Async(request);

                    // Process the response.
                    foreach (S3Object entry in response.S3Objects)
                    {
                        Console.WriteLine("key = {0} size = {1}",
                            entry.Key, entry.Size);
                    }
                    Console.WriteLine("Next Continuation Token: {0}",
                        response.NextContinuationToken);
                    request.ContinuationToken = response.NextContinuationToken;
                } while (response.IsTruncated == true);
            }
            catch (AmazonS3Exception amazonS3Exception)
            {
```

```
        Console.WriteLine("An AmazonS3Exception was thrown. Exception: " +
amazonS3Exception.ToString());
    }
    catch (Exception e)
    {
        Console.WriteLine("Exception: " + e.ToString());
    }
}
}
```

For a full .NET sample for listing objects, see [Listing object keys programmatically \(p. 245\)](#).

For information about how to create and test a working .NET sample, see [Running the Amazon S3 .NET Code Examples \(p. 1193\)](#).

Using dual-stack endpoints from the REST API

For information about making requests to dual-stack endpoints by using the REST API, see [Making requests to dual-stack endpoints by using the REST API \(p. 1175\)](#).

Making requests using the AWS SDKs

Topics

- [Making requests using AWS account or IAM user credentials \(p. 1147\)](#)
- [Making requests using IAM user temporary credentials \(p. 1155\)](#)
- [Making requests using federated user temporary credentials \(p. 1164\)](#)

You can send authenticated requests to Amazon S3 using either the AWS SDK or by making the REST API calls directly in your application. The AWS SDK API uses the credentials that you provide to compute the signature for authentication. If you use the REST API directly in your applications, you must write the necessary code to compute the signature for authenticating your request. For a list of available AWS SDKs go to, [Sample Code and Libraries](#).

Making requests using AWS account or IAM user credentials

You can use your AWS account or IAM user security credentials to send authenticated requests to Amazon S3. This section provides examples of how you can send authenticated requests using the AWS SDK for Java, AWS SDK for .NET, and AWS SDK for PHP. For a list of available AWS SDKs, go to [Sample Code and Libraries](#).

Each of these AWS SDKs uses an SDK-specific credentials provider chain to find and use credentials and perform actions on behalf of the credentials owner. What all these credentials provider chains have in common is that they all look for your local AWS credentials file.

For more information, see the topics below:

Topics

- [To create a local AWS credentials file \(p. 1147\)](#)
- [Sending authenticated requests using the AWS SDKs \(p. 1148\)](#)
- [Related resources \(p. 1154\)](#)

To create a local AWS credentials file

The easiest way to configure credentials for your AWS SDKs is to use an AWS credentials file. If you use the AWS Command Line Interface (AWS CLI), you may already have a local AWS credentials file configured. Otherwise, use the following procedure to set up a credentials file:

1. Sign in to the AWS Management Console and open the IAM console at <https://console.aws.amazon.com/iam/>.
2. Create a new user with permissions limited to the services and actions that you want your code to have access to. For more information about creating a new IAM user, see [Creating IAM Users \(Console\)](#), and follow the instructions through step 8.
3. Choose **Download .csv** to save a local copy of your AWS credentials.
4. On your computer, navigate to your home directory, and create an `.aws` directory. On Unix-based systems, such as Linux or OS X, this is in the following location:

```
~/.aws
```

On Windows, this is in the following location:

```
%HOMEPATH%\aws
```

5. In the `.aws` directory, create a new file named `credentials`.

6. Open the credentials .csv file that you downloaded from the IAM console, and copy its contents into the `credentials` file using the following format:

```
[default]
aws_access_key_id = your_access_key_id
aws_secret_access_key = your_secret_access_key
```

7. Save the `credentials` file, and delete the .csv file that you downloaded in step 3.

Your shared credentials file is now configured on your local computer, and it's ready to be used with the AWS SDKs.

Sending authenticated requests using the AWS SDKs

Use the AWS SDKs to send authenticated requests.

Java

To send authenticated requests to Amazon S3 using your AWS account or IAM user credentials, do the following:

- Use the `AmazonS3ClientBuilder` class to create an `AmazonS3Client` instance.
- Run one of the `AmazonS3Client` methods to send requests to Amazon S3. The client generates the necessary signature from the credentials that you provide and includes it in the request.

The following example performs the preceding tasks. For information on creating and testing a working sample, see [Testing the Amazon S3 Java Code Examples \(p. 1192\)](#).

Example

```
import com.amazonaws.AmazonServiceException;
import com.amazonaws.SdkClientException;
import com.amazonaws.auth.profile.ProfileCredentialsProvider;
import com.amazonaws.regions.Regions;
import com.amazonaws.services.s3.AmazonS3;
import com.amazonaws.services.s3.AmazonS3ClientBuilder;
import com.amazonaws.services.s3.model.ListObjectsRequest;
import com.amazonaws.services.s3.model.ObjectListing;
import com.amazonaws.services.s3.model.S3ObjectSummary;

import java.io.IOException;
import java.util.List;

public class MakingRequests {

    public static void main(String[] args) throws IOException {
        Regions clientRegion = Regions.DEFAULT_REGION;
        String bucketName = "*** Bucket name ***";

        try {
            AmazonS3 s3Client = AmazonS3ClientBuilder.standard()
                .withCredentials(new ProfileCredentialsProvider())
                .withRegion(clientRegion)
                .build();

            // Get a list of objects in the bucket, two at a time, and
            // print the name and size of each object.
            ListObjectsRequest listRequest = new
            ListObjectsRequest().withBucketName(bucketName).withMaxKeys(2);
```

```
ObjectListing objects = s3Client.listObjects(listRequest);
while (true) {
    List<S3ObjectSummary> summaries = objects.getObjectSummaries();
    for (S3ObjectSummary summary : summaries) {
        System.out.printf("Object \"%s\" retrieved with size %d\n",
summary.getKey(), summary.getSize());
    }
    if (objects.isTruncated()) {
        objects = s3Client.listNextBatchOfObjects(objects);
    } else {
        break;
    }
}
} catch (AmazonServiceException e) {
// The call was transmitted successfully, but Amazon S3 couldn't process
// it, so it returned an error response.
e.printStackTrace();
} catch (SdkClientException e) {
// Amazon S3 couldn't be contacted for a response, or the client
// couldn't parse the response from Amazon S3.
e.printStackTrace();
}
}
```

.NET

To send authenticated requests using your AWS account or IAM user credentials:

- Create an instance of the `AmazonS3Client` class.
- Run one of the `AmazonS3Client` methods to send requests to Amazon S3. The client generates the necessary signature from the credentials that you provide and includes it in the request it sends to Amazon S3.

For more information, see [Making requests using AWS account or IAM user credentials \(p. 1147\)](#).

Note

- You can create the `AmazonS3Client` client without providing your security credentials. Requests sent using this client are anonymous requests, without a signature. Amazon S3 returns an error if you send anonymous requests for a resource that is not publicly available.
- You can create an AWS account and create the required user accounts. You can also manage credentials for those user accounts. You need these credentials to perform the task in the following example. For more information, see [Configure AWS credentials](#) in the *AWS SDK for .NET Developer Guide*.

You can then also configure your application to actively retrieve profiles and credentials, and then explicitly use those credentials when creating an AWS service client. For more information, see [Accessing credentials and profiles in an application](#) in the *AWS SDK for .NET Developer Guide*.

The following C# example shows how to perform the preceding tasks. For information about running the .NET examples in this guide and for instructions on how to store your credentials in a configuration file, see [Running the Amazon S3 .NET Code Examples \(p. 1193\)](#).

Example

```
using Amazon;
```

```
using Amazon.S3;
using Amazon.S3.Model;
using System;
using System.Threading.Tasks;

namespace Amazon.DocSamples.S3
{
    class MakeS3RequestTest
    {
        private const string bucketName = "*** bucket name ***";
        // Specify your bucket region (an example region is shown).
        private static readonly RegionEndpoint bucketRegion = RegionEndpoint.USWest2;
        private static IAmazonS3 client;

        public static void Main()
        {
            using (client = new AmazonS3Client(bucketRegion))
            {
                Console.WriteLine("Listing objects stored in a bucket");
                ListingObjectsAsync().Wait();
            }
        }

        static async Task ListingObjectsAsync()
        {
            try
            {
                ListObjectsRequest request = new ListObjectsRequest
                {
                    BucketName = bucketName,
                    MaxKeys = 2
                };
                do
                {
                    ListObjectsResponse response = await
client.ListObjectsAsync(request);
                    // Process the response.
                    foreach (S3Object entry in response.S3Objects)
                    {
                        Console.WriteLine("key = {0} size = {1}",
                            entry.Key, entry.Size);
                    }

                    // If the response is truncated, set the marker to get the next
                    // set of keys.
                    if (response.IsTruncated)
                    {
                        request.Marker = response.NextMarker;
                    }
                    else
                    {
                        request = null;
                    }
                } while (request != null);
            }
            catch (AmazonS3Exception e)
            {
                Console.WriteLine("Error encountered on server. Message:'{0}' when
writing an object", e.Message);
            }
            catch (Exception e)
            {
                Console.WriteLine("Unknown encountered on server. Message:'{0}' when
writing an object", e.Message);
            }
        }
    }
}
```

```
    }  
}
```

For working examples, see [Amazon S3 objects overview \(p. 149\)](#) and [Buckets overview \(p. 114\)](#). You can test these examples using your AWS account or an IAM user credentials.

For example, to list all the object keys in your bucket, see [Listing object keys programmatically \(p. 245\)](#).

PHP

This section explains how to use a class from version 3 of the AWS SDK for PHP to send authenticated requests using your AWS account or IAM user credentials. It assumes that you are already following the instructions for [Using the AWS SDK for PHP and Running PHP Examples \(p. 1193\)](#) and have the AWS SDK for PHP properly installed.

The following PHP example shows how the client makes a request using your security credentials to list all of the buckets for your account.

Example

```
require 'vendor/autoload.php';  
  
use Aws\Sts\StsClient;  
use Aws\S3\S3Client;  
use Aws\S3\Exception\S3Exception;  
  
$bucket = '*** Your Bucket Name ***';  
  
$s3 = new S3Client([  
    'region' => 'us-east-1',  
    'version' => 'latest',  
]);  
  
// Retrieve the list of buckets.  
$result = $s3->listBuckets();  
  
try {  
    // Retrieve a paginator for listing objects.  
    $objects = $s3->getPaginator('ListObjects', [  
        'Bucket' => $bucket  
    ]);  
  
    echo "Keys retrieved!" . PHP_EOL;  
  
    // Print the list of objects to the page.  
    foreach ($objects as $object) {  
        echo $object['Key'] . PHP_EOL;  
    }  
} catch (S3Exception $e) {  
    echo $e->getMessage() . PHP_EOL;  
}
```

Note

You can create the `S3Client` client without providing your security credentials. Requests sent using this client are anonymous requests, without a signature. Amazon S3 returns an error if you send anonymous requests for a resource that is not publicly available. For more information, see [Creating Anonymous Clients in the AWS SDK for PHP Documentation](#).

For working examples, see [Amazon S3 objects overview \(p. 149\)](#). You can test these examples using your AWS account or IAM user credentials.

For an example of listing object keys in a bucket, see [Listing object keys programmatically \(p. 245\)](#).

Ruby

Before you can use version 3 of the AWS SDK for Ruby to make calls to Amazon S3, you must set the AWS access credentials that the SDK uses to verify your access to your buckets and objects. If you have shared credentials set up in the AWS credentials profile on your local system, version 3 of the SDK for Ruby can use those credentials without your having to declare them in your code. For more information about setting up shared credentials, see [Making requests using AWS account or IAM user credentials \(p. 1147\)](#).

The following Ruby code snippet uses the credentials in a shared AWS credentials file on a local computer to authenticate a request to get all of the object key names in a specific bucket. It does the following:

1. Creates an instance of the `Aws::S3::Client` class.
2. Makes a request to Amazon S3 by enumerating objects in a bucket using the `list_objects_v2` method of `Aws::S3::Client`. The client generates the necessary signature value from the credentials in the AWS credentials file on your computer, and includes it in the request it sends to Amazon S3.
3. Prints the array of object key names to the terminal.

Example

```
# Prerequisites:
# - An existing Amazon S3 bucket.

require "aws-sdk-s3"

# @param s3_client [Aws::S3::Client] An initialized Amazon S3 client.
# @param bucket_name [String] The bucket's name.
# @return [Boolean] true if all operations succeed; otherwise, false.
# @example
#   s3_client = Aws::S3::Client.new(region: 'us-west-2')
#   exit 1 unless list_bucket_objects?(s3_client, 'doc-example-bucket')
def list_bucket_objects?(s3_client, bucket_name)
    puts "Accessing the bucket named '#{bucket_name}'..."
    objects = s3_client.list_objects_v2(
        bucket: bucket_name,
        max_keys: 50
    )

    if objects.count.positive?
        puts "The object keys in this bucket are (first 50 objects):"
        objects.contents.each do |object|
            puts object.key
        end
    else
        puts "No objects found in this bucket."
    end

    return true
rescue StandardError => e
    puts "Error while accessing the bucket named '#{bucket_name}': #{e.message}"
    return false
end

# Full example call:
# Replace us-west-2 with the AWS Region you're using for Amazon S3.
def run_me
    region = "us-west-2"
```

```
bucket_name = "BUCKET_NAME"
s3_client = Aws::S3::Client.new(region: region)

exit 1 unless list_bucket_objects?(s3_client, bucket_name)
end

run_me if $PROGRAM_NAME == __FILE__
```

If you don't have a local AWS credentials file, you can still create the `Aws::S3::Client` resource and run code against Amazon S3 buckets and objects. Requests that are sent using version 3 of the SDK for Ruby are anonymous, with no signature by default. Amazon S3 returns an error if you send anonymous requests for a resource that's not publicly available.

You can use and expand the previous code snippet for SDK for Ruby applications, as in the following more robust example.

```
# Prerequisites:
# - An existing Amazon S3 bucket.

require "aws-sdk-s3"

# @param s3_client [Aws::S3::Client] An initialized Amazon S3 client.
# @param bucket_name [String] The bucket's name.
# @return [Boolean] true if all operations succeed; otherwise, false.
# @example
#   s3_client = Aws::S3::Client.new(region: 'us-west-2')
#   exit 1 unless list_bucket_objects?(s3_client, 'doc-example-bucket')
def list_bucket_objects?(s3_client, bucket_name)
    puts "Accessing the bucket named '#{bucket_name}'..."
    objects = s3_client.list_objects_v2(
        bucket: bucket_name,
        max_keys: 50
    )

    if objects.count.positive?
        puts "The object keys in this bucket are (first 50 objects):"
        objects.contents.each do |object|
            puts object.key
        end
    else
        puts "No objects found in this bucket."
    end

    return true
rescue StandardError => e
    puts "Error while accessing the bucket named '#{bucket_name}': #{e.message}"
    return false
end

# Full example call:
# Replace us-west-2 with the AWS Region you're using for Amazon S3.
def run_me
    region = "us-west-2"
    bucket_name = "BUCKET_NAME"
    s3_client = Aws::S3::Client.new(region: region)

    exit 1 unless list_bucket_objects?(s3_client, bucket_name)
end

run_me if $PROGRAM_NAME == __FILE__
```

Related resources

- [Developing with Amazon S3 using the AWS SDKs, and explorers \(p. 1184\)](#)
- [AWS SDK for PHP for Amazon S3 Aws\S3\S3Client Class](#)
- [AWS SDK for PHP Documentation](#)

Making requests using IAM user temporary credentials

An AWS account or an IAM user can request temporary security credentials and use them to send authenticated requests to Amazon S3. This section provides examples of how to use the AWS SDK for Java, .NET, and PHP to obtain temporary security credentials and use them to authenticate your requests to Amazon S3.

Java

An IAM user or an AWS account can request temporary security credentials (see [Making requests \(p. 1138\)](#)) using the AWS SDK for Java and use them to access Amazon S3. These credentials expire after the specified session duration.

By default, the session duration is one hour. If you use IAM user credentials, you can specify the duration when requesting the temporary security credentials from 15 minutes to the maximum session duration for the role. For more information about temporary security credentials, see [Temporary Security Credentials](#) in the *IAM User Guide*. For more information about making requests, see [Making requests \(p. 1138\)](#).

To get temporary security credentials and access Amazon S3

1. Create an instance of the `AWSSecurityTokenService` class. For information about providing credentials, see [Developing with Amazon S3 using the AWS SDKs, and explorers \(p. 1184\)](#).
2. Retrieve the temporary security credentials for the desired role by calling the `assumeRole()` method of the Security Token Service (STS) client.
3. Package the temporary security credentials into a `BasicSessionCredentials` object. You use this object to provide the temporary security credentials to your Amazon S3 client.
4. Create an instance of the `AmazonS3Client` class using the temporary security credentials. You send requests to Amazon S3 using this client. If you send requests using expired credentials, Amazon S3 will return an error.

Note

If you obtain temporary security credentials using your AWS account security credentials, the temporary credentials are valid for only one hour. You can specify the session duration only if you use IAM user credentials to request a session.

The following example lists a set of object keys in the specified bucket. The example obtains temporary security credentials for a session and uses them to send an authenticated request to Amazon S3.

If you want to test the sample using IAM user credentials, you will need to create an IAM user under your AWS account. For more information about how to create an IAM user, see [Creating Your First IAM User and Administrators Group](#) in the *IAM User Guide*.

For instructions on creating and testing a working sample, see [Testing the Amazon S3 Java Code Examples \(p. 1192\)](#).

```
import com.amazonaws.AmazonServiceException;
import com.amazonaws.SdkClientException;
import com.amazonaws.auth.AWSStaticCredentialsProvider;
import com.amazonaws.auth.BasicSessionCredentials;
import com.amazonaws.auth.profile.ProfileCredentialsProvider;
import com.amazonaws.services.s3.AmazonS3;
import com.amazonaws.services.s3.AmazonS3ClientBuilder;
import com.amazonaws.services.s3.model.ObjectListing;
import com.amazonaws.services.securitytoken.AWSecurityTokenService;
```

```
import com.amazonaws.services.securitytoken.AWSSecurityTokenServiceClientBuilder;
import com.amazonaws.services.securitytoken.model.AssumeRoleRequest;
import com.amazonaws.services.securitytoken.model.AssumeRoleResult;
import com.amazonaws.services.securitytoken.model.Credentials;

public class MakingRequestsWithIAMTempCredentials {
    public static void main(String[] args) {
        String clientRegion = "*** Client region ***";
        String roleARN = "*** ARN for role to be assumed ***";
        String roleSessionName = "*** Role session name ***";
        String bucketName = "*** Bucket name ***";

        try {
            // Creating the STS client is part of your trusted code. It has
            // the security credentials you use to obtain temporary security
            credentials.
            AWSSecurityTokenService stsClient =
            AWSSecurityTokenServiceClientBuilder.standard()
                .withCredentials(new
            ProfileCredentialsProvider())
                .withRegion(clientRegion)
                .build();

            // Obtain credentials for the IAM role. Note that you cannot assume the
            role of an AWS root account;
            // Amazon S3 will deny access. You must use credentials for an IAM user or
            an IAM role.
            AssumeRoleRequest roleRequest = new AssumeRoleRequest()
                .withRoleArn(roleARN)

            .withRoleSessionName(roleSessionName);
            AssumeRoleResult roleResponse = stsClient.assumeRole(roleRequest);
            Credentials sessionCredentials = roleResponse.getCredentials();

            // Create a BasicSessionCredentials object that contains the credentials
            you just retrieved.
            BasicSessionCredentials awsCredentials = new BasicSessionCredentials(
                sessionCredentials.getAccessKeyId(),
                sessionCredentials.getSecretAccessKey(),
                sessionCredentials.getSessionToken());

            // Provide temporary security credentials so that the Amazon S3 client
            // can send authenticated requests to Amazon S3. You create the client
            // using the sessionCredentials object.
            AmazonS3 s3Client = AmazonS3ClientBuilder.standard()
                .withCredentials(new
            AWSStaticCredentialsProvider(awsCredentials))
                .withRegion(clientRegion)
                .build();

            // Verify that assuming the role worked and the permissions are set
            correctly
            // by getting a set of object keys from the bucket.
            ObjectListing objects = s3Client.listObjects(bucketName);
            System.out.println("No. of Objects: " +
            objects.getObjectSummaries().size());
        }
        catch(AmazonServiceException e) {
            // The call was transmitted successfully, but Amazon S3 couldn't process
            // it, so it returned an error response.
            e.printStackTrace();
        }
        catch(SdkClientException e) {
            // Amazon S3 couldn't be contacted for a response, or the client
            // couldn't parse the response from Amazon S3.
            e.printStackTrace();
        }
    }
}
```

```
        }  
    }  
}
```

.NET

An IAM user or an AWS account can request temporary security credentials using the AWS SDK for .NET and use them to access Amazon S3. These credentials expire after the session duration.

By default, the session duration is one hour. If you use IAM user credentials, you can specify the duration when requesting the temporary security credentials from 15 minutes to the maximum session duration for the role. For more information about temporary security credentials, see [Temporary Security Credentials](#) in the *IAM User Guide*. For more information about making requests, see [Making requests \(p. 1138\)](#).

To get temporary security credentials and access Amazon S3

1. Create an instance of the AWS Security Token Service client, `AmazonSecurityTokenServiceClient`. For information about providing credentials, see [Developing with Amazon S3 using the AWS SDKs, and explorers \(p. 1184\)](#).
2. Start a session by calling the `GetSessionToken` method of the STS client you created in the preceding step. You provide session information to this method using a `GetSessionTokenRequest` object.

The method returns your temporary security credentials.
3. Package the temporary security credentials in an instance of the `SessionAWSCredentials` object. You use this object to provide the temporary security credentials to your Amazon S3 client.
4. Create an instance of the `AmazonS3Client` class by passing in the temporary security credentials. You send requests to Amazon S3 using this client. If you send requests using expired credentials, Amazon S3 returns an error.

Note

If you obtain temporary security credentials using your AWS account security credentials, those credentials are valid for only one hour. You can specify a session duration only if you use IAM user credentials to request a session.

The following C# example lists object keys in the specified bucket. For illustration, the example obtains temporary security credentials for a default one-hour session and uses them to send authenticated request to Amazon S3.

If you want to test the sample using IAM user credentials, you need to create an IAM user under your AWS account. For more information about how to create an IAM user, see [Creating Your First IAM User and Administrators Group](#) in the *IAM User Guide*. For more information about making requests, see [Making requests \(p. 1138\)](#).

For instructions on creating and testing a working example, see [Running the Amazon S3 .NET Code Examples \(p. 1193\)](#).

```
using Amazon;  
using Amazon.Runtime;  
using Amazon.S3;  
using Amazon.S3.Model;  
using Amazon.SecurityToken;  
using Amazon.SecurityToken.Model;  
using System;  
using System.Collections.Generic;
```

```
using System.Threading.Tasks;

namespace Amazon.DocSamples.S3
{
    class TempCredExplicitSessionStartTest
    {
        private const string bucketName = "*** bucket name ***";
        // Specify your bucket region (an example region is shown).
        private static readonly RegionEndpoint bucketRegion = RegionEndpoint.USWest2;
        private static IAmazonS3 s3Client;
        public static void Main()
        {
            ListObjectsAsync().Wait();
        }

        private static async Task ListObjectsAsync()
        {
            try
            {
                // Credentials use the default AWS SDK for .NET credential search
                chain.
                // On local development machines, this is your default profile.
                Console.WriteLine("Listing objects stored in a bucket");
                SessionAWSCredentials tempCredentials = await
GetTemporaryCredentialsAsync();

                // Create a client by providing temporary security credentials.
                using (s3Client = new AmazonS3Client(tempCredentials, bucketRegion))
                {
                    var listObjectRequest = new ListObjectsRequest
                    {
                        BucketName = bucketName
                    };
                    // Send request to Amazon S3.
                    ListObjectsResponse response = await
s3Client.ListObjectsAsync(listObjectRequest);
                    List<S3Object> objects = response.S3Objects;
                    Console.WriteLine("Object count = {0}", objects.Count);
                }
            }
            catch (AmazonS3Exception s3Exception)
            {
                Console.WriteLine(s3Exception.Message, s3Exception.InnerException);
            }
            catch (AmazonSecurityTokenServiceException stsException)
            {
                Console.WriteLine(stsException.Message, stsException.InnerException);
            }
        }

        private static async Task<SessionAWSCredentials> GetTemporaryCredentialsAsync()
        {
            using (var stsClient = new AmazonSecurityTokenServiceClient())
            {
                var getSessionTokenRequest = new GetSessionTokenRequest
                {
                    DurationSeconds = 7200 // seconds
                };

                GetSessionTokenResponse sessionTokenResponse =
                    await
stsClient.GetSessionTokenAsync(getSessionTokenRequest);

                Credentials credentials = sessionTokenResponse.Credentials;

                var sessionCredentials =

```

```
        new SessionAWSCredentials(credentials.AccessKeyId,
                                 credentials.SecretAccessKey,
                                 credentials.SessionToken);
    return sessionCredentials;
}
}
}
```

PHP

This example assumes that you are already following the instructions for [Using the AWS SDK for PHP and Running PHP Examples \(p. 1193\)](#) and have the AWS SDK for PHP properly installed.

An IAM user or an AWS account can request temporary security credentials using version 3 of the AWS SDK for PHP. It can then use the temporary credentials to access Amazon S3. The credentials expire when the session duration expires.

By default, the session duration is one hour. If you use IAM user credentials, you can specify the duration when requesting the temporary security credentials from 15 minutes to the maximum session duration for the role. For more information about temporary security credentials, see [Temporary Security Credentials](#) in the *IAM User Guide*. For more information about making requests, see [Making requests \(p. 1138\)](#).

Note

If you obtain temporary security credentials using your AWS account security credentials, the temporary security credentials are valid for only one hour. You can specify the session duration only if you use IAM user credentials to request a session.

Example

The following PHP example lists object keys in the specified bucket using temporary security credentials. The example obtains temporary security credentials for a default one-hour session, and uses them to send authenticated request to Amazon S3. For information about running the PHP examples in this guide, see [Running PHP Examples \(p. 1194\)](#).

If you want to test the example using IAM user credentials, you need to create an IAM user under your AWS account. For information about how to create an IAM user, see [Creating Your First IAM User and Administrators Group](#) in the *IAM User Guide*. For examples of setting the session duration when using IAM user credentials to request a session, see [Making requests using IAM user temporary credentials \(p. 1155\)](#).

```
require 'vendor/autoload.php';

use Aws\Sts\StsClient;
use Aws\S3\S3Client;
use Aws\S3\Exception\S3Exception;

$bucket = '*** Your Bucket Name ***';

$sts = new StsClient([
    'version' => 'latest',
    'region' => 'us-east-1'
]);

$sessionToken = $sts->getSessionToken();

$s3 = new S3Client([
    'region' => 'us-east-1',
    'version' => 'latest',
    'credentials' => [
        'key'      => $sessionToken['Credentials']['AccessKeyId'],
        'secret'   => $sessionToken['Credentials']['SecretAccessKey'],
        'token'    => $sessionToken['Credentials']['SessionToken']
    ]
]);

$objects = $s3->listObjects(['Bucket' => $bucket]);
foreach ($objects['Contents'] as $object) {
    echo $object['Key'];
}
```

```

        'secret' => $sessionToken['Credentials']['SecretAccessKey'],
        'token'  => $sessionToken['Credentials']['SessionToken']
    ];
});

$result = $s3->listBuckets();

try {
    // Retrieve a paginator for listing objects.
    $objects = $s3->getPaginator('ListObjects', [
        'Bucket' => $bucket
    ]);

    echo "Keys retrieved!" . PHP_EOL;

    // List objects
    foreach ($objects as $object) {
        echo $object['Key'] . PHP_EOL;
    }
} catch (S3Exception $e) {
    echo $e->getMessage() . PHP_EOL;
}

```

Ruby

An IAM user or an AWS account can request temporary security credentials using AWS SDK for Ruby and use them to access Amazon S3. These credentials expire after the session duration.

By default, the session duration is one hour. If you use IAM user credentials, you can specify the duration when requesting the temporary security credentials from 15 minutes to the maximum session duration for the role. For more information about temporary security credentials, see [Temporary Security Credentials](#) in the *IAM User Guide*. For more information about making requests, see [Making requests \(p. 1138\)](#).

Note

If you obtain temporary security credentials using your AWS account security credentials, the temporary security credentials are valid for only one hour. You can specify session duration only if you use IAM user credentials to request a session.

The following Ruby example creates a temporary user to list the items in a specified bucket for one hour. To use this example, you must have AWS credentials that have the necessary permissions to create new AWS Security Token Service (AWS STS) clients, and list Amazon S3 buckets.

```

# Prerequisites:
# - A user in AWS Identity and Access Management (IAM). This user must
#   be able to assume the following IAM role. You must run this code example
#   within the context of this user.
# - An existing role in IAM that allows all of the Amazon S3 actions for all of the
#   resources in this code example. This role must also trust the preceding IAM user.
# - An existing S3 bucket.

require "aws-sdk-core"
require "aws-sdk-s3"
require "aws-sdk-iam"

# Checks whether a user exists in IAM.
#
# @param iam [Aws::IAM::Client] An initialized IAM client.
# @param user_name [String] The user's name.
# @return [Boolean] true if the user exists; otherwise, false.
# @example

```

```

#   iam_client = Aws::IAM::Client.new(region: 'us-west-2')
#   exit 1 unless user_exists?(iam_client, 'my-user')
def user_exists?(iam_client, user_name)
  response = iam_client.get_user(user_name: user_name)
  return true if response.user.user_name
rescue Aws::IAM::Errors::NoSuchEntity
  # User doesn't exist.
rescue StandardError => e
  puts "Error while determining whether the user " \
    "'#{user_name}' exists: #{e.message}"
end

# Creates a user in IAM.
#
# @param iam_client [Aws::IAM::Client] An initialized IAM client.
# @param user_name [String] The user's name.
# @return [AWS::IAM::Types::User] The new user.
# @example
#   iam_client = Aws::IAM::Client.new(region: 'us-west-2')
#   user = create_user(iam_client, 'my-user')
#   exit 1 unless user.user_name
def create_user(iam_client, user_name)
  response = iam_client.create_user(user_name: user_name)
  return response.user
rescue StandardError => e
  puts "Error while creating the user '#{user_name}': #{e.message}"
end

# Gets a user in IAM.
#
# @param iam_client [Aws::IAM::Client] An initialized IAM client.
# @param user_name [String] The user's name.
# @return [AWS::IAM::Types::User] The existing user.
# @example
#   iam_client = Aws::IAM::Client.new(region: 'us-west-2')
#   user = get_user(iam_client, 'my-user')
#   exit 1 unless user.user_name
def get_user(iam_client, user_name)
  response = iam_client.get_user(user_name: user_name)
  return response.user
rescue StandardError => e
  puts "Error while getting the user '#{user_name}': #{e.message}"
end

# Checks whether a role exists in IAM.
#
# @param iam_client [Aws::IAM::Client] An initialized IAM client.
# @param role_name [String] The role's name.
# @return [Boolean] true if the role exists; otherwise, false.
# @example
#   iam_client = Aws::IAM::Client.new(region: 'us-west-2')
#   exit 1 unless role_exists?(iam_client, 'my-role')
def role_exists?(iam_client, role_name)
  response = iam_client.get_role(role_name: role_name)
  return true if response.role.role_name
rescue StandardError => e
  puts "Error while determining whether the role " \
    "'#{role_name}' exists: #{e.message}"
end

# Gets credentials for a role in IAM.
#
# @param sts_client [Aws::STS::Client] An initialized AWS STS client.
# @param role_arn [String] The role's Amazon Resource Name (ARN).
# @param role_session_name [String] A name for this role's session.
# @param duration_seconds [Integer] The number of seconds this session is valid.

```

```

# @return [AWS::AssumeRoleCredentials] The credentials.
# @example
#   sts_client = Aws::STS::Client.new(region: 'us-west-2')
#   credentials = get_credentials(
#     sts_client,
#     'arn:aws:iam::123456789012:role/AmazonS3ReadOnly',
#     'ReadAmazonS3Bucket',
#     3600
#   )
#   exit 1 if credentials.nil?
def get_credentials(sts_client, role_arn, role_session_name, duration_seconds)
  Aws::AssumeRoleCredentials.new(
    client: sts_client,
    role_arn: role_arn,
    role_session_name: role_session_name,
    duration_seconds: duration_seconds
  )
rescue StandardError => e
  puts "Error while getting credentials: #{e.message}"
end

# Checks whether a bucket exists in Amazon S3.
#
# @param s3_client [Aws::S3::Client] An initialized Amazon S3 client.
# @param bucket_name [String] The name of the bucket.
# @return [Boolean] true if the bucket exists; otherwise, false.
# @example
#   s3_client = Aws::S3::Client.new(region: 'us-west-2')
#   exit 1 unless bucket_exists?(s3_client, 'doc-example-bucket')
def bucket_exists?(s3_client, bucket_name)
  response = s3_client.list_buckets
  response.buckets.each do |bucket|
    return true if bucket.name == bucket_name
  end
rescue StandardError => e
  puts "Error while checking whether the bucket '#{bucket_name}' " \
    "exists: #{e.message}"
end

# Lists the keys and ETags for the objects in an Amazon S3 bucket.
#
# @param s3_client [Aws::S3::Client] An initialized Amazon S3 client.
# @param bucket_name [String] The bucket's name.
# @return [Boolean] true if the objects were listed; otherwise, false.
# @example
#   s3_client = Aws::S3::Client.new(region: 'us-west-2')
#   exit 1 unless list_objects_in_bucket?(s3_client, 'doc-example-bucket')
def list_objects_in_bucket?(s3_client, bucket_name)
  puts "Accessing the contents of the bucket named '#{bucket_name}'..."
  response = s3_client.list_objects_v2(
    bucket: bucket_name,
    max_keys: 50
  )

  if response.count.positive?
    puts "Contents of the bucket named '#{bucket_name}' (first 50 objects):"
    puts "Name => ETag"
    response.contents.each do |obj|
      puts "#{obj.key} => #{obj.etag}"
    end
  else
    puts "No objects in the bucket named '#{bucket_name}'."
  end
  return true
rescue StandardError => e
  puts "Error while accessing the bucket named '#{bucket_name}': #{e.message}"

```

end

Related resources

- [Developing with Amazon S3 using the AWS SDKs, and explorers \(p. 1184\)](#)
- [AWS SDK for PHP for Amazon S3 Aws\S3\S3Client Class](#)
- [AWS SDK for PHP Documentation](#)

Making requests using federated user temporary credentials

You can request temporary security credentials and provide them to your federated users or applications who need to access your AWS resources. This section provides examples of how you can use the AWS SDK to obtain temporary security credentials for your federated users or applications and send authenticated requests to Amazon S3 using those credentials. For a list of available AWS SDKs, see [Sample Code and Libraries](#).

Note

Both the AWS account and an IAM user can request temporary security credentials for federated users. However, for added security, only an IAM user with the necessary permissions should request these temporary credentials to ensure that the federated user gets at most the permissions of the requesting IAM user. In some applications, you might find it suitable to create an IAM user with specific permissions for the sole purpose of granting temporary security credentials to your federated users and applications.

Java

You can provide temporary security credentials for your federated users and applications so that they can send authenticated requests to access your AWS resources. When requesting these temporary credentials, you must provide a user name and an IAM policy that describes the resource permissions that you want to grant. By default, the session duration is one hour. You can explicitly set a different duration value when requesting the temporary security credentials for federated users and applications.

Note

For added security when requesting temporary security credentials for federated users and applications, we recommend that you use a dedicated IAM user with only the necessary access permissions. The temporary user you create can never get more permissions than the IAM user who requested the temporary security credentials. For more information, see [AWS Identity and Access Management FAQs](#).

To provide security credentials and send authenticated request to access resources, do the following:

- Create an instance of the `AWSSecurityTokenServiceClient` class. For information about providing credentials, see [Using the AWS SDK for Java \(p. 1191\)](#).
- Start a session by calling the `getFederationToken()` method of the Security Token Service (STS) client. Provide session information, including the user name and an IAM policy, that you want to attach to the temporary credentials. You can provide an optional session duration. This method returns your temporary security credentials.
- Package the temporary security credentials in an instance of the `BasicSessionCredentials` object. You use this object to provide the temporary security credentials to your Amazon S3 client.
- Create an instance of the `AmazonS3Client` class using the temporary security credentials. You send requests to Amazon S3 using this client. If you send requests using expired credentials, Amazon S3 returns an error.

Example

The example lists keys in the specified S3 bucket. In the example, you obtain temporary security credentials for a two-hour session for your federated user and use the credentials to send authenticated requests to Amazon S3. To run the example, you need to create an IAM user with an attached policy that allows the user to request temporary security credentials and list your AWS resources. The following policy accomplishes this:

```
{  
    "Statement": [ {
```

```
        "Action": ["s3:ListBucket",
            "sts:GetFederationToken*"
        ],
        "Effect": "Allow",
        "Resource": "*"
    }
]
```

For more information about how to create an IAM user, see [Creating Your First IAM User and Administrators Group](#) in the *IAM User Guide*.

After creating an IAM user and attaching the preceding policy, you can run the following example. For instructions on creating and testing a working sample, see [Testing the Amazon S3 Java Code Examples \(p. 1192\)](#).

```
import com.amazonaws.AmazonServiceException;
import com.amazonaws.SdkClientException;
import com.amazonaws.auth.AWSStaticCredentialsProvider;
import com.amazonaws.auth.BasicSessionCredentials;
import com.amazonaws.auth.policy.Policy;
import com.amazonaws.auth.policy.Resource;
import com.amazonaws.auth.policy.Statement;
import com.amazonaws.auth.policy.Statement.Effect;
import com.amazonaws.auth.policy.actions.S3Actions;
import com.amazonaws.auth.profile.ProfileCredentialsProvider;
import com.amazonaws.regions.Regions;
import com.amazonaws.services.s3.AmazonS3;
import com.amazonaws.services.s3.AmazonS3ClientBuilder;
import com.amazonaws.services.s3.model.ObjectListing;
import com.amazonaws.services.securitytoken.AWSecurityTokenService;
import com.amazonaws.services.securitytoken.AWSecurityTokenServiceClientBuilder;
import com.amazonaws.services.securitytoken.model.Credentials;
import com.amazonaws.services.securitytoken.model.GetFederationTokenRequest;
import com.amazonaws.services.securitytoken.model.GetFederationTokenResult;

import java.io.IOException;

public class MakingRequestsWithFederatedTempCredentials {

    public static void main(String[] args) throws IOException {
        Regions clientRegion = Regions.DEFAULT_REGION;
        String bucketName = "*** Specify bucket name ***";
        String federatedUser = "*** Federated user name ***";
        String resourceARN = "arn:aws:s3:::" + bucketName;

        try {
            AWSecurityTokenService stsClient = AWSecurityTokenServiceClientBuilder
                .standard()
                .withCredentials(new ProfileCredentialsProvider())
                .withRegion(clientRegion)
                .build();

            GetFederationTokenRequest getFederationTokenRequest = new
GetFederationTokenRequest();
            getFederationTokenRequest.setDurationSeconds(7200);
            getFederationTokenRequest.setName(federatedUser);

            // Define the policy and add it to the request.
            Policy policy = new Policy();
            policy.withStatements(new Statement(Effect.Allow)
                .withActions(S3Actions.ListObjects)
                .withResources(new Resource(resourceARN))));


```

```
getFederationTokenRequest.setPolicy(policy.toJson());

// Get the temporary security credentials.
GetFederationTokenResult federationTokenResult =
stsClient.getFederationToken(getFederationTokenRequest);
Credentials sessionCredentials = federationTokenResult.getCredentials();

// Package the session credentials as a BasicSessionCredentials
// object for an Amazon S3 client object to use.
BasicSessionCredentials basicSessionCredentials = new
BasicSessionCredentials(
    sessionCredentials.getAccessKeyId(),
    sessionCredentials.getSecretAccessKey(),
    sessionCredentials.getSessionToken());
AmazonS3 s3Client = AmazonS3ClientBuilder.standard()
    .withCredentials(new
AWSStaticCredentialsProvider(basicSessionCredentials))
    .withRegion(clientRegion)
    .build();

// To verify that the client works, send a listObjects request using
// the temporary security credentials.
ObjectListing objects = s3Client.listObjects(bucketName);
System.out.println("No. of Objects = " +
objects.getObjectSummaries().size());
} catch (AmazonServiceException e) {
    // The call was transmitted successfully, but Amazon S3 couldn't process
    // it, so it returned an error response.
    e.printStackTrace();
} catch (SdkClientException e) {
    // Amazon S3 couldn't be contacted for a response, or the client
    // couldn't parse the response from Amazon S3.
    e.printStackTrace();
}
}
}
```

.NET

You can provide temporary security credentials for your federated users and applications so that they can send authenticated requests to access your AWS resources. When requesting these temporary credentials, you must provide a user name and an IAM policy that describes the resource permissions that you want to grant. By default, the duration of a session is one hour. You can explicitly set a different duration value when requesting the temporary security credentials for federated users and applications. For information about sending authenticated requests, see [Making requests \(p. 1138\)](#).

Note

When requesting temporary security credentials for federated users and applications, for added security, we suggest that you use a dedicated IAM user with only the necessary access permissions. The temporary user you create can never get more permissions than the IAM user who requested the temporary security credentials. For more information, see [AWS Identity and Access Management FAQs](#).

You do the following:

- Create an instance of the AWS Security Token Service client, `AmazonSecurityTokenServiceClient` class. For information about providing credentials, see [Using the AWS SDK for .NET \(p. 1192\)](#).
- Start a session by calling the `GetFederationToken` method of the STS client. You need to provide session information, including the user name and an IAM policy that you want to attach

to the temporary credentials. Optionally, you can provide a session duration. This method returns your temporary security credentials.

- Package the temporary security credentials in an instance of the `SessionAWSCredentials` object. You use this object to provide the temporary security credentials to your Amazon S3 client.
- Create an instance of the `AmazonS3Client` class by passing the temporary security credentials. You use this client to send requests to Amazon S3. If you send requests using expired credentials, Amazon S3 returns an error.

Example

The following C# example lists the keys in the specified bucket. In the example, you obtain temporary security credentials for a two-hour session for your federated user (User1), and use the credentials to send authenticated requests to Amazon S3.

- For this exercise, you create an IAM user with minimal permissions. Using the credentials of this IAM user, you request temporary credentials for others. This example lists only the objects in a specific bucket. Create an IAM user with the following policy attached:

```
{  
    "Statement": [  
        {  
            "Action": ["s3>ListBucket",  
                      "sts:GetFederationToken*"  
            ],  
            "Effect": "Allow",  
            "Resource": "*"  
        }  
    ]  
}
```

The policy allows the IAM user to request temporary security credentials and access permission only to list your AWS resources. For more information about how to create an IAM user, see [Creating Your First IAM User and Administrators Group](#) in the *IAM User Guide*.

- Use the IAM user security credentials to test the following example. The example sends authenticated request to Amazon S3 using temporary security credentials. The example specifies the following policy when requesting temporary security credentials for the federated user (User1), which restricts access to listing objects in a specific bucket (`YourBucketName`). You must update the policy and provide your own existing bucket name.

```
{  
    "Statement": [  
        {  
            "Sid": "1",  
            "Action": ["s3>ListBucket"],  
            "Effect": "Allow",  
            "Resource": "arn:aws:s3:::YourBucketName"  
        }  
    ]  
}
```

- **Example**

Update the following sample and provide the bucket name that you specified in the preceding federated user access policy. For instructions on how to create and test a working example, see [Running the Amazon S3 .NET Code Examples \(p. 1193\)](#).

```
using Amazon;  
using Amazon.Runtime;  
using Amazon.S3;
```

```
using Amazon.S3.Model;
using Amazon.SecurityToken;
using Amazon.SecurityToken.Model;
using System;
using System.Collections.Generic;
using System.Threading.Tasks;

namespace Amazon.DocSamples.S3
{
    class TempFederatedCredentialsTest
    {
        private const string bucketName = "**** bucket name ****";
        // Specify your bucket region (an example region is shown).
        private static readonly RegionEndpoint bucketRegion = RegionEndpoint.USWest2;
        private static IAmazonS3 client;

        public static void Main()
        {
            ListObjectsAsync().Wait();
        }

        private static async Task ListObjectsAsync()
        {
            try
            {
                Console.WriteLine("Listing objects stored in a bucket");
                // Credentials use the default AWS SDK for .NET credential search
                chain.
                // On local development machines, this is your default profile.
                SessionAWSCredentials tempCredentials =
                    await GetTemporaryFederatedCredentialsAsync();

                // Create a client by providing temporary security credentials.
                using (client = new AmazonS3Client(bucketRegion))
                {
                    ListObjectsRequest listObjectRequest = new ListObjectsRequest();
                    listObjectRequest.BucketName = bucketName;

                    ListObjectsResponse response = await
client.ListObjectsAsync(listObjectRequest);
                    List<S3Object> objects = response.S3Objects;
                    Console.WriteLine("Object count = {0}", objects.Count);

                    Console.WriteLine("Press any key to continue...");
                    Console.ReadKey();
                }
            }
            catch (AmazonS3Exception e)
            {
                Console.WriteLine("Error encountered ***. Message:'{0}' when writing
an object", e.Message);
            }
            catch (Exception e)
            {
                Console.WriteLine("Unknown encountered on server. Message:'{0}' when
writing an object", e.Message);
            }
        }

        private static async Task<SessionAWSCredentials>
GetTemporaryFederatedCredentialsAsync()
        {
            AmazonSecurityTokenServiceConfig config = new
AmazonSecurityTokenServiceConfig();
            AmazonSecurityTokenServiceClient stsClient =
                new AmazonSecurityTokenServiceClient(
```

```
        config);

GetFederationTokenRequest federationTokenRequest =
    new GetFederationTokenRequest();
federationTokenRequest.DurationSeconds = 7200;
federationTokenRequest.Name = "User1";
federationTokenRequest.Policy = @{
    "Statement": [
        {
            "Sid": "Stmt1311212314284",
            "Action": ["s3>ListBucket"],
            "Effect": "Allow",
            "Resource": "arn:aws:s3:::" + bucketName + "*"
        }
    ]
};

GetFederationTokenResponse federationTokenResponse =
    await
stsClient.GetFederationTokenAsync(federationTokenRequest);
Credentials credentials = federationTokenResponse.Credentials;

SessionAWSCredentials sessionCredentials =
    new SessionAWSCredentials(credentials.AccessKeyId,
                               credentials.SecretAccessKey,
                               credentials.SessionToken);
return sessionCredentials;
}
}
}
```

PHP

This topic explains how to use classes from version 3 of the AWS SDK for PHP to request temporary security credentials for federated users and applications and use them to access resources stored in Amazon S3. It assumes that you are already following the instructions for [Using the AWS SDK for PHP and Running PHP Examples \(p. 1193\)](#) and have the AWS SDK for PHP properly installed.

You can provide temporary security credentials to your federated users and applications so they can send authenticated requests to access your AWS resources. When requesting these temporary credentials, you must provide a user name and an IAM policy that describes the resource permissions that you want to grant. These credentials expire when the session duration expires. By default, the session duration is one hour. You can explicitly set a different value for the duration when requesting the temporary security credentials for federated users and applications. For more information about temporary security credentials, see [Temporary Security Credentials](#) in the *IAM User Guide*. For information about providing temporary security credentials to your federated users and applications, see [Making requests \(p. 1138\)](#).

For added security when requesting temporary security credentials for federated users and applications, we recommend using a dedicated IAM user with only the necessary access permissions. The temporary user you create can never get more permissions than the IAM user who requested the temporary security credentials. For information about identity federation, see [AWS Identity and Access Management FAQs](#).

For information about running the PHP examples in this guide, see [Running PHP Examples \(p. 1194\)](#).

Example

The following PHP example lists keys in the specified bucket. In the example, you obtain temporary security credentials for an hour session for your federated user (User1). Then you use the temporary security credentials to send authenticated requests to Amazon S3.

For added security when requesting temporary credentials for others, you use the security credentials of an IAM user who has permissions to request temporary security credentials. To ensure that the IAM user grants only the minimum application-specific permissions to the federated user, you can also limit the access permissions of this IAM user. This example lists only objects in a specific bucket. Create an IAM user with the following policy attached:

```
{  
    "Statement": [  
        {  
            "Action": ["s3>ListBucket",  
                      "sts:GetFederationToken*"  
                ],  
            "Effect": "Allow",  
            "Resource": "*"  
        }  
    ]  
}
```

The policy allows the IAM user to request temporary security credentials and access permission only to list your AWS resources. For more information about how to create an IAM user, see [Creating Your First IAM User and Administrators Group](#) in the *IAM User Guide*.

You can now use the IAM user security credentials to test the following example. The example sends an authenticated request to Amazon S3 using temporary security credentials. When requesting temporary security credentials for the federated user (User1), the example specifies the following policy, which restricts access to list objects in a specific bucket. Update the policy with your bucket name.

```
{  
    "Statement": [  
        {  
            "Sid": "1",  
            "Action": ["s3>ListBucket"],  
            "Effect": "Allow",  
            "Resource": "arn:aws:s3:::YourBucketName"  
        }  
    ]  
}
```

In the following example, when specifying the policy resource, replace **YourBucketName** with the name of your bucket.:

```
require 'vendor/autoload.php';  
  
use Aws\Sts\StsClient;  
use Aws\S3\S3Client;  
use Aws\S3\Exception\S3Exception;  
  
$bucket = '*** Your Bucket Name ***';  
  
// In real applications, the following code is part of your trusted code. It has  
// the security credentials that you use to obtain temporary security credentials.  
$sts = new StsClient(  
    [  
        'version' => 'latest',  
        'region' => 'us-east-1']
```

```

);

// Fetch the federated credentials.
$sessionToken = $sts->getFederationToken([
    'Name'          => 'User1',
    'DurationSeconds' => '3600',
    'Policy'         => json_encode([
        'Statement' => [
            'Sid'           => 'randomstatementid' . time(),
            'Action'        => ['s3>ListBucket'],
            'Effect'        => 'Allow',
            'Resource'      => 'arn:aws:s3:::' . $bucket
        ]
    ])
]);

// The following will be part of your less trusted code. You provide temporary
// security credentials so the code can send authenticated requests to Amazon S3.

$s3 = new S3Client([
    'region' => 'us-east-1',
    'version' => 'latest',
    'credentials' => [
        'key'    => $sessionToken['Credentials']['AccessKeyId'],
        'secret' => $sessionToken['Credentials']['SecretAccessKey'],
        'token'  => $sessionToken['Credentials']['SessionToken']
    ]
]);

try {
    $result = $s3->listObjects([
        'Bucket' => $bucket
    ]);
} catch (S3Exception $e) {
    echo $e->getMessage() . PHP_EOL;
}

```

Ruby

You can provide temporary security credentials for your federated users and applications so that they can send authenticated requests to access your AWS resources. When requesting temporary credentials from the IAM service, you must provide a user name and an IAM policy that describes the resource permissions that you want to grant. By default, the session duration is one hour. However, if you are requesting temporary credentials using IAM user credentials, you can explicitly set a different duration value when requesting the temporary security credentials for federated users and applications. For information about temporary security credentials for your federated users and applications, see [Making requests \(p. 1138\)](#).

Note

For added security when you request temporary security credentials for federated users and applications, you might want to use a dedicated IAM user with only the necessary access permissions. The temporary user you create can never get more permissions than the IAM user who requested the temporary security credentials. For more information, see [AWS Identity and Access Management FAQs](#).

Example

The following Ruby code example allows a federated user with a limited set of permissions to lists keys in the specified bucket.

```

# Prerequisites:
# - An existing Amazon S3 bucket.

```

```
require "aws-sdk-s3"
require "aws-sdk-iam"
require "json"

# Checks to see whether a user exists in IAM; otherwise,
# creates the user.
#
# @param iam [Aws::IAM::Client] An initialized IAM client.
# @param user_name [String] The user's name.
# @return [Aws::IAM::Types::User] The existing or new user.
# @example
#   iam = Aws::IAM::Client.new(region: 'us-west-2')
#   user = get_user(iam, 'my-user')
#   exit 1 unless user.user_name
#   puts "User's name: #{user.user_name}"
def get_user(iam, user_name)
  puts "Checking for a user with the name '#{user_name}'..."
  response = iam.get_user(user_name: user_name)
  puts "A user with the name '#{user_name}' already exists."
  return response.user
# If the user doesn't exist, create them.
rescue Aws::IAM::Errors::NoSuchEntity
  puts "A user with the name '#{user_name}' doesn't exist. Creating this user..."
  response = iam.create_user(user_name: user_name)
  iam.wait_until(:user_exists, user_name: user_name)
  puts "Created user with the name '#{user_name}'."
  return response.user
rescue StandardError => e
  puts "Error while accessing or creating the user named '#{user_name}': #{e.message}"
end

# Gets temporary AWS credentials for an IAM user with the specified permissions.
#
# @param sts [Aws::STS::Client] An initialized AWS STS client.
# @param duration_seconds [Integer] The number of seconds for valid credentials.
# @param user_name [String] The user's name.
# @param policy [Hash] The access policy.
# @return [Aws::STS::Types::Credentials] AWS credentials for API authentication.
# @example
#   sts = Aws::STS::Client.new(region: 'us-west-2')
#   credentials = get_temporary_credentials(sts, duration_seconds, user_name,
#   {
#     'Version' => '2012-10-17',
#     'Statement' => [
#       { 'Sid' => 'Stmt1',
#         'Effect' => 'Allow',
#         'Action' => 's3>ListBucket',
#         'Resource' => 'arn:aws:s3:::doc-example-bucket'
#       ]
#     }
#   )
#   exit 1 unless credentials.access_key_id
#   puts "Access key ID: #{credentials.access_key_id}"
def get_temporary_credentials(sts, duration_seconds, user_name, policy)
  response = sts.get_federation_token(
    duration_seconds: duration_seconds,
    name: user_name,
    policy: policy.to_json
  )
  return response.credentials
rescue StandardError => e
  puts "Error while getting federation token: #{e.message}"
end

# Lists the keys and ETags for the objects in an Amazon S3 bucket.
#
```

```
# @param s3_client [Aws::S3::Client] An initialized Amazon S3 client.
# @param bucket_name [String] The bucket's name.
# @return [Boolean] true if the objects were listed; otherwise, false.
# @example
#   s3_client = Aws::S3::Client.new(region: 'us-west-2')
#   exit 1 unless list_objects_in_bucket?(s3_client, 'doc-example-bucket')
def list_objects_in_bucket?(s3_client, bucket_name)
    puts "Accessing the contents of the bucket named '#{bucket_name}'..."
    response = s3_client.list_objects_v2(
        bucket: bucket_name,
        max_keys: 50
    )

    if response.count.positive?
        puts "Contents of the bucket named '#{bucket_name}' (first 50 objects):"
        puts "Name => ETag"
        response.contents.each do |obj|
            puts "#{obj.key} => #{obj.etag}"
        end
    else
        puts "No objects in the bucket named '#{bucket_name}'."
    end
    return true
rescue StandardError => e
    puts "Error while accessing the bucket named '#{bucket_name}': #{e.message}"
end

# Full example call:
# Replace us-west-2 with the AWS Region you're using for Amazon S3.
def run_me
    region = "us-west-2"
    user_name = "my-user"
    bucket_name = "doc-example-bucket"

    iam = Aws::IAM::Client.new(region: region)
    user = get_user(iam, user_name)

    exit 1 unless user.user_name

    puts "User's name: #{user.user_name}"
    sts = Aws::STS::Client.new(region: region)
    credentials = get_temporary_credentials(sts, 3600, user_name,
    {
        "Version" => "2012-10-17",
        "Statement" => [
            "Sid" => "Stmt1",
            "Effect" => "Allow",
            "Action" => "s3>ListBucket",
            "Resource" => "arn:aws:s3:::#{{bucket_name}}"
        ]
    }
)
    exit 1 unless credentials.access_key_id

    puts "Access key ID: #{credentials.access_key_id}"
    s3_client = Aws::S3::Client.new(region: region, credentials: credentials)

    exit 1 unless list_objects_in_bucket?(s3_client, bucket_name)
end

run_me if $PROGRAM_NAME == __FILE__
```

Related resources

- [Developing with Amazon S3 using the AWS SDKs, and explorers \(p. 1184\)](#)
- [AWS SDK for PHP for Amazon S3 Aws\S3\S3Client Class](#)
- [AWS SDK for PHP Documentation](#)

Making requests using the REST API

This section contains information on how to make requests to Amazon S3 endpoints by using the REST API. For a list of Amazon S3 endpoints, see [Regions and Endpoints](#) in the *AWS General Reference*.

Constructing S3 hostnames for REST API requests

Amazon S3 endpoints follow the structure shown below:

```
s3.Region.amazonaws.com
```

Amazon S3 access points endpoints and dual-stack endpoints also follow the standard structure:

- **Amazon S3 access points** - s3-accesspoint.*Region*.amazonaws.com
- **Dual-stack** - s3.dualstack.*Region*.amazonaws.com

For a complete list of Amazon S3 Regions and endpoints, see [Amazon S3 endpoints and quotas](#) in the *Amazon Web Services General Reference*.

Virtual hosted-style and path-style requests

When making requests by using the REST API, you can use virtual hosted-style or path-style URIs for the Amazon S3 endpoints. For more information, see [Virtual hosting of buckets \(p. 1175\)](#).

Example Virtual hosted-Style request

Following is an example of a virtual hosted-style request to delete the puppy.jpg file from the bucket named examplebucket in the US West (Oregon) Region. For more information about virtual hosted-style requests, see [Virtual-hosted-style requests \(p. 1176\)](#).

```
DELETE /puppy.jpg HTTP/1.1
Host: examplebucket.s3.us-west-2.amazonaws.com
Date: Mon, 11 Apr 2016 12:00:00 GMT
x-amz-date: Mon, 11 Apr 2016 12:00:00 GMT
Authorization: authorization string
```

Example Path-style request

Following is an example of a path-style version of the same request.

```
DELETE /examplebucket/puppy.jpg HTTP/1.1
Host: s3.us-west-2.amazonaws.com
Date: Mon, 11 Apr 2016 12:00:00 GMT
x-amz-date: Mon, 11 Apr 2016 12:00:00 GMT
Authorization: authorization string
```

Currently, Amazon S3 supports both virtual-hosted-style and path-style URL access in all AWS Regions. However, path-style URLs will be discontinued in the future. For more information, see the following **Important note**.

For more information about path-style requests, see [Path-style requests \(p. 1176\)](#).

Important

Update (September 23, 2020) – To make sure that customers have the time that they need to transition to virtual-hosted-style URLs, we have decided to delay the deprecation of path-style URLs. For more information, see [Amazon S3 Path Deprecation Plan – The Rest of the Story](#) in the [AWS News Blog](#).

Making requests to dual-stack endpoints by using the REST API

When using the REST API, you can directly access a dual-stack endpoint by using a virtual hosted-style or a path style endpoint name (URI). All Amazon S3 dual-stack endpoint names include the region in the name. Unlike the standard IPv4-only endpoints, both virtual hosted-style and a path-style endpoints use region-specific endpoint names.

Example Virtual hosted-Style dual-stack endpoint request

You can use a virtual hosted-style endpoint in your REST request as shown in the following example that retrieves the `puppy.jpg` object from the bucket named `examplebucket` in the US West (Oregon) Region.

```
GET /puppy.jpg HTTP/1.1
Host: examplebucket.s3.dualstack.us-west-2.amazonaws.com
Date: Mon, 11 Apr 2016 12:00:00 GMT
x-amz-date: Mon, 11 Apr 2016 12:00:00 GMT
Authorization: authorization string
```

Example Path-style dual-stack endpoint request

Or you can use a path-style endpoint in your request as shown in the following example.

```
GET /examplebucket/puppy.jpg HTTP/1.1
Host: s3.dualstack.us-west-2.amazonaws.com
Date: Mon, 11 Apr 2016 12:00:00 GMT
x-amz-date: Mon, 11 Apr 2016 12:00:00 GMT
Authorization: authorization string
```

For more information about dual-stack endpoints, see [Using Amazon S3 dual-stack endpoints \(p. 1142\)](#).

For more information about making requests using the REST API, see the topics bellow.

Topics

- [Virtual hosting of buckets \(p. 1175\)](#)
- [Request redirection and the REST API \(p. 1181\)](#)

Virtual hosting of buckets

Virtual hosting is the practice of serving multiple websites from a single web server. One way to differentiate sites in your Amazon S3 REST API requests is by using the apparent hostname of the Request-URI instead of just the path name part of the URI. An ordinary Amazon S3 REST request specifies a bucket by using the first slash-delimited component of the Request-URI path. Instead, you can use Amazon S3 virtual hosting to address a bucket in a REST API call by using the `HTTP Host` header. In practice, Amazon S3 interprets `Host` as meaning that most buckets are automatically accessible

for limited types of requests at `https://bucket-name.s3.region-code.amazonaws.com`. For a complete list of Amazon S3 Regions and endpoints, see [Amazon S3 endpoints and quotas](#) in the *Amazon Web Services General Reference*.

Virtual hosting also has other benefits. By naming your bucket after your registered domain name and by making that name a DNS alias for Amazon S3, you can completely customize the URL of your Amazon S3 resources, for example, `http://my.bucket-name.com/`. You can also publish to the "root directory" of your bucket's virtual server. This ability can be important because many existing applications search for files in this standard location. For example, `favicon.ico`, `robots.txt`, and `crossdomain.xml` are all expected to be found at the root.

Important

When you're using virtual-hosted-style buckets with SSL, the SSL wildcard certificate matches only buckets that do not contain dots (.). To work around this limitation, use HTTP or write your own certificate-verification logic. For more information, see [Amazon S3 Path Deprecation Plan](#) on the *AWS News Blog*.

Topics

- [Path-style requests \(p. 1176\)](#)
- [Virtual-hosted-style requests \(p. 1176\)](#)
- [HTTP Host header bucket specification \(p. 1177\)](#)
- [Examples \(p. 1177\)](#)
- [Customizing Amazon S3 URLs with CNAME records \(p. 1178\)](#)
- [How to associate a hostname with an Amazon S3 bucket \(p. 1179\)](#)
- [Limitations \(p. 1180\)](#)
- [Backward compatibility \(p. 1180\)](#)

Path-style requests

Currently, Amazon S3 supports both virtual-hosted-style and path-style URL access in all AWS Regions. However, path-style URLs will be discontinued in the future. For more information, see the following **Important note**.

In Amazon S3, path-style URLs use the following format:

```
https://s3.region-code.amazonaws.com/bucket-name/key-name
```

For example, if you create a bucket named `DOC-EXAMPLE-BUCKET1` in the US West (Oregon) Region, and you want to access the `puppy.jpg` object in that bucket, you can use the following path-style URL:

```
https://s3.us-west-2.amazonaws.com/DOC-EXAMPLE-BUCKET1/puppy.jpg
```

Important

Update (September 23, 2020) – To make sure that customers have the time that they need to transition to virtual-hosted-style URLs, we have decided to delay the deprecation of path-style URLs. For more information, see [Amazon S3 Path Deprecation Plan – The Rest of the Story](#) in the *AWS News Blog*.

Virtual-hosted-style requests

In a virtual-hosted-style URI, the bucket name is part of the domain name in the URL.

Amazon S3 virtual-hosted-style URLs use the following format:

```
https://bucket-name.s3.region-code.amazonaws.com/key-name
```

In this example, `DOC-EXAMPLE-BUCKET1` is the bucket name, US West (Oregon) is the Region, and `puppy.png` is the key name:

```
https://DOC-EXAMPLE-BUCKET1.s3.us-west-2.amazonaws.com/puppy.png
```

HTTP Host header bucket specification

As long as your GET request does not use the SSL endpoint, you can specify the bucket for the request by using the HTTP Host header. The Host header in a REST request is interpreted as follows:

- If the Host header is omitted or its value is `s3.region-code.amazonaws.com`, the bucket for the request will be the first slash-delimited component of the Request-URI, and the key for the request will be the rest of the Request-URI. This is the ordinary method, as illustrated by the first and second examples in this section. Omitting the Host header is valid only for HTTP 1.0 requests.
- Otherwise, if the value of the Host header ends in `.s3.region-code.amazonaws.com`, the bucket name is the leading component of the Host header's value up to `.s3.region-code.amazonaws.com`. The key for the request is the Request-URI. This interpretation exposes buckets as subdomains of `.s3.region-code.amazonaws.com`, as illustrated by the third and fourth examples in this section.
- Otherwise, the bucket for the request is the lowercase value of the Host header, and the key for the request is the Request-URI. This interpretation is useful when you have registered the same DNS name as your bucket name and have configured that name to be a canonical name (CNAME) alias for Amazon S3. The procedure for registering domain names and configuring CNAME DNS records is beyond the scope of this guide, but the result is illustrated by the final example in this section.

Examples

This section provides example URLs and requests.

Example – Path-style URLs and requests

This example uses the following:

- Bucket Name - `example.com`
- Region - US East (N. Virginia)
- Key Name - `homepage.html`

The URL is as follows:

```
http://s3.us-east-1.amazonaws.com/example.com/homepage.html
```

The request is as follows:

```
GET /example.com/homepage.html HTTP/1.1
Host: s3.us-east-1.amazonaws.com
```

The request with HTTP 1.0 and omitting the Host header is as follows:

```
GET /example.com/homepage.html HTTP/1.0
```

For information about DNS-compatible names, see [Limitations \(p. 1180\)](#). For more information about keys, see [Keys \(p. 5\)](#).

Example – Virtual-hosted-style URLs and requests

This example uses the following:

- **Bucket name** - DOC-EXAMPLE-BUCKET1.eu
- **Region** - Europe (Ireland)
- **Key name** - homepage.html

The URL is as follows:

```
http://DOC-EXAMPLE-BUCKET1.eu.s3.eu-west-1.amazonaws.com/homepage.html
```

The request is as follows:

```
GET /homepage.html HTTP/1.1
Host: DOC-EXAMPLE-BUCKET1.eu.s3.eu-west-1.amazonaws.com
```

Example – CNAME alias method

To use this method, you must configure your DNS name as a CNAME alias for `bucket-name.s3.us-east-1.amazonaws.com`. For more information, see [Customizing Amazon S3 URLs with CNAME records \(p. 1178\)](#).

This example uses the following:

- **Bucket Name** - example.com
- **Key name** - homepage.html

The URL is as follows:

```
http://www.example.com/homepage.html
```

The example is as follows:

```
GET /homepage.html HTTP/1.1
Host: www.example.com
```

Customizing Amazon S3 URLs with CNAME records

Depending on your needs, you might not want `s3.region-code.amazonaws.com` to appear on your website or service. For example, if you're hosting website images on Amazon S3, you might prefer `http://images.example.com/` instead of `http://images.example.com.s3.us-east-1.amazonaws.com/`. Any bucket with a DNS-compatible name can be referenced as follows: `http://BucketName.s3.Region.amazonaws.com/[Filename]`, for example, `http://images.example.com.s3.us-east-1.amazonaws.com/mydog.jpg`. By using CNAME, you can map `images.example.com` to an Amazon S3 hostname so that the previous URL could become `http://images.example.com/mydog.jpg`.

Your bucket name must be the same as the CNAME. For example, if you create a CNAME to map `images.example.com` to `images.example.com.s3.us-east-1.amazonaws.com`, both `http://images.example.com/filename` and `http://images.example.com.s3.us-east-1.amazonaws.com/filename` will be the same.

The CNAME DNS record should alias your domain name to the appropriate virtual hosted-style hostname. For example, if your bucket name and domain name are `images.example.com`

and your bucket is in the US East (N. Virginia) Region, the CNAME record should alias to `images.example.com.s3.us-east-1.amazonaws.com`.

```
images.example.com CNAME images.example.com.s3.us-east-1.amazonaws.com.
```

Amazon S3 uses the hostname to determine the bucket name. So the CNAME and the bucket name must be the same. For example, suppose that you have configured `www.example.com` as a CNAME for `www.example.com.s3.us-east-1.amazonaws.com`. When you access `http://www.example.com`, Amazon S3 receives a request similar to the following:

Example

```
GET / HTTP/1.1
Host: www.example.com
Date: date
Authorization: signatureValue
```

Amazon S3 sees only the original hostname `www.example.com` and is unaware of the CNAME mapping used to resolve the request.

You can use any Amazon S3 endpoint in a CNAME alias. For example, `s3.ap-southeast-1.amazonaws.com` can be used in CNAME aliases. For more information about endpoints, see [Request Endpoints \(p. 1140\)](#). To create a static website by using a custom domain, see [Configuring a static website using a custom domain registered with Route 53 \(p. 97\)](#).

Important

When using custom URLs with CNAMEs, you will need to ensure a matching bucket exists for any CNAME or alias record you configure. For example, if you create DNS entries for `www.example.com` and `login.example.com` to publish web content using S3, you will need to create both buckets `www.example.com` and `login.example.com`.

When a CNAME or alias records is configured pointing to an S3 endpoint without a matching bucket, any AWS user can create that bucket and publish content under the configured alias, even if ownership is not the same.

For the same reason, we recommend that you change or remove the corresponding CNAME or alias when deleting a bucket.

How to associate a hostname with an Amazon S3 bucket

To associate a hostname with an Amazon S3 bucket by using a CNAME alias

1. Select a hostname that belongs to a domain that you control.

This example uses the `images` subdomain of the `example.com` domain.

2. Create a bucket that matches the hostname.

In this example, the host and bucket names are `images.example.com`. The bucket name must *exactly* match the hostname.

3. Create a CNAME DNS record that defines the hostname as an alias for the Amazon S3 bucket.

For example:

```
images.example.com CNAME images.example.com.s3.us-west-2.amazonaws.com
```

Important

For request-routing reasons, the CNAME DNS record must be defined exactly as shown in the preceding example. Otherwise, it might appear to operate correctly, but it will eventually result in unpredictable behavior.

The procedure for configuring CNAME DNS records depends on your DNS server or DNS provider. For specific information, see your server documentation or contact your provider.

Limitations

SOAP support over HTTP is deprecated, but SOAP is still available over HTTPS. New Amazon S3 features are not supported for SOAP. Instead of using SOAP, we recommend that you use either the REST API or the AWS SDKs.

Backward compatibility

The following sections cover various aspects of Amazon S3 backward compatibility that relate to path-style and virtual-hosted-style URL requests.

Legacy endpoints

Some Regions support legacy endpoints. You might see these endpoints in your server access logs or AWS CloudTrail logs. For more information, review the following information. For a complete list of Amazon S3 Regions and endpoints, see [Amazon S3 endpoints and quotas](#) in the *Amazon Web Services General Reference*.

Important

Although you might see legacy endpoints in your logs, we recommend that you always use the standard endpoint syntax to access your buckets.

Amazon S3 virtual-hosted-style URLs use the following format:

```
https://bucket-name.s3.region-code.amazonaws.com/key-name
```

In Amazon S3, path-style URLs use the following format:

```
https://s3.region-code.amazonaws.com/bucket-name/key-name
```

s3-Region

Some older Amazon S3 Regions support endpoints that contain a dash (-) between s3 and the Region code (for example, s3#us-west-2), instead of a dot (for example, s3.us-west-2). If your bucket is in one of these Regions, you might see the following endpoint format in your server access logs or CloudTrail logs:

```
https://bucket-name.s3-region-code.amazonaws.com
```

In this example, the bucket name is DOC-EXAMPLE-BUCKET1 and the Region is US West (Oregon):

```
https://DOC-EXAMPLE-BUCKET1.s3-us-west-2.amazonaws.com
```

Legacy global endpoint

For some Regions, you can use the legacy global endpoint to construct requests that do not specify a Region-specific endpoint. The legacy global endpoint point is as follows:

```
bucket-name.s3.amazonaws.com
```

In your server access logs or CloudTrail logs, you might see requests that use the legacy global endpoint. In this example, the bucket name is DOC-EXAMPLE-BUCKET1 and the legacy global endpoint is:

`https://DOC-EXAMPLE-BUCKET1.s3.amazonaws.com`

Virtual-hosted-style requests for US East (N. Virginia)

Requests made with the legacy global endpoint go to the US East (N. Virginia) Region by default. Therefore, the legacy global endpoint is sometimes used in place of the Regional endpoint for US East (N. Virginia). If you create a bucket in US East (N. Virginia) and use the global endpoint, Amazon S3 routes your request to this Region by default.

Virtual-hosted-style requests for other Regions

The legacy global endpoint is also used for virtual-hosted-style requests in other supported Regions. If you create a bucket in a Region that was launched before March 20, 2019, and use the legacy global endpoint, Amazon S3 updates the DNS record to reroute the request to the correct location, which might take time. In the meantime, the default rule applies, and your virtual-hosted-style request goes to the US East (N. Virginia) Region. Amazon S3 then redirects it with an HTTP 307 Temporary Redirect to the correct Region.

For S3 buckets in Regions launched after March 20, 2019, the DNS server doesn't route your request directly to the AWS Region where your bucket resides. It returns an HTTP 400 Bad Request error instead. For more information, see [Making requests \(p. 1138\)](#).

Path-style requests

For the US East (N. Virginia) Region, you can use the legacy global endpoint for path-style requests.

For all other Regions, the path-style syntax requires that you use the Region-specific endpoint when attempting to access a bucket. If you try to access a bucket with the legacy global endpoint or another endpoint that is different than the one for the Region where the bucket resides, you receive an HTTP response code 307 Temporary Redirect error and a message that indicates the correct URI for your resource. For example, if you use `https://s3.amazonaws.com/bucket-name` for a bucket that was created in the US West (Oregon) Region, you will receive an HTTP 307 Temporary Redirect error.

Request redirection and the REST API

Topics

- [Redirects and HTTP user-agents \(p. 1181\)](#)
- [Redirects and 100-Continue \(p. 1182\)](#)
- [Redirect example \(p. 1182\)](#)

This section describes how to handle HTTP redirects by using the Amazon S3 REST API. For general information about Amazon S3 redirects, see [Making requests \(p. 1138\)](#) in the Amazon Simple Storage Service API Reference.

Redirects and HTTP user-agents

Programs that use the Amazon S3 REST API should handle redirects either at the application layer or the HTTP layer. Many HTTP client libraries and user agents can be configured to correctly handle redirects automatically; however, many others have incorrect or incomplete redirect implementations.

Before you rely on a library to fulfill the redirect requirement, test the following cases:

- Verify all HTTP request headers are correctly included in the redirected request (the second request after receiving a redirect) including HTTP standards such as Authorization and Date.
- Verify non-GET redirects, such as PUT and DELETE, work correctly.
- Verify large PUT requests follow redirects correctly.
- Verify PUT requests follow redirects correctly if the 100-continue response takes a long time to arrive.

HTTP user-agents that strictly conform to RFC 2616 might require explicit confirmation before following a redirect when the HTTP request method is not GET or HEAD. It is generally safe to follow redirects generated by Amazon S3 automatically, as the system will issue redirects only to hosts within the amazonaws.com domain and the effect of the redirected request will be the same as that of the original request.

Redirects and 100-Continue

To simplify redirect handling, improve efficiencies, and avoid the costs associated with sending a redirected request body twice, configure your application to use 100-continues for PUT operations. When your application uses 100-continue, it does not send the request body until it receives an acknowledgement. If the message is rejected based on the headers, the body of the message is not sent. For more information about 100-continue, go to [RFC 2616 Section 8.2.3](#)

Note

According to RFC 2616, when using `Expect: Continue` with an unknown HTTP server, you should not wait an indefinite period before sending the request body. This is because some HTTP servers do not recognize 100-continue. However, Amazon S3 does recognize if your request contains an `Expect: Continue` and will respond with a provisional 100-continue status or a final status code. Additionally, no redirect error will occur after receiving the provisional 100 continue go-ahead. This will help you avoid receiving a redirect response while you are still writing the request body.

Redirect example

This section provides an example of client-server interaction using HTTP redirects and 100-continue.

Following is a sample PUT to the quotes.s3.amazonaws.com bucket.

```
PUT /nelson.txt HTTP/1.1
Host: quotes.s3.amazonaws.com
Date: Mon, 15 Oct 2007 22:18:46 +0000

Content-Length: 6
Expect: 100-continue
```

Amazon S3 returns the following:

```
HTTP/1.1 307 Temporary Redirect
Location: http://quotes.s3-4c25d83b.amazonaws.com/nelson.txt?rk=8d47490b
Content-Type: application/xml
Transfer-Encoding: chunked
Date: Mon, 15 Oct 2007 22:18:46 GMT

Server: AmazonS3

<?xml version="1.0" encoding="UTF-8"?>
<Error>
  <Code>TemporaryRedirect</Code>
  <Message>Please re-send this request to the
specified temporary endpoint. Continue to use the
original request endpoint for future requests.
</Message>
  <Endpoint>quotes.s3-4c25d83b.amazonaws.com</Endpoint>
  <Bucket>quotes</Bucket>
</Error>
```

The client follows the redirect response and issues a new request to the quotes.s3-4c25d83b.amazonaws.com temporary endpoint.

```
PUT /nelson.txt?rk=8d47490b HTTP/1.1
Host: quotes.s3-4c25d83b.amazonaws.com
Date: Mon, 15 Oct 2007 22:18:46 +0000

Content-Length: 6
Expect: 100-continue
```

Amazon S3 returns a 100-continue indicating the client should proceed with sending the request body.

```
HTTP/1.1 100 Continue
```

The client sends the request body.

```
ha ha\n
```

Amazon S3 returns the final response.

```
HTTP/1.1 200 OK
Date: Mon, 15 Oct 2007 22:18:48 GMT

ETag: "a2c8d6b872054293af41061e93bc289"
Content-Length: 0
Server: AmazonS3
```

Developing with Amazon S3 using the AWS CLI

Follow these steps to download and configure AWS Command Line Interface (AWS CLI).

For a list of Amazon S3 AWS CLI commands, see the following pages in the *AWS CLI Command Reference*:

- [s3](#)
- [s3api](#)
- [s3control](#)

Note

Services in AWS, such as Amazon S3, require that you provide credentials when you access them. The service can then determine whether you have permissions to access the resources that it owns. The console requires your password. You can create access keys for your AWS account to access the AWS CLI or API. However, we don't recommend that you access AWS using the credentials for your AWS account. Instead, we recommend that you use AWS Identity and Access Management (IAM). Create an IAM user, add the user to an IAM group with administrative permissions, and then grant administrative permissions to the IAM user that you created. You can then access AWS using a special URL and that IAM user's credentials. For instructions, go to [Creating Your First IAM User and Administrators Group](#) in the *IAM User Guide*.

To set up the AWS CLI

1. Download and configure the AWS CLI. For instructions, see the following topics in the *AWS Command Line Interface User Guide*:

- [Getting Set Up with the AWS Command Line Interface](#)
- [Configuring the AWS Command Line Interface](#)

2. Add a named profile for the administrator user in the AWS CLI config file. You use this profile when executing the AWS CLI commands. For more information, see [Named profiles for the AWS CLI](#) in the *AWS Command Line Interface User Guide*.

```
[adminuser]
aws_access_key_id = adminuser access key ID
aws_secret_access_key = adminuser secret access key
region = aws-region
```

For a list of available AWS Regions, see [Regions and Endpoints](#) in the *AWS General Reference*.

3. Verify the setup by typing the following commands at the command prompt.

- Try the `help` command to verify that the AWS CLI is installed on your computer:

```
aws help
```

- Run an `s3` command using the `adminuser` credentials that you just created. To do this, add the `--profile` parameter to your command to specify the profile name. In this example, the `ls` command lists buckets in your account. The AWS CLI uses the `adminuser` credentials to authenticate the request.

```
aws s3 ls --profile adminuser
```

Developing with Amazon S3 using the AWS SDKs, and explorers

You can use the AWS SDKs when developing applications with Amazon S3. The AWS SDKs simplify your programming tasks by wrapping the underlying REST API. The AWS Mobile SDKs and the AWS Amplify JavaScript library are also available for building connected mobile and web applications using AWS.

This section provides an overview of using AWS SDKs for developing Amazon S3 applications. This section also describes how you can test the AWS SDK code examples provided in this guide.

Topics

- [Using this service with an AWS SDK \(p. 1185\)](#)
- [Specifying the Signature Version in Request Authentication \(p. 1186\)](#)
- [Using the AWS SDK for Java \(p. 1191\)](#)
- [Using the AWS SDK for .NET \(p. 1192\)](#)
- [Using the AWS SDK for PHP and Running PHP Examples \(p. 1193\)](#)
- [Using the AWS SDK for Ruby - Version 3 \(p. 1194\)](#)
- [Using the AWS SDK for Python \(Boto\) \(p. 1196\)](#)
- [Using the AWS Mobile SDKs for iOS and Android \(p. 1196\)](#)
- [Using the AWS Amplify JavaScript Library \(p. 1196\)](#)
- [Using the AWS SDK for JavaScript \(p. 1196\)](#)

In addition to the AWS SDKs, AWS Explorers are available for Visual Studio and Eclipse for Java IDE. In this case, the SDKs and the explorers are available bundled together as AWS Toolkits.

You can also use the AWS Command Line Interface (AWS CLI) to manage Amazon S3 buckets and objects.

AWS Toolkit for Eclipse

The AWS Toolkit for Eclipse includes both the AWS SDK for Java and AWS Explorer for Eclipse. The AWS Explorer for Eclipse is an open source plugin for Eclipse for Java IDE that makes it easier for developers to develop, debug, and deploy Java applications using AWS. The easy-to-use GUI enables you to access and administer your AWS infrastructure including Amazon S3. You can perform common operations such as managing your buckets and objects and setting IAM policies, while developing applications, all from within the context of Eclipse for Java IDE. For set up instructions, see [Set up the Toolkit](#). For examples of using the explorer, see [How to Access AWS Explorer](#).

AWS Toolkit for Visual Studio

AWS Explorer for Visual Studio is an extension for Microsoft Visual Studio that makes it easier for developers to develop, debug, and deploy .NET applications using Amazon Web Services. The easy-to-use GUI enables you to access and administer your AWS infrastructure including Amazon S3. You can perform common operations such as managing your buckets and objects or setting IAM policies, while developing applications, all from within the context of Visual Studio. For setup instructions, go to [Setting Up the AWS Toolkit for Visual Studio](#). For examples of using Amazon S3 using the explorer, see [Using Amazon S3 from AWS Explorer](#).

AWS SDKs

You can download only the SDKs. For information about downloading the SDK libraries, see [Sample Code Libraries](#).

AWS CLI

The AWS CLI is a unified tool to manage your AWS services, including Amazon S3. For information about downloading the AWS CLI, see [AWS Command Line Interface](#).

Using this service with an AWS SDK

AWS software development kits (SDKs) are available for many popular programming languages. Each SDK provides an API, code examples, and documentation that make it easier for developers to build applications in their preferred language.

SDK documentation	Code examples
AWS SDK for C++	AWS SDK for C++ code examples
AWS SDK for Go	AWS SDK for Go code examples
AWS SDK for Java	AWS SDK for Java code examples
AWS SDK for JavaScript	AWS SDK for JavaScript code examples
AWS SDK for Kotlin	AWS SDK for Kotlin code examples
AWS SDK for .NET	AWS SDK for .NET code examples
AWS SDK for PHP	AWS SDK for PHP code examples
AWS SDK for Python (Boto3)	AWS SDK for Python (Boto3) code examples
AWS SDK for Ruby	AWS SDK for Ruby code examples
AWS SDK for Rust	AWS SDK for Rust code examples
AWS SDK for Swift	AWS SDK for Swift code examples

For examples specific to this service, see [Code examples for Amazon S3 using AWS SDKs \(p. 1326\)](#).

Example availability

Can't find what you need? Request a code example by using the **Provide feedback** link at the bottom of this page.

Specifying the Signature Version in Request Authentication

Amazon S3 supports only AWS Signature Version 4 in most AWS Regions. In some of the older AWS Regions, Amazon S3 supports both Signature Version 4 and Signature Version 2. However, Signature Version 2 is being turned off (deprecated). For more information about the end of support for Signature Version 2, see [AWS Signature Version 2 Turned Off \(Deprecated\) for Amazon S3 \(p. 1187\)](#).

For a list of all the Amazon S3 Regions and the signature versions they support, see [Regions and Endpoints](#) in the *AWS General Reference*.

For all AWS Regions, AWS SDKs use Signature Version 4 by default to authenticate requests. When using AWS SDKs that were released before May 2016, you might be required to request Signature Version 4, as shown in the following table.

SDK	Requesting Signature Version 4 for Request Authentication
AWS CLI	<p>For the default profile, run the following command:</p> <pre>\$ aws configure set default.s3.signature_version s3v4</pre> <p>For a custom profile, run the following command:</p> <pre>\$ aws configure set profile.your_profile_name.s3.signature_version s3v4</pre>
Java SDK	<p>Add the following in your code:</p> <pre>System.setProperty(SDKGlobalConfiguration.ENABLE_S3_SIGV4_SYSTEM_PROPERTY, "true");</pre> <p>Or, on the command line, specify the following:</p> <pre>-Dcom.amazonaws.services.s3.enableV4</pre>
JavaScript SDK	<p>Set the <code>signatureVersion</code> parameter to <code>v4</code> when constructing the client:</p> <pre>var s3 = new AWS.S3({signatureVersion: 'v4'});</pre>
PHP SDK	<p>Set the <code>signature</code> parameter to <code>v4</code> when constructing the Amazon S3 service client for PHP SDK v2:</p> <pre><?php \$client = S3Client::factory(['region' => 'YOUR-REGION', 'version' => 'latest',</pre>

SDK	Requesting Signature Version 4 for Request Authentication
	<pre>'signature' => 'v4']);</pre> <p>When using the PHP SDK v3, set the <code>signature_version</code> parameter to <code>v4</code> during construction of the Amazon S3 service client:</p> <pre><?php \$s3 = new Aws\S3\S3Client(['version' => '2006-03-01', 'region' => 'YOUR-REGION', 'signature_version' => 'v4']);</pre>
Python-Boto SDK	<p>Specify the following in the boto default config file:</p> <pre>[s3] use-sigv4 = True</pre>
Ruby SDK	<p>Ruby SDK - Version 1: Set the <code>:s3_signature_version</code> parameter to <code>:v4</code> when constructing the client:</p> <pre>s3 = AWS::S3::Client.new(:s3_signature_version => :v4)</pre> <p>Ruby SDK - Version 3: Set the <code>signature_version</code> parameter to <code>v4</code> when constructing the client:</p> <pre>s3 = Aws::S3::Client.new(signature_version: 'v4')</pre>
.NET SDK	<p>Add the following to the code before creating the Amazon S3 client:</p> <pre>AWSConfigsS3.UseSignatureVersion4 = true;</pre> <p>Or, add the following to the config file:</p> <pre><appSettings> <add key="AWS.S3.UseSignatureVersion4" value="true" /> </appSettings></pre>

AWS Signature Version 2 Turned Off (Deprecated) for Amazon S3

Signature Version 2 is being turned off (deprecated) in Amazon S3. Amazon S3 will then only accept API requests that are signed using Signature Version 4.

This section provides answers to common questions regarding the end of support for Signature Version 2.

What is Signature Version 2/4, and What Does It Mean to Sign Requests?

The Signature Version 2 or Signature Version 4 signing process is used to authenticate your Amazon S3 API requests. Signing requests enables Amazon S3 to identify who is sending the request and protects your requests from bad actors.

For more information about signing AWS requests, see [Signing AWS API Requests](#) in the *AWS General Reference*.

What Update Are You Making?

We currently support Amazon S3 API requests that are signed using Signature Version 2 and Signature Version 4 processes. After that, Amazon S3 will only accept requests that are signed using Signature Version 4.

For more information about signing AWS requests, see [Changes in Signature Version 4](#) in the *AWS General Reference*.

Why Are You Making the Update?

Signature Version 4 provides improved security by using a signing key instead of your secret access key. Signature Version 4 is currently supported in all AWS Regions, whereas Signature Version 2 is only supported in Regions that were launched before January 2014. This update allows us to provide a more consistent experience across all Regions.

How Do I Ensure That I'm Using Signature Version 4, and What Updates Do I Need?

The signature version that is used to sign your requests is usually set by the tool or the SDK on the client side. By default, the latest versions of our AWS SDKs use Signature Version 4. For third-party software, contact the appropriate support team for your software to confirm what version you need. If you are sending direct REST calls to Amazon S3, you must modify your application to use the Signature Version 4 signing process.

For information about which version of the AWS SDKs to use when moving to Signature Version 4, see [Moving from Signature Version 2 to Signature Version 4 \(p. 1189\)](#).

For information about using Signature Version 4 with the Amazon S3 REST API, see [Authenticating Requests \(AWS Signature Version 4\)](#) in the *Amazon Simple Storage Service API Reference*.

What Happens if I Don't Make Updates?

Requests signed with Signature Version 2 that are made after that will fail to authenticate with Amazon S3. Requesters will see errors stating that the request must be signed with Signature Version 4.

Should I Make Changes Even if I'm Using a Presigned URL That Requires Me to Sign for More than 7 Days?

If you are using a presigned URL that requires you to sign for more than 7 days, no action is currently needed. You can continue to use AWS Signature Version 2 to sign and authenticate the presigned URL. We will follow up and provide more details on how to migrate to Signature Version 4 for a presigned URL scenario.

More Info

- For more information about using Signature Version 4, see [Signing AWS API Requests](#).
- View the list of changes between Signature Version 2 and Signature Version 4 in [Changes in Signature Version 4](#).
- View the post [AWS Signature Version 4 to replace AWS Signature Version 2 for signing Amazon S3 API requests](#) in the AWS forums.
- If you have any questions or concerns, contact [AWS Support](#).

Moving from Signature Version 2 to Signature Version 4

If you currently use Signature Version 2 for Amazon S3 API request authentication, you should move to using Signature Version 4. Support is ending for Signature Version 2, as described in [AWS Signature Version 2 Turned Off \(Deprecated\) for Amazon S3 \(p. 1187\)](#).

For information about using Signature Version 4 with the Amazon S3 REST API, see [Authenticating Requests \(AWS Signature Version 4\)](#) in the *Amazon Simple Storage Service API Reference*.

The following table lists the SDKs with the necessary minimum version to use Signature Version 4 (SigV4). If you are using presigned URLs with the AWS Java, JavaScript (Node.js), or Python (Boto/CLI) SDKs, you must set the correct AWS Region and set Signature Version 4 in the client configuration. For information about setting SigV4 in the client configuration, see [Specifying the Signature Version in Request Authentication \(p. 1186\)](#).

If you use this SDK/Product	Upgrade to this SDK version	Code change needed to the client to use Sigv4?	Link to SDK documentation
AWS SDK for Java v1	Upgrade to Java 1.11.201+ or v2 in Q4 2018.	Yes	Specifying the Signature Version in Request Authentication (p. 1186)
AWS SDK for Java v2 (preview)	No SDK upgrade is needed.	No	AWS SDK for Java
AWS SDK for .NET v1	Upgrade to 3.1.10 or later.	Yes	AWS SDK for .NET
AWS SDK for .NET v2	Upgrade to 3.1.10 or later.	No	AWS SDK for .NET v2
AWS SDK for .NET v3	Upgrade to 3.3.0.0 or later.	Yes	AWS SDK for .NET v3
AWS SDK for JavaScript v1	Upgrade to 2.68.0 or later.	Yes	AWS SDK for JavaScript
AWS SDK for JavaScript v2	Upgrade to 2.68.0 or later.	Yes	AWS SDK for JavaScript
AWS SDK for JavaScript v3	No action is currently needed. Upgrade to major version V3 in Q3 2019.	No	AWS SDK for JavaScript
AWS SDK for PHP v1	Recommend to upgrade to the most recent version of PHP or, at least to v2.7.4 with the signature parameter	Yes	AWS SDK for PHP

If you use this SDK/Product	Upgrade to this SDK version	Code change needed to the client to use Sigv4?	Link to SDK documentation
	set to v4 in the S3 client's configuration.		
AWS SDK for PHP v2	Recommend to upgrade to the most recent version of PHP or, at least to v2.7.4 with the signature parameter set to v4 in the S3 client's configuration.	No	AWS SDK for PHP
AWS SDK for PHP v3	No SDK upgrade is needed.	No	AWS SDK for PHP
Boto2	Upgrade to Boto2 v2.49.0.	Yes	Boto 2 Upgrade
Boto3	Upgrade to 1.5.71 (Botocore), 1.4.6 (Boto3).	Yes	Boto 3 - AWS SDK for Python
AWS CLI	Upgrade to 1.11.108.	Yes	AWS Command Line Interface
AWS CLI v2 (preview)	No SDK upgrade is needed.	No	AWS Command Line Interface version 2
AWS SDK for Ruby v1	Upgrade to Ruby V3.	Yes	Ruby V3 for AWS
AWS SDK for Ruby v2	Upgrade to Ruby V3.	Yes	Ruby V3 for AWS
AWS SDK for Ruby v3	No SDK upgrade is needed.	No	Ruby V3 for AWS
Go	No SDK upgrade is needed.	No	AWS SDK for Go
C++	No SDK upgrade is needed.	No	AWS SDK for C++

AWS Tools for Windows PowerShell or AWS Tools for PowerShell Core

If you are using module versions *earlier* than 3.3.0.0, you must upgrade to 3.3.0.0.

To get the version information, use the `Get-Module` cmdlet:

```
Get-Module -Name AWSPowershell
Get-Module -Name AWSPowershell.NetCore
```

To update the 3.3.0.0 version, use the `Update-Module` cmdlet:

```
Update-Module -Name AWSPowershell
Update-Module -Name AWSPowershell.NetCore
```

You can use presigned URLs that are valid for more than 7 days that you will send Signature Version 2 traffic on.

Using the AWS SDK for Java

The AWS SDK for Java provides an API for the Amazon S3 bucket and object operations. For object operations, in addition to providing the API to upload objects in a single operation, the SDK provides an API to upload large objects in parts. For more information, see [Uploading and copying objects using multipart upload \(p. 167\)](#).

Topics

- [The Java API Organization \(p. 1192\)](#)
- [Testing the Amazon S3 Java Code Examples \(p. 1192\)](#)

The AWS SDK for Java gives you the option of using a high-level or low-level API.

Low-Level API

The low-level APIs correspond to the underlying Amazon S3 REST operations, such as create, update, and delete operations that apply to buckets and objects. When you upload large objects using the low-level multipart upload API, it provides greater control. For example, it lets you pause and resume multipart uploads, vary part sizes during the upload, or begin uploads when you don't know the size of the data in advance. If you don't have these requirements, use the high-level API to upload objects.

High-Level API

For uploading objects, the SDK provides a higher level of abstraction by providing the `TransferManager` class. The high-level API is a simpler API, where in just a few lines of code you can upload files and streams to Amazon S3. You should use this API to upload data unless you need to control the upload as described in the preceding Low-Level API section.

For smaller data size, the `TransferManager` API uploads data in a single operation. However, the `TransferManager` switches to using the multipart upload API when the data size reaches a certain threshold. When possible, the `TransferManager` uses multiple threads to concurrently upload the parts. If a part upload fails, the API retries the failed part upload up to three times. However, these are configurable options using the `TransferManagerConfiguration` class.

Note

When you're using a stream for the source of data, the `TransferManager` class does not do concurrent uploads.

The Java API Organization

The following packages in the AWS SDK for Java provide the API:

- **com.amazonaws.services.s3**—Provides the APIs for creating Amazon S3 clients and working with buckets and objects. For example, it enables you to create buckets, upload objects, get objects, delete objects, and list keys.
- **com.amazonaws.services.s3.transfer**—Provides the high-level API data operations.

This high-level API is designed to simplify transferring objects to and from Amazon S3. It includes the `TransferManager` class, which provides asynchronous methods for working with, querying, and manipulating transfers. It also includes the `TransferManagerConfiguration` class, which you can use to configure the minimum part size for uploading parts and the threshold in bytes of when to use multipart uploads.

- **com.amazonaws.services.s3.model**—Provides the low-level API classes to create requests and process responses. For example, it includes the `GetObjectRequest` class to describe your get object request, the `ListObjectsRequest` class to describe your list keys requests, and the `InitiateMultipartUploadRequest` class to create multipart uploads.

For more information about the AWS SDK for Java API, see [AWS SDK for Java API Reference](#).

Testing the Amazon S3 Java Code Examples

The Java examples in this guide are compatible with the AWS SDK for Java version 1.11.321. For instructions on setting up and running code samples, see [Getting Started](#) in the AWS SDK for Java Developer Guide.

Using the AWS SDK for .NET

The AWS SDK for .NET provides the API for the Amazon S3 bucket and object operations. For object operations, in addition to providing the API to upload objects in a single operation, the SDK provides the API to upload large objects in parts (see [Uploading and copying objects using multipart upload \(p. 167\)](#)).

Topics

- [The .NET API Organization \(p. 1193\)](#)
- [Running the Amazon S3 .NET Code Examples \(p. 1193\)](#)

The AWS SDK for .NET gives you the option of using a high-level or low-level API.

Low-Level API

The low-level APIs correspond to the underlying Amazon S3 REST operations, including the create, update, and delete operations that apply to buckets and objects. When you upload large objects using the low-level multipart upload API (see [Uploading and copying objects using multipart upload \(p. 167\)](#)), it provides greater control. For example, it lets you pause and resume multipart uploads, vary part sizes during the upload, or begin uploads when you don't know the size of the data in advance. If you do not have these requirements, use the high-level API for uploading objects.

High-Level API

For uploading objects, the SDK provides a higher level of abstraction by providing the `TransferUtility` class. The high-level API is a simpler API, where in just a few lines of code, you can upload files and streams to Amazon S3. You should use this API to upload data unless you need to control the upload as described in the preceding Low-Level API section.

For smaller data size, the `TransferUtility` API uploads data in a single operation. However, the `TransferUtility` switches to using the multipart upload API when the data size reaches a certain threshold. By default, it uses multiple threads to concurrently upload the parts. If a part upload fails, the API retries the failed part upload up to three times. However, these are configurable options.

Note

When you're using a stream for the source of data, the `TransferUtility` class does not do concurrent uploads.

The .NET API Organization

When writing Amazon S3 applications using the AWS SDK for .NET, you use the `AWSSDK.dll`. The following namespaces in this assembly provide the multipart upload API:

- **Amazon.S3.Transfer**—Provides the high-level API to upload your data in parts.

It includes the `TransferUtility` class that enables you to specify a file, directory, or stream for uploading your data. It also includes the `TransferUtilityUploadRequest` and `TransferUtilityUploadDirectoryRequest` classes to configure advanced settings, such as the number of concurrent threads, part size, object metadata, the storage class (STANDARD, REDUCED_REDUNDANCY), and object access control list (ACL).

- **Amazon.S3**—Provides the implementation for the low-level APIs.

It provides methods that correspond to the Amazon S3 REST multipart upload API (see [Using the REST API \(p. 186\)](#)).

- **Amazon.S3.Model**—Provides the low-level API classes to create requests and process responses. For example, it provides the `InitiateMultipartUploadRequest` and `InitiateMultipartUploadResponse` classes you can use when initiating a multipart upload, and the `UploadPartRequest` and `UploadPartResponse` classes when uploading parts.
- **Amazon.S3.Encryption**— Provides `AmazonS3EncryptionClient`.
- **Amazon.S3.Util**— Provides various utility classes such as `AmazonS3Util` and `BucketRegionDetector`.

For more information about the AWS SDK for .NET API, see [AWS SDK for .NET Version 3 API Reference](#).

Running the Amazon S3 .NET Code Examples

The .NET code examples in this guide are compatible with the AWS SDK for .NET version 3.0. For information about setting up and running the code examples, see [Getting Started with the AWS SDK for .NET](#) in the *AWS SDK for .NET Developer Guide*.

Using the AWS SDK for PHP and Running PHP Examples

The AWS SDK for PHP provides access to the API for Amazon S3 bucket and object operations. The SDK gives you the option of using the service's low-level API or using higher-level abstractions.

The SDK is available at [AWS SDK for PHP](#), which also has instructions for installing and getting started with the SDK.

The setup for using the AWS SDK for PHP depends on your environment and how you want to run your application. To set up your environment to run the examples in this documentation, see the [AWS SDK for PHP Getting Started Guide](#).

Topics

- [AWS SDK for PHP Levels \(p. 1194\)](#)
- [Running PHP Examples \(p. 1194\)](#)
- [Related Resources \(p. 1194\)](#)

AWS SDK for PHP Levels

The AWS SDK for PHP gives you the option of using a high-level or low-level API.

Low-Level API

The low-level APIs correspond to the underlying Amazon S3 REST operations, including the create, update, and delete operations on buckets and objects. The low-level APIs provide greater control over these operations. For example, you can batch your requests and run them in parallel. Or, when using the multipart upload API, you can manage the object parts individually. Note that these low-level API calls return a result that includes all of the Amazon S3 response details. For more information about the multipart upload API, see [Uploading and copying objects using multipart upload \(p. 167\)](#).

High-Level Abstractions

The high-level abstractions are intended to simplify common use cases. For example, for uploading large objects using the low-level API, you call `Aws\S3\S3Client::createMultipartUpload()`, call the `Aws\S3\S3Client::uploadPart()` method to upload the object parts, then call the `Aws\S3\S3Client::completeMultipartUpload()` method to complete the upload. You can use the higher-level `Aws\S3\MultipartUploader` object that simplifies creating a multipart upload instead.

As another example, when enumerating objects in a bucket, you can use the iterators feature of the AWS SDK for PHP to return all of the object keys, regardless of how many objects you have stored in the bucket. If you use the low-level API, the response returns a maximum of 1,000 keys. If a bucket contains more than 1,000 objects, the result is truncated and you have to manage the response and check for truncation.

Running PHP Examples

To set up and use the Amazon S3 samples for version 3 of the AWS SDK for PHP, see [Installation](#) in the AWS SDK for PHP Developer Guide.

Related Resources

- [AWS SDK for PHP for Amazon S3](#)
- [AWS SDK for PHP Documentation](#)
- [AWS SDK for PHP API for Amazon S3](#)
- [AWS SDK for PHP Version 3 Code Examples](#)

Using the AWS SDK for Ruby - Version 3

The AWS SDK for Ruby provides an API for Amazon S3 bucket and object operations. For object operations, you can use the API to upload objects in a single operation or upload large objects in parts (see [Uploading an object using multipart upload \(p. 174\)](#)). However, the API for a single operation upload can also accept large objects and behind the scenes manage the upload in parts for you, thereby reducing the amount of script you need to write.

The Ruby API Organization

When creating Amazon S3 applications using the AWS SDK for Ruby, you must install the SDK for Ruby gem. For more information, see the [AWS SDK for Ruby - Version 3](#). Once installed, you can access the API, including the following key classes:

- **Aws::S3::Resource**—Represents the interface to Amazon S3 for the Ruby SDK and provides methods for creating and enumerating buckets.

The `s3` class provides the `#buckets` instance method for accessing existing buckets or creating new ones.

- **Aws::S3::Bucket**—Represents an Amazon S3 bucket.

The `Bucket` class provides the `#object(key)` and `#objects` methods for accessing the objects in a bucket, as well as methods to delete a bucket and return information about a bucket, like the bucket policy.

- **Aws::S3::Object**—Represents an Amazon S3 object identified by its key.

The `Object` class provides methods for getting and setting properties of an object, specifying the storage class for storing objects, and setting object permissions using access control lists. The `Object` class also has methods for deleting, uploading and copying objects. When uploading objects in parts, this class provides options for you to specify the order of parts uploaded and the part size.

For more information about the AWS SDK for Ruby API, go to [AWS SDK for Ruby - Version 2](#).

Testing the Ruby Script Examples

The easiest way to get started with the Ruby script examples is to install the latest AWS SDK for Ruby gem. For information about installing or updating to the latest gem, go to [AWS SDK for Ruby - Version 3](#). The following tasks guide you through the creation and testing of the Ruby script examples assuming that you have installed the AWS SDK for Ruby.

General Process of Creating and Testing Ruby Script Examples

1	To access AWS, you must provide a set of credentials for your SDK for Ruby application. For more information, see Configuring the AWS SDK for Ruby .
2	Create a new SDK for Ruby script and add the following lines to the top of the script. <code>#!/usr/bin/env ruby</code> <code>require 'rubygems'</code> <code>require 'aws-sdk-s3'</code>
	The first line is the interpreter directive and the two <code>require</code> statements import two required gems into your script.
3	Copy the code from the section you are reading to your script.
4	Update the code by providing any required data. For example, if uploading a file, provide the file path and the bucket name.
5	Run the script. Verify changes to buckets and objects by using the AWS Management Console. For more information about the AWS Management Console, go to https://aws.amazon.com/console/ .

Ruby Samples

The following links contain samples to help get you started with the SDK for Ruby - Version 3:

- [Creating a bucket \(p. 119\)](#)
- [Uploading objects \(p. 158\)](#)

Using the AWS SDK for Python (Boto)

Boto is a Python package that provides interfaces to AWS including Amazon S3. For more information about Boto, go to the [AWS SDK for Python \(Boto\)](#). The getting started link on this page provides step-by-step instructions to get started.

Using the AWS Mobile SDKs for iOS and Android

You can use the AWS Mobile SDKs for [Android](#) and [iOS](#) to quickly and easily integrate robust cloud backends into your existing mobile apps. You can configure and use features like user sign-in, databases, push notifications, and more, without being an AWS expert.

The AWS Mobile SDKs provide easy access to Amazon S3 and many other AWS services. To get started using the AWS Mobile SDKs, see [Getting Started with the AWS Mobile SDKs](#).

More Info

[Using the AWS Amplify JavaScript Library \(p. 1196\)](#)

Using the AWS Amplify JavaScript Library

AWS Amplify is an open source JavaScript library for web and mobile developers who build cloud-enabled applications. AWS Amplify provides customizable UI components and a declarative interface to work with an S3 bucket, along with other high-level categories for AWS services.

To get started using the AWS Amplify JavaScript library, choose one of the following links:

- [Getting Started with the AWS Amplify Library for the Web](#)
- [Getting Started with Amplify](#)

For more information about AWS Amplify, see [AWS Amplify](#) on GitHub.

More Info

[Using the AWS Mobile SDKs for iOS and Android \(p. 1196\)](#)

Using the AWS SDK for JavaScript

The AWS SDK for JavaScript provides a JavaScript API for AWS services. You can use the JavaScript API to build libraries or applications for Node.js or the browser.

For more information about using the AWS SDK for JavaScript for Amazon S3, see below.

- [What is the AWS SDK for JavaScript? \(v2\)](#)

- [AWS SDK for JavaScript - Amazon S3 examples \(v2\)](#)
- [What is the AWS SDK for JavaScript? \(v3\)](#)
- [AWS SDK for JavaScript - Amazon S3 examples \(v3\)](#)
- [AWS SDK for JavaScript for Amazon S3](#)

Developing with Amazon S3 using the REST API

The Amazon S3 architecture is designed to be programming language-neutral, using our supported interfaces to store and retrieve objects.

Amazon S3 currently provides a REST interface. With REST, metadata is returned in HTTP headers. Because we only support HTTP requests of up to 4 KB (not including the body), the amount of metadata you can supply is restricted. The REST API is an HTTP interface to Amazon S3. Using REST, you use standard HTTP requests to create, fetch, and delete buckets and objects.

You can use any toolkit that supports HTTP to use the REST API. You can even use a browser to fetch objects, as long as they are anonymously readable.

The REST API uses the standard HTTP headers and status codes, so that standard browsers and toolkits work as expected. In some areas, we have added functionality to HTTP (for example, we added headers to support access control). In these cases, we have done our best to add the new functionality in a way that matched the style of standard HTTP usage.

For more information about sending requests using the REST API, see [Making requests using the REST API \(p. 1174\)](#). For some considerations you should keep in mind when using the REST API, see the topics below.

Topics

- [Request routing \(p. 1197\)](#)

Request routing

Programs that make requests against buckets created using the [CreateBucket](#) API that include a [CreateBucketConfiguration](#) must support redirects. Additionally, some clients that do not respect DNS TTLs might encounter issues.

This section describes routing and DNS issues to consider when designing your service or application for use with Amazon S3.

Request redirection and the REST API

Amazon S3 uses the Domain Name System (DNS) to route requests to facilities that can process them. This system works effectively, but temporary routing errors can occur. If a request arrives at the wrong Amazon S3 location, Amazon S3 responds with a temporary redirect that tells the requester to resend the request to a new endpoint. If a request is incorrectly formed, Amazon S3 uses permanent redirects to provide direction on how to perform the request correctly.

Important

To use this feature, you must have an application that can handle Amazon S3 redirect responses. The only exception is for applications that work exclusively with buckets that were created without `<CreateBucketConfiguration>`. For more information about location constraints, see [Methods for accessing a bucket \(p. 125\)](#).

For all Regions that launched after March 20, 2019, if a request arrives at the wrong Amazon S3 location, Amazon S3 returns an HTTP 400 Bad Request error.

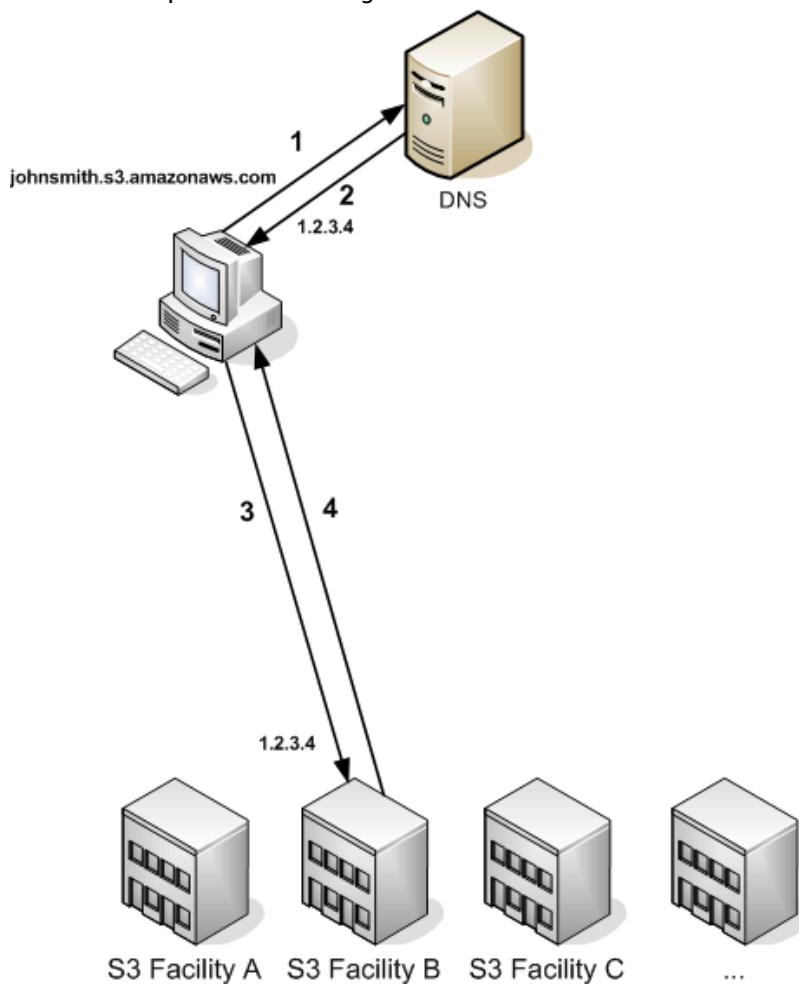
For more information about enabling or disabling an AWS Region, see [AWS Regions and Endpoints](#) in the [AWS General Reference](#).

Topics

- [DNS routing \(p. 1198\)](#)
- [Temporary request redirection \(p. 1199\)](#)
- [Permanent request redirection \(p. 1200\)](#)
- [Request redirection examples \(p. 1200\)](#)

DNS routing

DNS routing routes requests to appropriate Amazon S3 facilities. The following figure and procedure show an example of DNS routing.



DNS routing request steps

1. The client makes a DNS request to get an object stored on Amazon S3.
2. The client receives one or more IP addresses for facilities that can process the request. In this example, the IP address is for Facility B.
3. The client makes a request to Amazon S3 Facility B.

4. Facility B returns a copy of the object to the client.

Temporary request redirection

A temporary redirect is a type of error response that signals to the requester that they should resend the request to a different endpoint. Due to the distributed nature of Amazon S3, requests can be temporarily routed to the wrong facility. This is most likely to occur immediately after buckets are created or deleted.

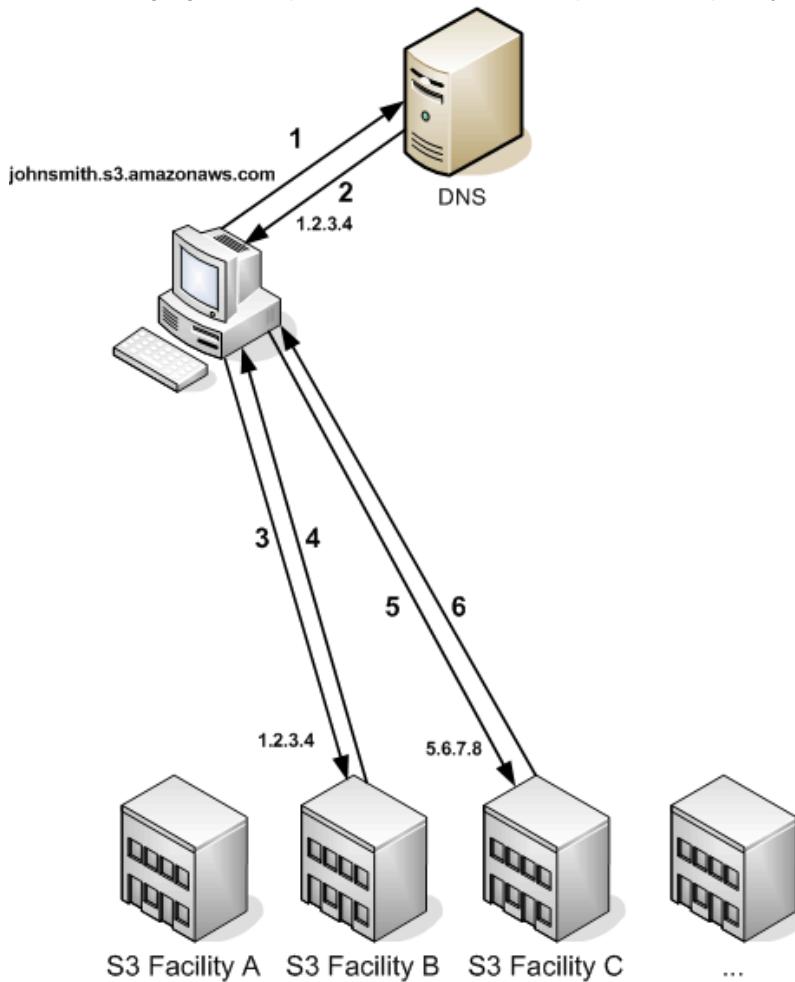
For example, if you create a new bucket and immediately make a request to the bucket, you might receive a temporary redirect, depending on the location constraint of the bucket. If you created the bucket in the US East (N. Virginia) AWS Region, you will not see the redirect because this is also the default Amazon S3 endpoint.

However, if the bucket is created in any other Region, any requests for the bucket go to the default endpoint while the bucket's DNS entry is propagated. The default endpoint redirects the request to the correct endpoint with an HTTP 302 response. Temporary redirects contain a URI to the correct facility, which you can use to immediately resend the request.

Important

Don't reuse an endpoint provided by a previous redirect response. It might appear to work (even for long periods of time), but it might provide unpredictable results and will eventually fail without notice.

The following figure and procedure shows an example of a temporary redirect.



Temporary request redirection steps

1. The client makes a DNS request to get an object stored on Amazon S3.
2. The client receives one or more IP addresses for facilities that can process the request.
3. The client makes a request to Amazon S3 Facility B.
4. Facility B returns a redirect indicating the object is available from Location C.
5. The client resends the request to Facility C.
6. Facility C returns a copy of the object.

Permanent request redirection

A permanent redirect indicates that your request addressed a resource inappropriately. For example, permanent redirects occur if you use a path-style request to access a bucket that was created using <CreateBucketConfiguration>. For more information, see [Methods for accessing a bucket \(p. 125\)](#).

To help you find these errors during development, this type of redirect does not contain a Location HTTP header that allows you to automatically follow the request to the correct location. Consult the resulting XML error document for help using the correct Amazon S3 endpoint.

Request redirection examples

The following are examples of temporary request redirection responses.

REST API temporary redirect response

```
HTTP/1.1 307 Temporary Redirect
Location: http://awsexamplebucket1.s3-gztb4pa9sq.amazonaws.com/photos/puppy.jpg?rk=e2c69a31
Content-Type: application/xml
Transfer-Encoding: chunked
Date: Fri, 12 Oct 2007 01:12:56 GMT
Server: AmazonS3

<?xml version="1.0" encoding="UTF-8"?>
<Error>
  <Code>TemporaryRedirect</Code>
  <Message>Please re-send this request to the specified temporary endpoint.
  Continue to use the original request endpoint for future requests.</Message>
  <Endpoint>awsexamplebucket1.s3-gztb4pa9sq.amazonaws.com</Endpoint>
</Error>
```

SOAP API temporary redirect response

Note

SOAP support over HTTP is deprecated, but SOAP is still available over HTTPS. New Amazon S3 features are not supported for SOAP. Instead of using SOAP, we recommend that you use either the REST API or the AWS SDKs.

```
<soapenv:Body>
<soapenv:Fault>
  <Faultcode>soapenv:Client.TemporaryRedirect</Faultcode>
  <Faultstring>Please re-send this request to the specified temporary endpoint.
  Continue to use the original request endpoint for future requests.</Faultstring>
  <Detail>
    <Bucket>images</Bucket>
    <Endpoint>s3-gztb4pa9sq.amazonaws.com</Endpoint>
  </Detail>
</soapenv:Fault>
```

```
</soapenv:Body>
```

DNS considerations

One of the design requirements of Amazon S3 is extremely high availability. One of the ways we meet this requirement is by updating the IP addresses associated with the Amazon S3 endpoint in DNS as needed. These changes are automatically reflected in short-lived clients, but not in some long-lived clients. Long-lived clients will need to take special action to re-resolve the Amazon S3 endpoint periodically to benefit from these changes. For more information about virtual machines (VMs), refer to the following:

- For Java, Sun's JVM caches DNS lookups forever by default; go to the "InetAddress Caching" section of the [InetAddress documentation](#) for information on how to change this behavior.
- For PHP, the persistent PHP VM that runs in the most popular deployment configurations caches DNS lookups until the VM is restarted. Go to the [getHostByName PHP docs](#).

Handling REST and SOAP errors

Topics

- [The REST error response \(p. 1201\)](#)
- [The SOAP error response \(p. 1202\)](#)
- [Amazon S3 error best practices \(p. 1203\)](#)

This section describes REST and SOAP errors and how to handle them.

Note

SOAP support over HTTP is deprecated, but SOAP is still available over HTTPS. New Amazon S3 features are not supported for SOAP. Instead of using SOAP, we recommend that you use either the REST API or the AWS SDKs.

The REST error response

If a REST request results in an error, the HTTP reply has:

- An XML error document as the response body
- Content-Type: application/xml
- An appropriate 3xx, 4xx, or 5xx HTTP status code

Following is an example of a REST Error Response.

```
<?xml version="1.0" encoding="UTF-8"?>
<Error>
<Code>NoSuchKey</Code>
<Message>The resource you requested does not exist</Message>
<Resource>/mybucket/myfoto.jpg</Resource>
<RequestId>4442587FB7D0A2F9</RequestId>
</Error>
```

For more information about Amazon S3 errors, go to [ErrorCodeList](#).

Response headers

Following are response headers returned by all operations:

- **x-amz-request-id**: A unique ID assigned to each request by the system. In the unlikely event that you have problems with Amazon S3, Amazon can use this to help troubleshoot the problem.
- **x-amz-id-2**: A special token that will help us to troubleshoot problems.

Error response

When an Amazon S3 request is in error, the client receives an error response. The exact format of the error response is API specific: For example, the REST error response differs from the SOAP error response. However, all error responses have common elements.

Note

SOAP support over HTTP is deprecated, but SOAP is still available over HTTPS. New Amazon S3 features are not supported for SOAP. Instead of using SOAP, we recommend that you use either the REST API or the AWS SDKs.

Error code

The error code is a string that uniquely identifies an error condition. It is meant to be read and understood by programs that detect and handle errors by type. Many error codes are common across SOAP and REST APIs, but some are API-specific. For example, `NoSuchKey` is universal, but `UnexpectedContent` can occur only in response to an invalid REST request. In all cases, SOAP fault codes carry a prefix as indicated in the table of error codes, so that a `NoSuchKey` error is actually returned in SOAP as `Client.NoSuchKey`.

Note

SOAP support over HTTP is deprecated, but SOAP is still available over HTTPS. New Amazon S3 features are not supported for SOAP. Instead of using SOAP, we recommend that you use either the REST API or the AWS SDKs.

Error message

The error message contains a generic description of the error condition in English. It is intended for a human audience. Simple programs display the message directly to the end user if they encounter an error condition they don't know how or don't care to handle. Sophisticated programs with more exhaustive error handling and proper internationalization are more likely to ignore the error message.

Further details

Many error responses contain additional structured data meant to be read and understood by a developer diagnosing programming errors. For example, if you send a `Content-MD5` header with a REST `PUT` request that doesn't match the digest calculated on the server, you receive a `BadDigest` error. The error response also includes as detail elements the digest we calculated, and the digest you told us to expect. During development, you can use this information to diagnose the error. In production, a well-behaved program might include this information in its error log.

The SOAP error response

Note

SOAP support over HTTP is deprecated, but SOAP is still available over HTTPS. New Amazon S3 features are not supported for SOAP. Instead of using SOAP, we recommend that you use either the REST API or the AWS SDKs.

In SOAP, an error result is returned to the client as a SOAP fault, with the HTTP response code 500. If you do not receive a SOAP fault, then your request was successful. The Amazon S3 SOAP fault code is comprised of a standard SOAP 1.1 fault code (either "Server" or "Client") concatenated with the Amazon S3-specific error code. For example: "Server.InternalError" or "Client.NoSuchBucket". The SOAP fault string element contains a generic, human readable error message in English. Finally, the SOAP fault detail element contains miscellaneous information relevant to the error.

For example, if you attempt to delete the object "Fred", which does not exist, the body of the SOAP response contains a "NoSuchKey" SOAP fault.

Example

```
<soapenv:Body>
  <soapenv:Fault>
    <Faultcode>soapenv:Client.NoSuchKey</Faultcode>
    <Faultstring>The specified key does not exist.</Faultstring>
    <Detail>
      <Key>Fred</Key>
    </Detail>
  </soapenv:Fault>
</soapenv:Body>
```

For more information about Amazon S3 errors, go to [ErrorCodeList](#).

Amazon S3 error best practices

When designing an application for use with Amazon S3, it is important to handle Amazon S3 errors appropriately. This section describes issues to consider when designing your application.

Retry InternalErrors

Internal errors are errors that occur within the Amazon S3 environment.

Requests that receive an InternalError response might not have processed. For example, if a PUT request returns InternalError, a subsequent GET might retrieve the old value or the updated value.

If Amazon S3 returns an InternalError response, retry the request.

Tune application for repeated SlowDown errors

As with any distributed system, S3 has protection mechanisms which detect intentional or unintentional resource over-consumption and react accordingly. SlowDown errors can occur when a high request rate triggers one of these mechanisms. Reducing your request rate will decrease or eliminate errors of this type. Generally speaking, most users will not experience these errors regularly; however, if you would like more information or are experiencing high or unexpected SlowDown errors, please post to our [Amazon S3 developer forum](#) or sign up for AWS Support <https://aws.amazon.com/premiumsupport/>.

Isolate errors

Note

SOAP support over HTTP is deprecated, but SOAP is still available over HTTPS. New Amazon S3 features are not supported for SOAP. Instead of using SOAP, we recommend that you use either the REST API or the AWS SDKs.

Amazon S3 provides a set of error codes that are used by both the SOAP and REST API. The SOAP API returns standard Amazon S3 error codes. The REST API is designed to look like a standard HTTP server and interact with existing HTTP clients (e.g., browsers, HTTP client libraries, proxies, caches, and so on). To ensure the HTTP clients handle errors properly, we map each Amazon S3 error to an HTTP status code.

HTTP status codes are less expressive than Amazon S3 error codes and contain less information about the error. For example, the `NoSuchKey` and `NoSuchBucket` Amazon S3 errors both map to the `HTTP 404 Not Found` status code.

Although the HTTP status codes contain less information about the error, clients that understand HTTP, but not the Amazon S3 API, will usually handle the error correctly.

Therefore, when handling errors or reporting Amazon S3 errors to end users, use the Amazon S3 error code instead of the HTTP status code as it contains the most information about the error. Additionally, when debugging your application, you should also consult the human readable <Details> element of the XML error response.

Developer reference

This appendix include the following sections.

Topics

- [Appendix a: Using the SOAP API \(p. 1204\)](#)
- [Appendix b: Authenticating requests \(AWS signature version 2\) \(p. 1207\)](#)

Appendix a: Using the SOAP API

Note

SOAP support over HTTP is deprecated, but SOAP is still available over HTTPS. New Amazon S3 features are not supported for SOAP. Instead of using SOAP, we recommend that you use either the REST API or the AWS SDKs.

This section contains information specific to the Amazon S3 SOAP API.

Note

SOAP requests, both authenticated and anonymous, must be sent to Amazon S3 using SSL. Amazon S3 returns an error when you send a SOAP request over HTTP.

Topics

- [Common SOAP API elements \(p. 1204\)](#)
- [Authenticating SOAP requests \(p. 1205\)](#)
- [Setting access policy with SOAP \(p. 1205\)](#)

Common SOAP API elements

Note

SOAP support over HTTP is deprecated, but SOAP is still available over HTTPS. New Amazon S3 features are not supported for SOAP. Instead of using SOAP, we recommend that you use either the REST API or the AWS SDKs.

You can interact with Amazon S3 using SOAP 1.1 over HTTP. The Amazon S3 WSDL, which describes the Amazon S3 API in a machine-readable way, is available at: <https://doc.s3.amazonaws.com/2006-03-01/AmazonS3.wsdl>. The Amazon S3 schema is available at <https://doc.s3.amazonaws.com/2006-03-01/AmazonS3.xsd>.

Most users will interact with Amazon S3 using a SOAP toolkit tailored for their language and development environment. Different toolkits will expose the Amazon S3 API in different ways. Please refer to your specific toolkit documentation to understand how to use it. This section illustrates the Amazon S3 SOAP operations in a toolkit-independent way by exhibiting the XML requests and responses as they appear "on the wire."

Common elements

You can include the following authorization-related elements with any SOAP request:

- **AWSAccessKeyId:** The AWS Access Key ID of the requester

- **Timestamp**: The current time on your system
- **Signature**: The signature for the request

Authenticating SOAP requests

Note

SOAP support over HTTP is deprecated, but SOAP is still available over HTTPS. New Amazon S3 features are not supported for SOAP. Instead of using SOAP, we recommend that you use either the REST API or the AWS SDKs.

Every non-anonymous request must contain authentication information to establish the identity of the principal making the request. In SOAP, the authentication information is put into the following elements of the SOAP request:

- Your AWS Access Key ID

Note

When making authenticated SOAP requests, temporary security credentials are not supported. For more information about types of credentials, see [Making requests \(p. 1138\)](#).

- **Timestamp**: This must be a `dateTime` (go to <http://www.w3.org/TR/xmlschema-2/#dateTime>) in the Coordinated Universal Time (Greenwich Mean Time) time zone, such as `2009-01-01T12:00:00.000Z`. Authorization will fail if this timestamp is more than 15 minutes away from the clock on Amazon S3 servers.
- **Signature**: The RFC 2104 HMAC-SHA1 digest (go to <http://www.ietf.org/rfc/rfc2104.txt>) of the concatenation of "AmazonS3" + OPERATION + Timestamp, using your AWS Secret Access Key as the key. For example, in the following CreateBucket sample request, the signature element would contain the HMAC-SHA1 digest of the value "AmazonS3CreateBucket2009-01-01T12:00:00.000Z":

For example, in the following CreateBucket sample request, the signature element would contain the HMAC-SHA1 digest of the value "AmazonS3CreateBucket2009-01-01T12:00:00.000Z":

Example

```
<CreateBucket xmlns="https://doc.s3.amazonaws.com/2006-03-01">
  <Bucket>quotes</Bucket>
  <Acl>private</Acl>
  <AWSAccessKeyId>AKIAIOSFODNN7EXAMPLE</AWSAccessKeyId>
  <Timestamp>2009-01-01T12:00:00.000Z</Timestamp>
  <Signature>Iuyz3d3P0aTou39dzbqaEXAMPLE=</Signature>
</CreateBucket>
```

Note

SOAP requests, both authenticated and anonymous, must be sent to Amazon S3 using SSL. Amazon S3 returns an error when you send a SOAP request over HTTP.

Important

Due to different interpretations regarding how extra time precision should be dropped, .NET users should take care not to send Amazon S3 overly specific time stamps. This can be accomplished by manually constructing `DateTime` objects with only millisecond precision.

Setting access policy with SOAP

Note

SOAP support over HTTP is deprecated, but SOAP is still available over HTTPS. New Amazon S3 features are not supported for SOAP. Instead of using SOAP, we recommend that you use either the REST API or the AWS SDKs.

Access control can be set at the time a bucket or object is written by including the "AccessControlList" element with the request to `CreateBucket`, `PutObjectInline`, or `PutObject`. The `AccessControlList` element is described in [Identity and access management in Amazon S3 \(p. 394\)](#). If no access control list is specified with these operations, the resource is created with a default access policy that gives the requester `FULL_CONTROL` access (this is the case even if the request is a `PutObjectInline` or `PutObject` request for an object that already exists).

Following is a request that writes data to an object, makes the object readable by anonymous principals, and gives the specified user `FULL_CONTROL` rights to the bucket (Most developers will want to give themselves `FULL_CONTROL` access to their own bucket).

Example

Following is a request that writes data to an object and makes the object readable by anonymous principals.

Sample Request

```
<PutObjectInline xmlns="https://doc.s3.amazonaws.com/2006-03-01">
  <Bucket>quotes</Bucket>
  <Key>Nelson</Key>
  <Metadata>
    <Name>Content-Type</Name>
    <Value>text/plain</Value>
  </Metadata>
  <Data>aGETaGE=</Data>
  <ContentLength>5</ContentLength>
  <AccessControlList>
    <Grant>
      <Grantee xsi:type="CanonicalUser">
        <ID>75cc57f09aa0c8caeab4f8c24e99d10f8e7faeef76c078efc7c6caea54ba06a</ID>
        <DisplayName>chriscustomer</DisplayName>
      </Grantee>
      <Permission>FULL_CONTROL</Permission>
    </Grant>
    <Grant>
      <Grantee xsi:type="Group">
        <URI>http://acs.amazonaws.com/groups/global/AllUsers<URI>
      </Grantee>
      <Permission>READ</Permission>
    </Grant>
  </AccessControlList>
  <AWSAccessKeyId>AKIAIOSFODNN7EXAMPLE</AWSAccessKeyId>
  <Timestamp>2009-03-01T12:00:00.183Z</Timestamp>
  <Signature>Iuyz3d3P0aTou39dzbqaEXAMPLE=</Signature>
</PutObjectInline>
```

Sample Response

```
<PutObjectInlineResponse xmlns="https://s3.amazonaws.com/doc/2006-03-01">
  <PutObjectInlineResponse>
    <ETag>"828ef3fd96f00ad9f27c383fc9ac7f"</ETag>
    <LastModified>2009-01-01T12:00:00.000Z</LastModified>
  </PutObjectInlineResponse>
</PutObjectInlineResponse>
```

The access control policy can be read or set for an existing bucket or object using the `GetBucketAccessControlPolicy`, `GetObjectAccessControlPolicy`, `SetBucketAccessControlPolicy`, and `SetObjectAccessControlPolicy` methods. For more information, see the detailed explanation of these methods.

Appendix b: Authenticating requests (AWS signature version 2)

Important

This section describes how to authenticate requests using AWS Signature Version 2. Signature Version 2 is being turned off (deprecated), Amazon S3 will only accept API requests that are signed using Signature Version 4. For more information, see [AWS Signature Version 2 Turned Off \(Deprecated\) for Amazon S3 \(p. 1187\)](#)

Signature Version 4 is supported in all AWS Regions, and it is the only version that is supported for new Regions. For more information, see [Authenticating Requests \(AWS Signature Version 4\) in the Amazon Simple Storage Service API Reference](#).

Amazon S3 offers you the ability to identify what API signature version was used to sign a request. It is important to identify if any of your workflows are utilizing Signature Version 2 signing and upgrading them to use Signature Version 4 to prevent impact to your business.

- If you are using CloudTrail event logs(recommended option), please see [Identifying Amazon S3 Signature Version 2 requests using CloudTrail \(p. 974\)](#) on how to query and identify such requests.
- If you are using the Amazon S3 Server Access logs, see [Identifying Signature Version 2 requests using Amazon S3 access logs \(p. 1001\)](#)

Topics

- [Authenticating requests using the REST API \(p. 1208\)](#)
- [Signing and authenticating REST requests \(p. 1210\)](#)
- [Browser-based uploads using POST \(AWS signature version 2\) \(p. 1219\)](#)

Authenticating requests using the REST API

When accessing Amazon S3 using REST, you must provide the following items in your request so the request can be authenticated:

Request elements

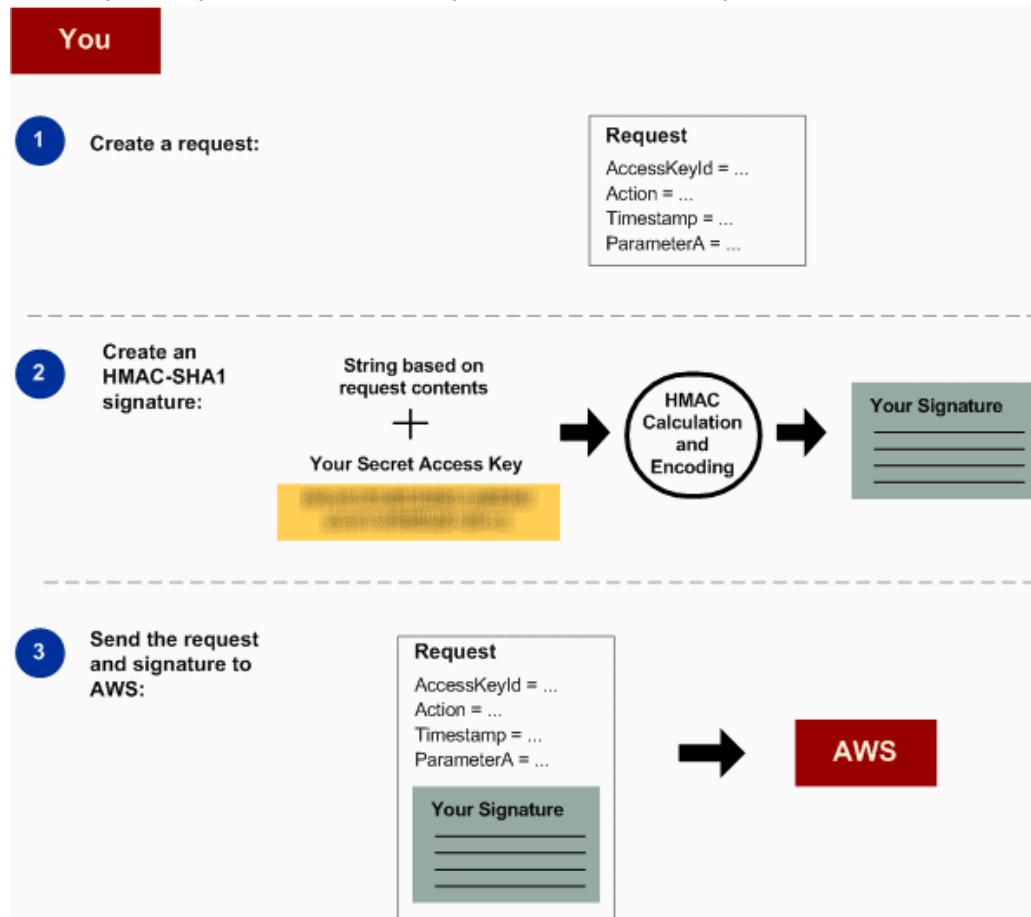
- **AWS access key Id** – Each request must contain the access key ID of the identity you are using to send your request.
- **Signature** – Each request must contain a valid request signature, or the request is rejected.

A request signature is calculated using your secret access key, which is a shared secret known only to you and

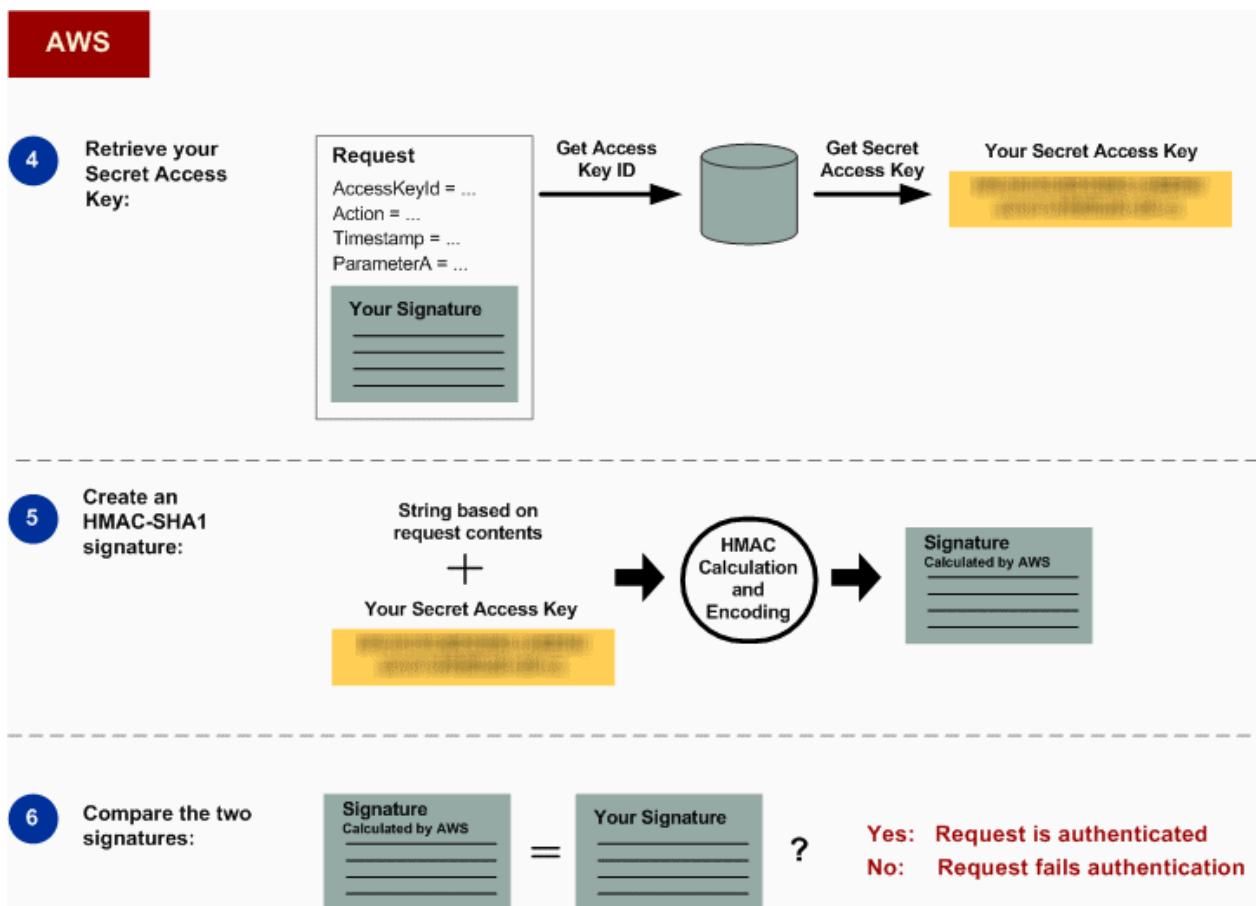
- **Time stamp** – Each request must contain the date and time the request was created, represented as a string in UTC.
- **Date** – Each request must contain the time stamp of the request.

Depending on the API action you're using, you can provide an expiration date and time for the request instead of or in addition to the time stamp. See the authentication topic for the particular action to determine what it requires.

Following are the general steps for authenticating requests to Amazon S3. It is assumed you have the necessary security credentials, access key ID and secret access key.



1	Construct a request to
2	Calculate the signature using your secret access key.
3	Send the request to Amazon S3. Include your access key ID and the signature in your request. Amazon S3 performs the next three steps.



4	Amazon S3 uses the access key ID to look up your secret access key.
5	Amazon S3 calculates a signature from the request data and the secret access key using the same algorithm that you used to calculate the signature you sent in the request.
6	If the signature generated by Amazon S3 matches the one you sent in the request, the request is considered authentic. If the comparison fails, the request is discarded, and Amazon S3 returns an error response.

Detailed authentication information

For detailed information about REST authentication, see [Signing and authenticating REST requests \(p. 1210\)](#).

Signing and authenticating REST requests

Topics

- [Using temporary security credentials \(p. 1211\)](#)
- [The authentication header \(p. 1211\)](#)
- [Request canonicalization for signing \(p. 1212\)](#)
- [Constructing the CanonicalizedResource element \(p. 1212\)](#)
- [Constructing the CanonicalizedAmzHeaders element \(p. 1213\)](#)
- [Positional versus named HTTP header StringToSign elements \(p. 1213\)](#)
- [Time stamp requirement \(p. 1213\)](#)
- [Authentication examples \(p. 1214\)](#)
- [REST request signing problems \(p. 1217\)](#)
- [Query string request authentication alternative \(p. 1218\)](#)

Note

This topic explains authenticating requests using Signature Version 2. Amazon S3 now supports the latest Signature Version 4. This latest signature version is supported in all regions and any new regions after January 30, 2014 will support only Signature Version 4. For more information, go to [Authenticating Requests \(AWS Signature Version 4\)](#) in the *Amazon Simple Storage Service API Reference*.

Authentication is the process of proving your identity to the system. Identity is an important factor in Amazon S3 access control decisions. Requests are allowed or denied in part based on the identity of the requester. For example, the right to create buckets is reserved for registered developers and (by default) the right to create objects in a bucket is reserved for the owner of the bucket in question. As a developer, you'll be making requests that invoke these privileges, so you'll need to prove your identity to the system by authenticating your requests. This section shows you how.

Note

The content in this section does not apply to HTTP POST. For more information, see [Browser-based uploads using POST \(AWS signature version 2\) \(p. 1219\)](#).

The Amazon S3 REST API uses a custom HTTP scheme based on a keyed-HMAC (Hash Message Authentication Code) for authentication. To authenticate a request, you first concatenate selected elements of the request to form a string. You then use your AWS secret access key to calculate the HMAC of that string. Informally, we call this process "signing the request," and we call the output of the HMAC algorithm the signature, because it simulates the security properties of a real signature. Finally, you add this signature as a parameter of the request by using the syntax described in this section.

When the system receives an authenticated request, it fetches the AWS secret access key that you claim to have and uses it in the same way to compute a signature for the message it received. It then compares the signature it calculated against the signature presented by the requester. If the two signatures match, the system concludes that the requester must have access to the AWS secret access key and therefore acts with the authority of the principal to whom the key was issued. If the two signatures do not match, the request is dropped and the system responds with an error message.

Example Authenticated Amazon S3 REST request

```
GET /photos/puppy.jpg HTTP/1.1
Host: awsexamplebucket1.us-west-1.s3.amazonaws.com
Date: Tue, 27 Mar 2007 19:36:42 +0000
```

Authorization: AWS AKIAIOSFODNN7EXAMPLE:

qgk2+6Sv9/oM7G3qLEjTH1a111g=

Using temporary security credentials

If you are signing your request using temporary security credentials (see [Making requests \(p. 1138\)](#)), you must include the corresponding security token in your request by adding the `x-amz-security-token` header.

When you obtain temporary security credentials using the AWS Security Token Service API, the response includes temporary security credentials and a session token. You provide the session token value in the `x-amz-security-token` header when you send requests to Amazon S3. For information about the AWS Security Token Service API provided by IAM, go to [Action in the AWS Security Token Service API Reference Guide](#).

The authentication header

The Amazon S3 REST API uses the standard HTTP Authorization header to pass authentication information. (The name of the standard header is unfortunate because it carries authentication information, not authorization.) Under the Amazon S3 authentication scheme, the Authorization header has the following form:

Authorization: AWS `AWSAccessKeyId:Signature`

Developers are issued an AWS access key ID and AWS secret access key when they register. For request authentication, the `AWSAccessKeyId` element identifies the access key ID that was used to compute the signature and, indirectly, the developer making the request.

The `Signature` element is the RFC 2104 HMAC-SHA1 of selected elements from the request, and so the `Signature` part of the Authorization header will vary from request to request. If the request signature calculated by the system matches the `Signature` included with the request, the requester will have demonstrated possession of the AWS secret access key. The request will then be processed under the identity, and with the authority, of the developer to whom the key was issued.

Following is pseudogrammar that illustrates the construction of the Authorization request header. (In the example, `\n` means the Unicode code point U+000A, commonly called newline).

```
Authorization = "AWS" + " " + AWSAccessKeyId + ":" + Signature;

Signature = Base64( HMAC-SHA1( UTF-8-Encoding-Of(YourSecretAccessKey), UTF-8-Encoding-
Of( StringToSign ) ) );

StringToSign = HTTP-Verb + "\n" +
Content-MD5 + "\n" +
Content-Type + "\n" +
Date + "\n" +
CanonicalizedAmzHeaders +
CanonicalizedResource;

CanonicalizedResource = [ "/" + Bucket ] +
<HTTP-Request-URI, from the protocol name up to the query string> +
[ subresource, if present. For example "?acl", "?location", or "?logging" ];

CanonicalizedAmzHeaders = <described below>
```

HMAC-SHA1 is an algorithm defined by [RFC 2104 - Keyed-Hashing for Message Authentication](#). The algorithm takes as input two byte-strings, a key and a message. For Amazon S3 request authentication,

use your AWS secret access key (`YourSecretAccessKey`) as the key, and the UTF-8 encoding of the `StringToSign` as the message. The output of HMAC-SHA1 is also a byte string, called the digest. The `Signature` request parameter is constructed by Base64 encoding this digest.

Request canonicalization for signing

Recall that when the system receives an authenticated request, it compares the computed request signature with the signature provided in the request in `StringToSign`. For that reason, you must compute the signature by using the same method used by Amazon S3. We call the process of putting a request in an agreed-upon form for signing *canonicalization*.

Constructing the CanonicalizedResource element

`CanonicalizedResource` represents the Amazon S3 resource targeted by the request. Construct it for a REST request as follows:

Launch process

1	Start with an empty string ("").
2	<p>If the request specifies a bucket using the HTTP Host header (virtual hosted-style), append the bucket name preceded by a "/" (e.g., "/bucketname"). For path-style requests and requests that don't address a bucket, do nothing. For more information about virtual hosted-style requests, see Virtual hosting of buckets (p. 1175).</p> <p>For a virtual hosted-style request "https://awsexamplebucket1.s3.us-west-1.amazonaws.com/photos/puppy.jpg", the <code>CanonicalizedResource</code> is "/awsexamplebucket1".</p> <p>For the path-style request, "https://s3.us-west-1.amazonaws.com/awsexamplebucket1/photos/puppy.jpg", the <code>CanonicalizedResource</code> is "".</p>
3	<p>Append the path part of the un-decoded HTTP Request-URI, up-to but not including the query string.</p> <p>For a virtual hosted-style request "https://awsexamplebucket1.s3.us-west-1.amazonaws.com/photos/puppy.jpg", the <code>CanonicalizedResource</code> is "/awsexamplebucket1/photos/puppy.jpg".</p> <p>For a path-style request, "https://s3.us-west-1.amazonaws.com/awsexamplebucket1/photos/puppy.jpg", the <code>CanonicalizedResource</code> is "/awsexamplebucket1/photos/puppy.jpg". At this point, the <code>CanonicalizedResource</code> is the same for both the virtual hosted-style and path-style request.</p> <p>For a request that does not address a bucket, such as GET Service, append "/".</p>
4	<p>If the request addresses a subresource, such as <code>?versioning</code>, <code>?location</code>, <code>?acl</code>, <code>?lifecycle</code>, or <code>?versionid</code>, append the subresource, its value if it has one, and the question mark. Note that in case of multiple subresources, subresources must be lexicographically sorted by subresource name and separated by '&', e.g., <code>?acl&versionId=value</code>.</p> <p>The subresources that must be included when constructing the <code>CanonicalizedResource</code> Element are <code>acl</code>, <code>lifecycle</code>, <code>location</code>, <code>logging</code>, <code>notification</code>, <code>partNumber</code>, <code>policy</code>, <code>requestPayment</code>, <code>uploadId</code>, <code>uploads</code>, <code>versionId</code>, <code>versioning</code>, <code>versions</code>, and <code>website</code>.</p> <p>If the request specifies query string parameters overriding the response header values (see Get Object), append the query string parameters and their values. When signing, you do not encode these values; however, when making the request, you must encode these parameter values. The query string parameters in a GET request include <code>response-content-type</code>, <code>response-content-language</code>, <code>response-expires</code>, <code>response-cache-control</code>, <code>response-content-disposition</code>, and <code>response-content-encoding</code>.</p>

The delete query string parameter must be included when you create the CanonicalizedResource for a multi-object Delete request.

Elements of the CanonicalizedResource that come from the HTTP Request-URI should be signed literally as they appear in the HTTP request, including URL-Encoding meta characters.

The CanonicalizedResource might be different than the HTTP Request-URI. In particular, if your request uses the HTTP Host header to specify a bucket, the bucket does not appear in the HTTP Request-URI. However, the CanonicalizedResource continues to include the bucket. Query string parameters might also appear in the Request-URI but are not included in CanonicalizedResource. For more information, see [Virtual hosting of buckets \(p. 1175\)](#).

Constructing the CanonicalizedAmzHeaders element

To construct the CanonicalizedAmzHeaders part of StringToSign, select all HTTP request headers that start with 'x-amz-' (using a case-insensitive comparison), and use the following process.

CanonicalizedAmzHeaders process

1	Convert each HTTP header name to lowercase. For example, 'x-Amz-Date' becomes 'x-amz-date'.
2	Sort the collection of headers lexicographically by header name.
3	Combine header fields with the same name into one "header-name:comma-separated-value-list" pair as prescribed by RFC 2616, section 4.2, without any spaces between values. For example, the two metadata headers 'x-amz-meta-username: fred' and 'x-amz-meta-username: barney' would be combined into the single header 'x-amz-meta-username: fred,barney'.
4	"Unfold" long headers that span multiple lines (as allowed by RFC 2616, section 4.2) by replacing the folding spaces (including new-line) by a single space.
5	Trim any spaces around the colon in the header. For example, the header 'x-amz-meta-username: fred,barney' would become 'x-amz-meta-username:fred,barney'
6	Finally, append a newline character (U+000A) to each canonicalized header in the resulting list. Construct the CanonicalizedResource element by concatenating all headers in this list into a single string.

Positional versus named HTTP header StringToSign elements

The first few header elements of StringToSign (Content-Type, Date, and Content-MD5) are positional in nature. StringToSign does not include the names of these headers, only their values from the request. In contrast, the 'x-amz-' elements are named. Both the header names and the header values appear in StringToSign.

If a positional header called for in the definition of StringToSign is not present in your request (for example, Content-Type or Content-MD5 are optional for PUT requests and meaningless for GET requests), substitute the empty string ("") for that position.

Time stamp requirement

A valid time stamp (using either the HTTP Date header or an x-amz-date alternative) is mandatory for authenticated requests. Furthermore, the client timestamp included with an authenticated request must be within 15 minutes of the Amazon S3 system time when the request is received. If not, the request

will fail with the `RequestTimeTooSkewed` error code. The intention of these restrictions is to limit the possibility that intercepted requests could be replayed by an adversary. For stronger protection against eavesdropping, use the HTTPS transport for authenticated requests.

Note

The validation constraint on request date applies only to authenticated requests that do not use query string authentication. For more information, see [Query string request authentication alternative \(p. 1218\)](#).

Some HTTP client libraries do not expose the ability to set the `Date` header for a request. If you have trouble including the value of the '`Date`' header in the canonicalized headers, you can set the timestamp for the request by using an '`x-amz-date`' header instead. The value of the `x-amz-date` header must be in one of the RFC 2616 formats (<http://www.ietf.org/rfc/rfc2616.txt>). When an `x-amz-date` header is present in a request, the system will ignore any `Date` header when computing the request signature. Therefore, if you include the `x-amz-date` header, use the empty string for the `Date` when constructing the `StringToSign`. See the next section for an example.

Authentication examples

The examples in this section use the (non-working) credentials in the following table.

Parameter	Value
AWSAccessKeyId	AKIAIOSFODNN7EXAMPLE
AWSSecretAccessKey	wJalrXUtnFEMI/K7MDENG/bPxRfCYEXAMPLEKEY

In the example `StringToSigns`, formatting is not significant, and `\n` means the Unicode code point `U+000A`, commonly called newline. Also, the examples use "+0000" to designate the time zone. You can use "GMT" to designate timezone instead, but the signatures shown in the examples will be different.

Object GET

This example gets an object from the `awsexamplebucket1` bucket.

Request	StringToSign
<code>GET /photos/puppy.jpg HTTP/1.1</code> <code>Host: awsexamplebucket1.us-west-1.s3.amazonaws.com</code> <code>Date: Tue, 27 Mar 2007 19:36:42 +0000</code> <code>Authorization: AWS</code> <code>AKIAIOSFODNN7EXAMPLE:</code> <code>gqk2+6Sv9/oM7G3qLEjTH1a1l1g=</code>	<code>GET\n\n\nTue, 27 Mar 2007 19:36:42 +0000\n/awsexamplebucket1/photos/puppy.jpg</code>

Note that the CanonicalizedResource includes the bucket name, but the HTTP Request-URI does not. (The bucket is specified by the Host header.)

Note

The following Python script calculates the preceding signature, using the provided parameters. You can use this script to construct your own signatures, replacing the keys and `StringToSign` as appropriate.

```
import base64
```

```

import hmac
from hashlib import sha1

access_key = 'AKIAIOSFODNN7EXAMPLE'.encode("UTF-8")
secret_key = 'wJalrXUtnFEMI/K7MDENG/bPxRfiCYEXAMPLEKEY'.encode("UTF-8")

string_to_sign = 'GET\n\n\nTue, 27 Mar 2007 19:36:42 +0000\n/n/awsexamplebucket1/
photos/puppy.jpg'.encode("UTF-8")
signature = base64.b64encode(
    hmac.new(
        secret_key, string_to_sign, sha1
    ).digest()
).strip()

print(f"AWS {access_key.decode()}:{signature.decode()}")

```

Object PUT

This example puts an object into the awsexamplebucket1 bucket.

Request	StringToSign
<pre>PUT /photos/puppy.jpg HTTP/1.1 Content-Type: image/jpeg Content-Length: 94328 Host: awsexamplebucket1.s3.us- west-1.amazonaws.com Date: Tue, 27 Mar 2007 21:15:45 +0000 Authorization: AWS AKIAIOSFODNN7EXAMPLE: iqRzw+ileNPulfhspnRs8nOjjIA=</pre>	<pre>PUT\n \n image/jpeg\n Tue, 27 Mar 2007 21:15:45 +0000\n /awsexamplebucket1/photos/puppy.jpg</pre>

Note the Content-Type header in the request and in the StringToSign. Also note that the Content-MD5 is left blank in the StringToSign, because it is not present in the request.

List

This example lists the content of the awsexamplebucket1 bucket.

Request	StringToSign
<pre>GET /?prefix=photos&max-keys=50&marker=puppy HTTP/1.1 User-Agent: Mozilla/5.0 Host: awsexamplebucket1.s3.us-west-1.amazonaws.com Date: Tue, 27 Mar 2007 19:42:41 +0000 Authorization: AWS AKIAIOSFODNN7EXAMPLE: m0WP8eCtspQl5Ahe6L1SozdX9YA=</pre>	<pre>GET\n \n \n Tue, 27 Mar 2007 19:42:41 +0000\n /awsexamplebucket1/</pre>

Note the trailing slash on the CanonicalizedResource and the absence of query string parameters.

Fetch

This example fetches the access control policy subresource for the 'awsexamplebucket1' bucket.

Request	StringToSign
<pre>GET /?acl HTTP/1.1 Host: awsexamplebucket1.s3.us-west-1.amazonaws.com Date: Tue, 27 Mar 2007 19:44:46 +0000 Authorization: AWS AKIAIOSFODNN7EXAMPLE: 82ZHiFIjc+WbcwFKGUVEQspPn+0=</pre>	<pre>GET\n \n \n Tue, 27 Mar 2007 19:44:46 +0000\n /awsexamplebucket1/?acl</pre>

Notice how the subresource query string parameter is included in the CanonicalizedResource.

Delete

This example deletes an object from the 'awsexamplebucket1' bucket using the path-style and Date alternative.

Request	StringToSign
<pre>DELETE /awsexamplebucket1/photos/puppy.jpg HTTP/1.1 User-Agent: dotnet Host: s3.us-west-1.amazonaws.com Date: Tue, 27 Mar 2007 21:20:27 +0000 x-amz-date: Tue, 27 Mar 2007 21:20:26 +0000 Authorization: AWS AKIAIOSFODNN7EXAMPLE:XbyTlbQdu9Xw5o8P4iMwPktxQd8=</pre>	<pre>DELETE\n \n \n Tue, 27 Mar 2007 21:20:26 +0000\n /awsexamplebucket1/photos/puppy.jpg</pre>

Note how we used the alternate 'x-amz-date' method of specifying the date (because our client library prevented us from setting the date, say). In this case, the x-amz-date takes precedence over the Date header. Therefore, date entry in the signature must contain the value of the x-amz-date header.

Upload

This example uploads an object to a CNAME style virtual hosted bucket with metadata.

Request	StringToSign
<pre>PUT /db-backup.dat.gz HTTP/1.1 User-Agent: curl/7.15.5 Host: static.example.com:8080 Date: Tue, 27 Mar 2007 21:06:08 +0000 x-amz-acl: public-read content-type: application/x-download Content-MD5: 4gJE4saaMU4BqNR0kLY+lw== X-Amz-Meta-ReviewedBy: joe@example.com X-Amz-Meta-ReviewedBy: jane@example.com X-Amz-Meta-FileChecksum: 0x02661779 X-Amz-Meta-ChecksumAlgorithm: crc32 Content-Disposition: attachment; filename=database.dat Content-Encoding: gzip Content-Length: 5913339</pre>	<pre>PUT\n 4gJE4saaMU4BqNR0kLY+lw==\n application/x-download\n Tue, 27 Mar 2007 21:06:08 +0000\n x-amz-acl:public-read\n x-amz-meta-checksumalgorithm:crc32\n x-amz-meta-filechecksum:0x02661779\n x-amz-meta-reviewedby: joe@example.com,jane@example.com\n /static.example.com/db-backup.dat.gz</pre>

Request	StringToSign
<pre style="font-family: monospace; margin: 0;">Authorization: AWS AKIAIOSFODNN7EXAMPLE: dKZcB+bz2EPXgSdXZp9ozGeOM4I=</pre>	

Notice how the 'x-amz-' headers are sorted, trimmed of extra spaces, and converted to lowercase. Note also that multiple headers with the same name have been joined using commas to separate values.

Note how only the Content-Type and Content-MD5 HTTP entity headers appear in the StringToSign. The other Content-* entity headers do not.

Again, note that the CanonicalizedResource includes the bucket name, but the HTTP Request-URI does not. (The bucket is specified by the Host header.)

List all my buckets

Request	StringToSign
<pre style="font-family: monospace; margin: 0;">GET / HTTP/1.1 Host: s3.us-west-1.amazonaws.com Date: Wed, 28 Mar 2007 01:29:59 +0000 Authorization: AWS AKIAIOSFODNN7EXAMPLE:qGdzdERIC03wnaRNKh6OqZehG9s=</pre>	<pre style="font-family: monospace; margin: 0;">GET\n \n \n Wed, 28 Mar 2007 01:29:59 +0000\n /</pre>

Unicode keys

Request	StringToSign
<pre style="font-family: monospace; margin: 0;">GET /dictionary/fran%C3%A7ais/pr%c3%a9re HTTP/1.1 Host: s3.us-west-1.amazonaws.com Date: Wed, 28 Mar 2007 01:49:49 +0000 Authorization: AWS AKIAIOSFODNN7EXAMPLE:DNEZGsoieTZ92F3bUfSPQcbGm%20%C3%a8re</pre>	<pre style="font-family: monospace; margin: 0;">GET\n \n \n Wed, 28 Mar 2007 01:49:49 +0000\n /dictionary/fran%C3%A7ais/pr%c3%a9re</pre>

Note

The elements in StringToSign that were derived from the Request-URI are taken literally, including URL-Encoding and capitalization.

REST request signing problems

When REST request authentication fails, the system responds to the request with an XML error document. The information contained in this error document is meant to help developers diagnose the problem. In particular, the StringToSign element of the SignatureDoesNotMatch error document tells you exactly what request canonicalization the system is using.

Some toolkits silently insert headers that you do not know about beforehand, such as adding the header Content-Type during a PUT. In most of these cases, the value of the inserted header remains constant, allowing you to discover the missing headers by using tools such as Ethereal or tcpmon.

Query string request authentication alternative

You can authenticate certain types of requests by passing the required information as query-string parameters instead of using the `Authorization` HTTP header. This is useful for enabling direct third-party browser access to your private Amazon S3 data without proxying the request. The idea is to construct a "presigned" request and encode it as a URL that an end-user's browser can retrieve. Additionally, you can limit a presigned request by specifying an expiration time.

For more information on using query parameters to authenticate requests, see [Authenticating Requests: Using Query Parameters \(AWS Signature Version 4\)](#) in the *Amazon Simple Storage Service API Reference*. For examples of using the AWS SDKs to generating presigned URLs, see [Sharing objects using presigned URLs \(p. 258\)](#).

Creating a signature

Following is an example query string authenticated Amazon S3 REST request.

```
GET /photos/puppy.jpg
?AWSAccessKeyId=AKIAIOSFODNN7EXAMPLE&Expires=1141889120&Signature=vjbyPxybdZaNmGa
%2ByT272YEAiv4%3D HTTP/1.1
Host: awsexamplebucket1.s3.us-west-1.amazonaws.com
Date: Mon, 26 Mar 2007 19:37:58 +0000
```

The query string request authentication method doesn't require any special HTTP headers. Instead, the required authentication elements are specified as query string parameters:

Query string parameter name	Example value	Description
<code>AWSAccessKeyId</code>	<code>AKIAIOSFODNN7EXAMPLE</code>	Your AWS access key ID. Specifies the AWS secret access key used to sign the request and, indirectly, the identity of the developer making the request.
<code>Expires</code>	<code>1141889120</code>	The time when the signature expires, specified as the number of seconds since the epoch (00:00:00 UTC on January 1, 1970). A request received after this time (according to the server) will be rejected.
<code>Signature</code>	<code>vjbyPxybdZaNmGa</code> <code>%2ByT272YEAiv4%3D</code>	The URL encoding of the Base64 encoding of the HMAC-SHA1 of <code>StringToSign</code> .

The query string request authentication method differs slightly from the ordinary method but only in the format of the `Signature` request parameter and the `StringToSign` element. Following is pseudo-grammar that illustrates the query string request authentication method.

```
Signature = URL-Encode( Base64( HMAC-SHA1( YourSecretAccessKey, UTF-8-Encoding-
Of( StringToSign ) ) ) );
StringToSign = HTTP-VERB + "\n" +
Content-MD5 + "\n" +
Content-Type + "\n" +
```

```
Expires + "\n" +
CanonicalizedAmzHeaders +
CanonicalizedResource;
```

YourSecretAccessKey is the AWS secret access key ID that Amazon assigns to you when you sign up to be an Amazon Web Service developer. Notice how the Signature is URL-Encoded to make it suitable for placement in the query string. Note also that in StringToSign, the HTTP Date positional element has been replaced with Expires. The CanonicalizedAmzHeaders and CanonicalizedResource are the same.

Note

In the query string authentication method, you do not use the Date or the x-amz-date request header when calculating the string to sign.

Query string request authentication

Request	StringToSign
<pre>GET /photos/puppy.jpg? AWSAccessKeyId=AKIAIOSFODNN7EXAMPLE& Signature=NpgCjnDzrM %2BWFWzoENXmpNDUSn8%3D& Expires=1175139620 HTTP/1.1 Host: awsexamplebucket1.s3.us- west-1.amazonaws.com</pre>	<pre>GET\n \n \n 1175139620\n /awsexamplebucket1/photos/puppy.jpg</pre>

We assume that when a browser makes the GET request, it won't provide a Content-MD5 or a Content-Type header, nor will it set any x-amz- headers, so those parts of the StringToSign are left blank.

Using Base64 encoding

HMAC request signatures must be Base64 encoded. Base64 encoding converts the signature into a simple ASCII string that can be attached to the request. Characters that could appear in the signature string like plus (+), forward slash (/), and equals (=) must be encoded if used in a URI. For example, if the authentication code includes a plus (+) sign, encode it as %2B in the request. Encode a forward slash as %2F and equals as %3D.

For examples of Base64 encoding, refer to the Amazon S3 [Authentication examples \(p. 1214\)](#).

Browser-based uploads using POST (AWS signature version 2)

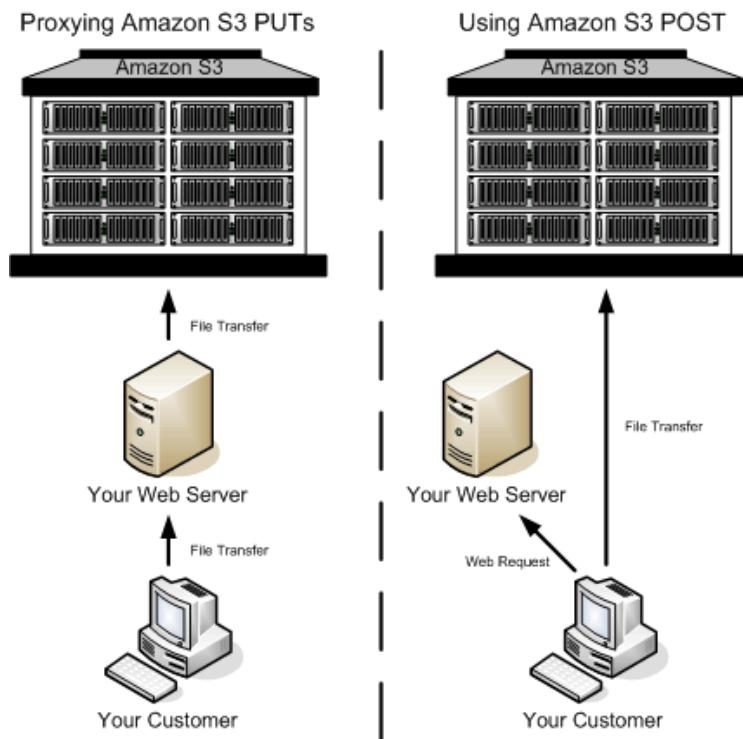
Amazon S3 supports POST, which allows your users to upload content directly to Amazon S3. POST is designed to simplify uploads, reduce upload latency, and save you money on applications where users upload data to store in Amazon S3.

Note

The request authentication discussed in this section is based on AWS Signature Version 2, a protocol for authenticating inbound API requests to AWS services.

Amazon S3 now supports Signature Version 4, a protocol for authenticating inbound API requests to AWS services, in all AWS Regions. At this time, AWS Regions created before January 30, 2014 will continue to support the previous protocol, Signature Version 2. Any new regions after January 30, 2014 will support only Signature Version 4 and therefore all requests to those regions must be made with Signature Version 4. For more information, see [Authenticating Requests in Browser-Based Uploads Using POST \(AWS Signature Version 4\)](#) in the *Amazon Simple Storage Service API Reference*.

The following figure shows an upload using Amazon S3 POST.



Uploading using POST

1	The user opens a web browser and accesses your web page.
2	Your web page contains an HTTP form that contains all the information necessary for the user to upload content to Amazon S3.
3	The user uploads content directly to Amazon S3.

Note

Query string authentication is not supported for POST.

HTML forms (AWS signature version 2)

Topics

- [HTML form encoding \(p. 1221\)](#)
- [HTML form declaration \(p. 1221\)](#)
- [HTML form fields \(p. 1222\)](#)
- [Policy construction \(p. 1224\)](#)
- [Constructing a signature \(p. 1227\)](#)
- [Redirection \(p. 1227\)](#)

When you communicate with Amazon S3, you normally use the REST or SOAP API to perform put, get, delete, and other operations. With POST, users upload data directly to Amazon S3 through their browsers, which cannot process the SOAP API or create a REST PUT request.

Note

SOAP support over HTTP is deprecated, but SOAP is still available over HTTPS. New Amazon S3 features are not supported for SOAP. Instead of using SOAP, we recommend that you use either the REST API or the AWS SDKs.

To allow users to upload content to Amazon S3 by using their browsers, you use HTML forms. HTML forms consist of a form declaration and form fields. The form declaration contains high-level information about the request. The form fields contain detailed information about the request, as well as the policy that is used to authenticate it and ensure that it meets the conditions that you specify.

Note

The form data and boundaries (excluding the contents of the file) cannot exceed 20 KB.

This section explains how to use HTML forms.

HTML form encoding

The form and policy must be UTF-8 encoded. You can apply UTF-8 encoding to the form by specifying it in the HTML heading or as a request header.

Note

The HTML form declaration does not accept query string authentication parameters.

The following is an example of UTF-8 encoding in the HTML heading:

```
<html>
  <head>
    ...
    <meta http-equiv="Content-Type" content="text/html; charset=UTF-8" />
    ...
  </head>
<body>
```

The following is an example of UTF-8 encoding in a request header:

```
Content-Type: text/html; charset=UTF-8
```

HTML form declaration

The form declaration has three components: the action, the method, and the enclosure type. If any of these values is improperly set, the request fails.

The action specifies the URL that processes the request, which must be set to the URL of the bucket. For example, if the name of your bucket is awsexamplebucket1 and the Region is US West (N. California), the URL is <https://awsexamplebucket1.s3.us-west-1.amazonaws.com/>.

Note

The key name is specified in a form field.

The method must be POST.

The enclosure type (enctype) must be specified and must be set to multipart/form-data for both file uploads and text area uploads. For more information, go to [RFC 1867](#).

Example

The following example is a form declaration for the bucket "awsexamplebucket1".

```
<form action="https://awsexamplebucket1.s3.us-west-1.amazonaws.com/" method="post">
```

```
enctype="multipart/form-data">
```

HTML form fields

The following table describes fields that can be used within an HTML form.

Note

The variable \${filename} is automatically replaced with the name of the file provided by the user and is recognized by all form fields. If the browser or client provides a full or partial path to the file, only the text following the last slash (/) or backslash (\) will be used. For example, "C:\Program Files\directory1\file.txt" will be interpreted as "file.txt". If no file or file name is provided, the variable is replaced with an empty string.

Field name	Description	Required
AWSAccessKeyId	The AWS Access Key ID of the owner of the bucket who grants an anonymous user access for a request that satisfies the set of constraints in the policy. This field is required if the request includes a policy document.	Conditional
acl	An Amazon S3 access control list (ACL). If an invalid access control list is specified, an error is generated. For more information on ACLs, see Access control lists (ACLs) (p. 6) . Type: String Default: private Valid Values: private public-read public-read-write aws-exec-read authenticated-read bucket-owner-read bucket-owner-full-control	No
Cache-Control, Content-Type, Content-Disposition, Content-Encoding, Expires	REST-specific headers. For more information, see PUT Object .	No
key	The name of the uploaded key. To use the filename provided by the user, use the \${filename} variable. For example, if user Betty uploads the file lolcatz.jpg and you specify /user/betty/\${filename}, the file is stored as /user/betty/lolcatz.jpg. For more information, see Working with object metadata (p. 153) .	Yes
policy	Security policy describing what is permitted in the request. Requests without a security policy are considered anonymous and will succeed only on publicly writable buckets.	No
success_action_redirect, redirect	The URL to which the client is redirected upon successful upload. Amazon S3 appends the	No

Field name	Description	Required
	<p>bucket, key, and etag values as query string parameters to the URL.</p> <p>If success_action_redirect is not specified, Amazon S3 returns the empty document type specified in the success_action_status field.</p> <p>If Amazon S3 cannot interpret the URL, it ignores the field.</p> <p>If the upload fails, Amazon S3 displays an error and does not redirect the user to a URL.</p> <p>For more information, see Redirection (p. 1227).</p> <p>Note The redirect field name is deprecated and support for the redirect field name will be removed in the future.</p>	
success_action_status	<p>The status code returned to the client upon successful upload if success_action_redirect is not specified.</p> <p>Valid values are 200, 201, or 204 (default).</p> <p>If the value is set to 200 or 204, Amazon S3 returns an empty document with a 200 or 204 status code.</p> <p>If the value is set to 201, Amazon S3 returns an XML document with a 201 status code. For information about the content of the XML document, see POST Object.</p> <p>If the value is not set or if it is set to an invalid value, Amazon S3 returns an empty document with a 204 status code.</p> <p>Note Some versions of the Adobe Flash player do not properly handle HTTP responses with an empty body. To support uploads through Adobe Flash, we recommend setting success_action_status to 201.</p>	No
signature	<p>The HMAC signature constructed by using the secret access key that corresponds to the provided AWSAccessKeyId. This field is required if a policy document is included with the request.</p> <p>For more information, see Identity and access management in Amazon S3 (p. 394).</p>	Conditional

Field name	Description	Required
x-amz-security-token	<p>A security token used by session credentials</p> <p>If the request is using Amazon DevPay then it requires two x-amz-security-token form fields: one for the product token and one for the user token.</p> <p>If the request is using session credentials, then it requires one x-amz-security-token form. For more information, see Temporary Security Credentials in the <i>IAM User Guide</i>.</p>	No
Other field names prefixed with x-amz-meta-	<p>User-specified metadata.</p> <p>Amazon S3 does not validate or use this data.</p> <p>For more information, see PUT Object.</p>	No
file	<p>File or text content.</p> <p>The file or content must be the last field in the form. Any fields below it are ignored.</p> <p>You cannot upload more than one file at a time.</p>	Yes

Policy construction

Topics

- [Expiration \(p. 1225\)](#)
- [Conditions \(p. 1225\)](#)
- [Condition matching \(p. 1226\)](#)
- [Character escaping \(p. 1227\)](#)

The policy is a UTF-8 and Base64-encoded JSON document that specifies conditions that the request must meet and is used to authenticate the content. Depending on how you design your policy documents, you can use them per upload, per user, for all uploads, or according to other designs that meet your needs.

Note

Although the policy document is optional, we highly recommend it over making a bucket publicly writable.

The following is an example of a policy document:

```
{
  "expiration": "2007-12-01T12:00:00.000Z",
  "conditions": [
    {"acl": "public-read" },
    {"bucket": "awsexamplebucket1" },
    ["starts-with", "$key", "user/eric/"],
  ]
}
```

```
    ]  
}
```

The policy document contains the expiration and conditions.

Expiration

The expiration element specifies the expiration date of the policy in ISO 8601 UTC date format. For example, "2007-12-01T12:00:00.000Z" specifies that the policy is not valid after midnight UTC on 2007-12-01. Expiration is required in a policy.

Conditions

The conditions in the policy document validate the contents of the uploaded object. Each form field that you specify in the form (except AWSAccessKeyId, signature, file, policy, and field names that have an x-ignore- prefix) must be included in the list of conditions.

Note

If you have multiple fields with the same name, the values must be separated by commas. For example, if you have two fields named "x-amz-meta-tag" and the first one has a value of "Ninja" and second has a value of "Stallman", you would set the policy document to Ninja, Stallman. All variables within the form are expanded before the policy is validated. Therefore, all condition matching should be performed against the expanded fields. For example, if you set the key field to user/betty/\${filename}, your policy might be ["starts-with", "\$key", "user/betty/"]. Do not enter ["starts-with", "\$key", "user/betty/\${filename}"]. For more information, see [Condition matching \(p. 1226\)](#).

The following table describes policy document conditions.

Element name	Description
acl	Specifies conditions that the ACL must meet. Supports exact matching and starts-with.
content-length-range	Specifies the minimum and maximum allowable size for the uploaded content. Supports range matching.
Cache-Control, Content-Type, Content-Disposition, Content-Encoding, Expires	REST-specific headers. Supports exact matching and starts-with.
key	The name of the uploaded key. Supports exact matching and starts-with.
success_action_redirect, redirect	The URL to which the client is redirected upon successful upload. Supports exact matching and starts-with.
success_action_status	The status code returned to the client upon successful upload if success_action_redirect is not specified. Supports exact matching.
x-amz-security-token	Amazon DevPay security token.

Element name	Description
	Each request that uses Amazon DevPay requires two <code>x-amz-security-token</code> form fields: one for the product token and one for the user token. As a result, the values must be separated by commas. For example, if the user token is <code>eW91dHVIZQ==</code> and the product token is <code>b0hnNVNKVVJIQTA=</code> , you set the policy entry to: { "x-amz-security-token": "eW91dHVIZQ==,b0hnNVNKVVJIQTA=" }.
Other field names prefixed with <code>x-amz-meta-</code>	User-specified metadata. Supports exact matching and <code>starts-with</code> .

Note

If your toolkit adds additional fields (e.g., Flash adds filename), you must add them to the policy document. If you can control this functionality, prefix `x-ignore-` to the field so Amazon S3 ignores the feature and it won't affect future versions of this feature.

Condition matching

The following table describes condition matching types. Although you must specify one condition for each form field that you specify in the form, you can create more complex matching criteria by specifying multiple conditions for a form field.

Condition	Description
Exact Matches	<p>Exact matches verify that fields match specific values. This example indicates that the ACL must be set to public-read:</p> <pre>{"acl": "public-read"}</pre> <p>This example is an alternate way to indicate that the ACL must be set to public-read:</p> <pre>["eq", "\$acl", "public-read"]</pre>
Starts With	<p>If the value must start with a certain value, use <code>starts-with</code>. This example indicates that the key must start with user/betty:</p> <pre>["starts-with", "\$key", "user/betty/"]</pre>
Matching Any Content	<p>To configure the policy to allow any content within a field, use <code>starts-with</code> with an empty value. This example allows any <code>success_action_redirect</code>:</p> <pre>["starts-with", "\$success_action_redirect", ""]</pre>
Specifying Ranges	<p>For fields that accept ranges, separate the upper and lower ranges with a comma. This example allows a file size from 1 to 10 megabytes:</p> <pre>["content-length-range", 1048579, 10485760]</pre>

Character escaping

The following table describes characters that must be escaped within a policy document.

Escape sequence	Description
\\	Backslash
\\$	Dollar sign
\b	Backspace
\f	Form feed
\n	New line
\r	Carriage return
\t	Horizontal tab
\v	Vertical tab
\uXXXX	All Unicode characters

Constructing a signature

Step	Description
1	Encode the policy by using UTF-8.
2	Encode those UTF-8 bytes by using Base64.
3	Sign the policy with your secret access key by using HMAC SHA-1.
4	Encode the SHA-1 signature by using Base64.

For general information about authentication, see [Identity and access management in Amazon S3 \(p. 394\)](#).

Redirection

This section describes how to handle redirects.

General redirection

On completion of the POST request, the user is redirected to the location that you specified in the `success_action_redirect` field. If Amazon S3 cannot interpret the URL, it ignores the `success_action_redirect` field.

If `success_action_redirect` is not specified, Amazon S3 returns the empty document type specified in the `success_action_status` field.

If the POST request fails, Amazon S3 displays an error and does not provide a redirect.

Pre-upload redirection

If your bucket was created using <CreateBucketConfiguration>, your end users might require a redirect. If this occurs, some browsers might handle the redirect incorrectly. This is relatively rare but is most likely to occur right after a bucket is created.

Upload examples (AWS signature version 2)

Topics

- [File upload \(p. 1228\)](#)
- [Text area upload \(p. 1230\)](#)

Note

The request authentication discussed in this section is based on AWS Signature Version 2, a protocol for authenticating inbound API requests to AWS services.

Amazon S3 now supports Signature Version 4, a protocol for authenticating inbound API requests to AWS services, in all AWS Regions. At this time, AWS Regions created before January 30, 2014 will continue to support the previous protocol, Signature Version 2. Any new regions after January 30, 2014 will support only Signature Version 4 and therefore all requests to those regions must be made with Signature Version 4. For more information, see [Examples: Browser-Based Upload using HTTP POST \(Using AWS Signature Version 4\)](#) in the *Amazon Simple Storage Service API Reference*.

File upload

This example shows the complete process for constructing a policy and form that can be used to upload a file attachment.

Policy and form construction

The following policy supports uploads to Amazon S3 for the awsexamplebucket1 bucket.

```
{ "expiration": "2007-12-01T12:00:00.000Z",
  "conditions": [
    {"bucket": "awsexamplebucket1"},
    ["starts-with", "$key", "user/eric/"],
    {"acl": "public-read"},
    {"success_action_redirect": "https://awsexamplebucket1.s3.us-west-1.amazonaws.com/
successful_upload.html"},
    ["starts-with", "$Content-Type", "image/"],
    {"x-amz-meta-uuid": "14365123651274"},
    ["starts-with", "$x-amz-meta-tag", ""]
  ]
}
```

This policy requires the following:

- The upload must occur before 12:00 UTC on December 1, 2007.
- The content must be uploaded to the awsexamplebucket1 bucket.
- The key must start with "user/eric/".
- The ACL is set to public-read.
- The success_action_redirect is set to https://awsexamplebucket1.s3.us-west-1.amazonaws.com/successful_upload.html.
- The object is an image file.
- The x-amz-meta-uuid tag must be set to 14365123651274.

- The x-amz-meta-tag can contain any value.

The following is a Base64-encoded version of this policy.

```
eyAizXhwaXJhdGlvbiI6IClEyMDA3LTEyLTAxVDEyOjAwOjAwLjAwMFoiLAogICJjb25kaXRpb25zIjogWwogICAgeyJidWNrZXQiOiA
```

Using your credentials create a signature, for example 0RavWzkygo6QX9caELEqKi9kDbU= is the signature for the preceding policy document.

The following form supports a POST request to the DOC-EXAMPLE-BUCKET bucket that uses this policy.

```
<html>
  <head>
    ...
    <meta http-equiv="Content-Type" content="text/html; charset=UTF-8" />
    ...
  </head>
  <body>
    ...
    <form action="https://DOC-EXAMPLE-BUCKET.s3.us-west-1.amazonaws.com/" method="post"
      enctype="multipart/form-data">
      Key to upload: <input type="input" name="key" value="user/eric/" /><br />
      <input type="hidden" name="acl" value="public-read" />
      <input type="hidden" name="success_action_redirect" value="https://
      awsexamplebucket1.s3.us-west-1.amazonaws.com/successful_upload.html" />
      Content-Type: <input type="input" name="Content-Type" value="image/jpeg" /><br />
      <input type="hidden" name="x-amz-meta-uuid" value="14365123651274" />
      Tags for File: <input type="input" name="x-amz-meta-tag" value="" /><br />
      <input type="hidden" name="AWSAccessKeyId" value="AKIAIOSFODNN7EXAMPLE" />
      <input type="hidden" name="Policy" value="POLICY" />
      <input type="hidden" name="Signature" value="SIGNATURE" />
      File: <input type="file" name="file" /> <br />
      <!-- The elements after this will be ignored -->
      <input type="submit" name="submit" value="Upload to Amazon S3" />
    </form>
    ...
  </html>
```

Sample request

This request assumes that the image uploaded is 117,108 bytes; the image data is not included.

```
POST / HTTP/1.1
Host: awsexamplebucket1.s3.us-west-1.amazonaws.com
User-Agent: Mozilla/5.0 (Windows; U; Windows NT 5.1; en-US; rv:1.8.1.10) Gecko/20071115
  Firefox/2.0.0.10
Accept: text/xml,application/xml,application/xhtml+xml,text/html;q=0.9,text/
plain;q=0.8,image/png,*/*;q=0.5
Accept-Language: en-us,en;q=0.5
Accept-Encoding: gzip,deflate
Accept-Charset: ISO-8859-1,utf-8;q=0.7,*;q=0.7
Keep-Alive: 300
Connection: keep-alive
Content-Type: multipart/form-data; boundary=9431149156168
Content-Length: 118698

--9431149156168
Content-Disposition: form-data; name="key"

user/eric/MyPicture.jpg
```

```
--9431149156168
Content-Disposition: form-data; name="acl"

public-read
--9431149156168
Content-Disposition: form-data; name="success_action_redirect"

https://awsexamplebucket1.s3.us-west-1.amazonaws.com/successful_upload.html
--9431149156168
Content-Disposition: form-data; name="Content-Type"

image/jpeg
--9431149156168
Content-Disposition: form-data; name="x-amz-meta-uuid"

14365123651274
--9431149156168
Content-Disposition: form-data; name="x-amz-meta-tag"

Some,Tag,For,Picture
--9431149156168
Content-Disposition: form-data; name="AWSAccessKeyId"

AKIAIOSFODNN7EXAMPLE
--9431149156168
Content-Disposition: form-data; name="Policy"

eyAiZXhwaXJhdGlvbiI6IClEyMDA3LTEyVDEyOjAwOjAwLjAwMFOiLAogICJjb25kaXRpb25zIjogWwogICAgeyJidWNrZXQioIA
--9431149156168
Content-Disposition: form-data; name="Signature"

0RavWzkygo6QX9caELEqKi9kDbU=
--9431149156168
Content-Disposition: form-data; name="file"; filename="MyFilename.jpg"
Content-Type: image/jpeg

...file content...
--9431149156168
Content-Disposition: form-data; name="submit"

Upload to Amazon S3
--9431149156168--
```

Sample response

```
HTTP/1.1 303 Redirect
x-amz-request-id: 1AEE782442F35865
x-amz-id-2: cxzFLJRatFHy+NGtaDFRR8YvI9BHmgLxjvJzNiGGICARZ/mVXHj7T+qQKhdpzHFh
Content-Type: application/xml
Date: Wed, 14 Nov 2007 21:21:33 GMT
Connection: close
Location: https://awsexamplebucket1.s3.us-west-1.amazonaws.com/
successful_upload.html?bucket=awsexamplebucket1&key=user/eric/
MyPicture.jpg&etag="39d459dfbc0faabbb5e179358dfb94c3";
Server: AmazonS3
```

Text area upload

Topics

- [Policy and form construction \(p. 1231\)](#)
- [Sample request \(p. 1232\)](#)
- [Sample response \(p. 1233\)](#)

The following example shows the complete process for constructing a policy and form to upload a text area. Uploading a text area is useful for submitting user-created content, such as blog postings.

Policy and form construction

The following policy supports text area uploads to Amazon S3 for the awsexamplebucket1 bucket.

```
{ "expiration": "2007-12-01T12:00:00.000Z",
  "conditions": [
    {"bucket": "awsexamplebucket1"},
    ["starts-with", "$key", "user/eric/"],
    {"acl": "public-read"},
    {"success_action_redirect": "https://awsexamplebucket1.s3.us-west-1.amazonaws.com/
new_post.html"},
    ["eq", "$Content-Type", "text/html"],
    {"x-amz-meta-uuid": "14365123651274"},
    ["starts-with", "$x-amz-meta-tag", ""]
  ]
}
```

This policy requires the following:

- The upload must occur before 12:00 GMT on 2007-12-01.
- The content must be uploaded to the awsexamplebucket1 bucket.
- The key must start with "user/eric/".
- The ACL is set to public-read.
- The success_action_redirect is set to https://awsexamplebucket1.s3.us-west-1.amazonaws.com/new_post.html.
- The object is HTML text.
- The x-amz-meta-uuid tag must be set to 14365123651274.
- The x-amz-meta-tag can contain any value.

Following is a Base64-encoded version of this policy.

```
eyAiZXhwaXJhdGlvbiI6IClEyMDA3LTEyVDEyOjAwOjAwLjAwMFoiLAogICJjb25kaXR
pb25zIjogWwogICAseyJidWNrZXQiOiAiam9obnNtaXR0In0sCiAgICBbInN0YXJ0cy13aXRoIiwgIiRrZXkiLC
AidXNlcj9lcmljLyLAogICAseyJhY2wiOiAicHVibGljLXJlYQifSwKICAgIHsic3VjY2Vzc19hY3Rpb25fc
mVkaXJlY3QiOiaHR0cDovL2pvaG5zbWL
C5zMy5hbWF6b25hd3MuY29tL25ld19wb3N0Lmh0bWwifSwKICAgIFsizXEiLC
AiJENvbnnRlbnQtVHlwZSISICJ0ZXh0L2h0bWwiXSwsK
CAgIHSieC1hbXotbWV0YS11dWlkIjogIjE0MzY1MTIzNjUxMjc0In0sCiAgICBbInN0YXJ0cy13aXRoIiwgIiR4LWFtei1tZXRhLXRh
IsICIiXQogIF0KfQo=
```

Using your credentials, create a signature. For example, qA7FWXKq6VvU68lI9KdveT1cWgF= is the signature for the preceding policy document.

The following form supports a POST request to the DOC-EXAMPLE-BUCKET bucket that uses this policy.

```
<html>
<head>
  ...
  <meta http-equiv="Content-Type" content="text/html; charset=UTF-8" />
  ...
</head>
<body>
  ...
<form action="https://DOC-EXAMPLE-BUCKET.s3.us-west-1.amazonaws.com/" method="post"
enctype="multipart/form-data">
  Key to upload: <input type="input" name="key" value="user/eric/" /><br />
  <input type="hidden" name="acl" value="public-read" />
```

```
<input type="hidden" name="success_action_redirect" value="https://  
awsexamplebucket1.s3.us-west-1.amazonaws.com/new_post.html" />  
<input type="hidden" name="Content-Type" value="text/html" />  
<input type="hidden" name="x-amz-meta-uuid" value="14365123651274" />  
Tags for File: <input type="input" name="x-amz-meta-tag" value="" /><br />  
<input type="hidden" name="AWSAccessKeyId" value="AKIAIOSFODNN7EXAMPLE" />  
<input type="hidden" name="Policy" value="POLICY" />  
<input type="hidden" name="Signature" value="SIGNATURE" />  
Entry: <textarea name="file" cols="60" rows="10">  
  
Your blog post goes here.  
  
</textarea><br />  
<!-- The elements after this will be ignored -->  
<input type="submit" name="submit" value="Upload to Amazon S3" />  
</form>  
...  
</html>
```

Sample request

This request assumes that the image uploaded is 117,108 bytes; the image data is not included.

```
POST / HTTP/1.1  
Host: awsexamplebucket1.s3.us-west-1.amazonaws.com  
User-Agent: Mozilla/5.0 (Windows; U; Windows NT 5.1; en-US; rv:1.8.1.10) Gecko/20071115  
Firefox/2.0.0.10  
Accept: text/xml,application/xml,application/xhtml+xml,text/html;q=0.9,text/  
plain;q=0.8,image/png,*/*;q=0.5  
Accept-Language: en-us,en;q=0.5  
Accept-Encoding: gzip,deflate  
Accept-Charset: ISO-8859-1,utf-8;q=0.7,*;q=0.7  
Keep-Alive: 300  
Connection: keep-alive  
Content-Type: multipart/form-data; boundary=178521717625888  
Content-Length: 118635  
  
-178521717625888  
Content-Disposition: form-data; name="key"  
  
ser/eric/NewEntry.html  
--178521717625888  
Content-Disposition: form-data; name="acl"  
  
public-read  
--178521717625888  

```

```
AKIAIOSFODNN7EXAMPLE
--178521717625888
Content-Disposition: form-data; name="Policy"
eyAiZXhwaXJhdGvbiI6ICiyMDA3LTEyLTAxVDEyOjAwOjAwMFoiLAogICJjb25kaXRpb25zIjogWwogICAgeyJidWNrZXQiOiA
--178521717625888
Content-Disposition: form-data; name="Signature"

qA7FWXKq6VvU68lI9KdveT1cWgF=
--178521717625888
Content-Disposition: form-data; name="file"

...content goes here...
--178521717625888
Content-Disposition: form-data; name="submit"

Upload to Amazon S3
--178521717625888--
```

Sample response

```
HTTP/1.1 303 Redirect
x-amz-request-id: 1AEE782442F35865
x-amz-id-2: cxzFLJRatFHy+NGtaDFRR8YvI9BHmgLxjvJzNiGGICARZ/mVXHj7T+qQKhdpzHFh
Content-Type: application/xml
Date: Wed, 14 Nov 2007 21:21:33 GMT
Connection: close
Location: https://awsexamplebucket1.s3.us-west-1.amazonaws.com/new_post.html?
bucket=awsexamplebucket1&key=user/eric/NewEntry.html&etag=40c3271af26b7f1672e41b8a274d28d4
Server: AmazonS3
```

POST with adobe flash

This section describes how to use POST with Adobe Flash.

Adobe flash player security

By default, the Adobe Flash Player security model prohibits Adobe Flash Players from making network connections to servers outside the domain that serves the SWF file.

To override the default, you must upload a publicly readable crossdomain.xml file to the bucket that will accept POST uploads. The following is a sample crossdomain.xml file.

```
<?xml version="1.0"?>
<!DOCTYPE cross-domain-policy SYSTEM
"http://www.macromedia.com/xml/dtds/cross-domain-policy.dtd">
<cross-domain-policy>
<allow-access-from domain="*" secure="false" />
</cross-domain-policy>
```

Note

For more information about the Adobe Flash security model, go to the Adobe website. Adding the crossdomain.xml file to your bucket allows any Adobe Flash Player to connect to the crossdomain.xml file within your bucket; however, it does not grant access to the actual Amazon S3 bucket.

Adobe flash considerations

The FileReference API in Adobe Flash adds the `Filename` form field to the POST request. When you build Adobe Flash applications that upload to Amazon S3 by using the FileReference API action, include the following condition in your policy:

```
[ 'starts-with', '$Filename', '' ]
```

Some versions of the Adobe Flash Player do not properly handle HTTP responses that have an empty body. To configure POST to return a response that does not have an empty body, set `success_action_status` to 201. Amazon S3 will then return an XML document with a 201 status code. For information about the content of the XML document, see [POST Object](#). For information about form fields, see [HTML form fields \(p. 1222\)](#).

Best practices design patterns: optimizing Amazon S3 performance

Your applications can easily achieve thousands of transactions per second in request performance when uploading and retrieving storage from Amazon S3. Amazon S3 automatically scales to high request rates. For example, your application can achieve at least 3,500 PUT/COPY/POST/DELETE or 5,500 GET/HEAD requests per second per partitioned [prefix](#). There are no limits to the number of prefixes in a bucket. You can increase your read or write performance by using parallelization. For example, if you create 10 prefixes in an Amazon S3 bucket to parallelize reads, you could scale your read performance to 55,000 read requests per second. Similarly, you can scale write operations by writing to multiple prefixes. For more information about creating and using prefixes, see [Organizing objects using prefixes \(p. 243\)](#).

Some data lake applications on Amazon S3 scan millions or billions of objects for queries that run over petabytes of data. These data lake applications achieve single-instance transfer rates that maximize the network interface use for their [Amazon EC2](#) instance, which can be up to 100 Gb/s on a single instance. These applications then aggregate throughput across multiple instances to get multiple terabits per second.

Other applications are sensitive to latency, such as social media messaging applications. These applications can achieve consistent small object latencies (and first-byte-out latencies for larger objects) of roughly 100–200 milliseconds.

Other AWS services can also help accelerate performance for different application architectures. For example, if you want higher transfer rates over a single HTTP connection or single-digit millisecond latencies, use [Amazon CloudFront](#) or [Amazon ElastiCache](#) for caching with Amazon S3.

Additionally, if you want fast data transport over long distances between a client and an S3 bucket, use [Configuring fast, secure file transfers using Amazon S3 Transfer Acceleration \(p. 136\)](#). Transfer Acceleration uses the globally distributed edge locations in CloudFront to accelerate data transport over geographical distances. If your Amazon S3 workload uses server-side encryption with AWS Key Management Service (SSE-KMS), see [AWS KMS Limits](#) in the AWS Key Management Service Developer Guide for information about the request rates supported for your use case.

The following topics describe best practice guidelines and design patterns for optimizing performance for applications that use Amazon S3. Refer to the [Performance Guidelines for Amazon S3 \(p. 1235\)](#) and [Performance Design Patterns for Amazon S3 \(p. 1237\)](#) for the most current information about performance optimization for Amazon S3.

Topics

- [Performance Guidelines for Amazon S3 \(p. 1235\)](#)
- [Performance Design Patterns for Amazon S3 \(p. 1237\)](#)

Performance Guidelines for Amazon S3

When building applications that upload and retrieve objects from Amazon S3, follow our best practices guidelines to optimize performance. We also offer more detailed [Performance Design Patterns \(p. 1237\)](#).

To obtain the best performance for your application on Amazon S3, we recommend the following guidelines.

Topics

- [Measure Performance \(p. 1236\)](#)
- [Scale Storage Connections Horizontally \(p. 1236\)](#)
- [Use Byte-Range Fetches \(p. 1236\)](#)
- [Retry Requests for Latency-Sensitive Applications \(p. 1236\)](#)
- [Combine Amazon S3 \(Storage\) and Amazon EC2 \(Compute\) in the Same AWS Region \(p. 1236\)](#)
- [Use Amazon S3 Transfer Acceleration to Minimize Latency Caused by Distance \(p. 1237\)](#)
- [Use the Latest Version of the AWS SDKs \(p. 1237\)](#)

Measure Performance

When optimizing performance, look at network throughput, CPU, and DRAM requirements. Depending on the mix of demands for these different resources, it might be worth evaluating different [Amazon EC2 instance types](#). For more information about instance types, see [Instance Types](#) in the *Amazon EC2 User Guide for Linux Instances*.

It's also helpful to look at DNS lookup time, latency, and data transfer speed using HTTP analysis tools when measuring performance.

Scale Storage Connections Horizontally

Spreading requests across many connections is a common design pattern to horizontally scale performance. When you build high performance applications, think of Amazon S3 as a very large distributed system, not as a single network endpoint like a traditional storage server. You can achieve the best performance by issuing multiple concurrent requests to Amazon S3. Spread these requests over separate connections to maximize the accessible bandwidth from Amazon S3. Amazon S3 doesn't have any limits for the number of connections made to your bucket.

Use Byte-Range Fetches

Using the Range HTTP header in a [GET Object](#) request, you can fetch a byte-range from an object, transferring only the specified portion. You can use concurrent connections to Amazon S3 to fetch different byte ranges from within the same object. This helps you achieve higher aggregate throughput versus a single whole-object request. Fetching smaller ranges of a large object also allows your application to improve retry times when requests are interrupted. For more information, see [Downloading an object \(p. 209\)](#).

Typical sizes for byte-range requests are 8 MB or 16 MB. If objects are PUT using a multipart upload, it's a good practice to GET them in the same part sizes (or at least aligned to part boundaries) for best performance. GET requests can directly address individual parts; for example, `GET ?partNumber=N`.

Retry Requests for Latency-Sensitive Applications

Aggressive timeouts and retries help drive consistent latency. Given the large scale of Amazon S3, if the first request is slow, a retried request is likely to take a different path and quickly succeed. The AWS SDKs have configurable timeout and retry values that you can tune to the tolerances of your specific application.

Combine Amazon S3 (Storage) and Amazon EC2 (Compute) in the Same AWS Region

Although S3 bucket names are [globally unique](#), each bucket is stored in a Region that you select when you create the bucket. To optimize performance, we recommend that you access the bucket from

Amazon EC2 instances in the same AWS Region when possible. This helps reduce network latency and data transfer costs.

For more information about data transfer costs, see [Amazon S3 Pricing](#).

Use Amazon S3 Transfer Acceleration to Minimize Latency Caused by Distance

[Configuring fast, secure file transfers using Amazon S3 Transfer Acceleration \(p. 136\)](#) manages fast, easy, and secure transfers of files over long geographic distances between the client and an S3 bucket. Transfer Acceleration takes advantage of the globally distributed edge locations in [Amazon CloudFront](#). As the data arrives at an edge location, it is routed to Amazon S3 over an optimized network path. Transfer Acceleration is ideal for transferring gigabytes to terabytes of data regularly across continents. It's also useful for clients that upload to a centralized bucket from all over the world.

You can use the [Amazon S3 Transfer Acceleration Speed Comparison tool](#) to compare accelerated and non-accelerated upload speeds across Amazon S3 Regions. The Speed Comparison tool uses multipart uploads to transfer a file from your browser to various Amazon S3 Regions with and without using Amazon S3 Transfer Acceleration.

Use the Latest Version of the AWS SDKs

The AWS SDKs provide built-in support for many of the recommended guidelines for optimizing Amazon S3 performance. The SDKs provide a simpler API for taking advantage of Amazon S3 from within an application and are regularly updated to follow the latest best practices. For example, the SDKs include logic to automatically retry requests on HTTP 503 errors and are investing in code to respond and adapt to slow connections.

The SDKs also provide the [Transfer Manager](#), which automates horizontally scaling connections to achieve thousands of requests per second, using byte-range requests where appropriate. It's important to use the latest version of the AWS SDKs to obtain the latest performance optimization features.

You can also optimize performance when you are using HTTP REST API requests. When using the REST API, you should follow the same best practices that are part of the SDKs. Allow for timeouts and retries on slow requests, and multiple connections to allow fetching of object data in parallel. For information about using the REST API, see the [Amazon Simple Storage Service API Reference](#).

Performance Design Patterns for Amazon S3

When designing applications to upload and retrieve objects from Amazon S3, use our best practices design patterns for achieving the best performance for your application. We also offer [Performance Guidelines \(p. 1235\)](#) for you to consider when planning your application architecture.

To optimize performance, you can use the following design patterns.

Topics

- [Using Caching for Frequently Accessed Content \(p. 1238\)](#)
- [Timeouts and Retries for Latency-Sensitive Applications \(p. 1238\)](#)
- [Horizontal Scaling and Request Parallelization for High Throughput \(p. 1239\)](#)
- [Using Amazon S3 Transfer Acceleration to Accelerate Geographically Disparate Data Transfers \(p. 1240\)](#)

Using Caching for Frequently Accessed Content

Many applications that store data in Amazon S3 serve a “working set” of data that is repeatedly requested by users. If a workload is sending repeated GET requests for a common set of objects, you can use a cache such as [Amazon CloudFront](#), [Amazon ElastiCache](#), or [AWS Elemental MediaStore](#) to optimize performance. Successful cache adoption can result in low latency and high data transfer rates. Applications that use caching also send fewer direct requests to Amazon S3, which can help reduce request costs.

Amazon CloudFront is a fast content delivery network (CDN) that transparently caches data from Amazon S3 in a large set of geographically distributed points of presence (PoPs). When objects might be accessed from multiple Regions, or over the internet, CloudFront allows data to be cached close to the users that are accessing the objects. This can result in high performance delivery of popular Amazon S3 content. For information about CloudFront, see the [Amazon CloudFront Developer Guide](#).

Amazon ElastiCache is a managed, in-memory cache. With ElastiCache, you can provision Amazon EC2 instances that cache objects in memory. This caching results in orders of magnitude reduction in GET latency and substantial increases in download throughput. To use ElastiCache, you modify application logic to both populate the cache with hot objects and check the cache for hot objects before requesting them from Amazon S3. For examples of using ElastiCache to improve Amazon S3 GET performance, see the blog post [Turbocharge Amazon S3 with Amazon ElastiCache for Redis](#).

AWS Elemental MediaStore is a caching and content distribution system specifically built for video workflows and media delivery from Amazon S3. MediaStore provides end-to-end storage APIs specifically for video, and is recommended for performance-sensitive video workloads. For information about MediaStore, see the [AWS Elemental MediaStore User Guide](#).

Timeouts and Retries for Latency-Sensitive Applications

There are certain situations where an application receives a response from Amazon S3 indicating that a retry is necessary. Amazon S3 maps bucket and object names to the object data associated with them. If an application generates high request rates (typically sustained rates of over 5,000 requests per second to a small number of objects), it might receive HTTP 503 *slowdown* responses. If these errors occur, each AWS SDK implements automatic retry logic using exponential backoff. If you are not using an AWS SDK, you should implement retry logic when receiving the HTTP 503 error. For information about back-off techniques, see [Error Retries and Exponential Backoff in AWS](#) in the *Amazon Web Services General Reference*.

Amazon S3 automatically scales in response to sustained new request rates, dynamically optimizing performance. While Amazon S3 is internally optimizing for a new request rate, you will receive HTTP 503 request responses temporarily until the optimization completes. After Amazon S3 internally optimizes performance for the new request rate, all requests are generally served without retries.

For latency-sensitive applications, Amazon S3 advises tracking and aggressively retrying slower operations. When you retry a request, we recommend using a new connection to Amazon S3 and performing a fresh DNS lookup.

When you make large variably sized requests (for example, more than 128 MB), we advise tracking the throughput being achieved and retrying the slowest 5 percent of the requests. When you make smaller requests (for example, less than 512 KB), where median latencies are often in the tens of milliseconds range, a good guideline is to retry a GET or PUT operation after 2 seconds. If additional retries are needed, the best practice is to back off. For example, we recommend issuing one retry after 2 seconds and a second retry after an additional 4 seconds.

If your application makes fixed-size requests to Amazon S3, you should expect more consistent response times for each of these requests. In this case, a simple strategy is to identify the slowest 1 percent of requests and to retry them. Even a single retry is frequently effective at reducing latency.

If you are using AWS Key Management Service (AWS KMS) for server-side encryption, see [Limits](#) in the *AWS Key Management Service Developer Guide* for information about the request rates that are supported for your use case.

Horizontal Scaling and Request Parallelization for High Throughput

Amazon S3 is a very large distributed system. To help you take advantage of its scale, we encourage you to horizontally scale parallel requests to the Amazon S3 service endpoints. In addition to distributing the requests within Amazon S3, this type of scaling approach helps distribute the load over multiple paths through the network.

For high-throughput transfers, Amazon S3 advises using applications that use multiple connections to GET or PUT data in parallel. For example, this is supported by [Amazon S3 Transfer Manager](#) in the AWS Java SDK, and most of the other AWS SDKs provide similar constructs. For some applications, you can achieve parallel connections by launching multiple requests concurrently in different application threads, or in different application instances. The best approach to take depends on your application and the structure of the objects that you are accessing.

You can use the AWS SDKs to issue GET and PUT requests directly rather than employing the management of transfers in the AWS SDK. This approach lets you tune your workload more directly, while still benefiting from the SDK's support for retries and its handling of any HTTP 503 responses that might occur. As a general rule, when you download large objects within a Region from Amazon S3 to [Amazon EC2](#), we suggest making concurrent requests for byte ranges of an object at the granularity of 8–16 MB. Make one concurrent request for each 85–90 MB/s of desired network throughput. To saturate a 10 Gb/s network interface card (NIC), you might use about 15 concurrent requests over separate connections. You can scale up the concurrent requests over more connections to saturate faster NICs, such as 25 Gb/s or 100 Gb/s NICs.

Measuring performance is important when you tune the number of requests to issue concurrently. We recommend starting with a single request at a time. Measure the network bandwidth being achieved and the use of other resources that your application uses in processing the data. You can then identify the bottleneck resource (that is, the resource with the highest usage), and hence the number of requests that are likely to be useful. For example, if processing one request at a time leads to a CPU usage of 25 percent, it suggests that up to four concurrent requests can be accommodated. Measurement is essential, and it is worth confirming resource use as the request rate is increased.

If your application issues requests directly to Amazon S3 using the REST API, we recommend using a pool of HTTP connections and re-using each connection for a series of requests. Avoiding per-request connection setup removes the need to perform TCP slow-start and Secure Sockets Layer (SSL) handshakes on each request. For information about using the REST API, see the [Amazon Simple Storage Service API Reference](#).

Finally, it's worth paying attention to DNS and double-checking that requests are being spread over a wide pool of Amazon S3 IP addresses. DNS queries for Amazon S3 cycle through a large list of IP endpoints. But caching resolvers or application code that reuses a single IP address do not benefit from address diversity and the load balancing that follows from it. Network utility tools such as the `netstat` command line tool can show the IP addresses being used for communication with Amazon S3, and we provide guidelines for DNS configurations to use. For more information about these guidelines, see [Making requests \(p. 1138\)](#).

Using Amazon S3 Transfer Acceleration to Accelerate Geographically Disparate Data Transfers

[Configuring fast, secure file transfers using Amazon S3 Transfer Acceleration \(p. 136\)](#) is effective at minimizing or eliminating the latency caused by geographic distance between globally dispersed clients and a regional application using Amazon S3. Transfer Acceleration uses the globally distributed edge locations in CloudFront for data transport. The AWS edge network has points of presence in more than 50 locations. Today, it is used to distribute content through CloudFront and to provide rapid responses to DNS queries made to [Amazon Route 53](#).

The edge network also helps to accelerate data transfers into and out of Amazon S3. It is ideal for applications that transfer data across or between continents, have a fast internet connection, use large objects, or have a lot of content to upload. As the data arrives at an edge location, data is routed to Amazon S3 over an optimized network path. In general, the farther away you are from an Amazon S3 Region, the higher the speed improvement you can expect from using Transfer Acceleration.

You can set up Transfer Acceleration on new or existing buckets. You can use a separate Amazon S3 Transfer Acceleration endpoint to use the AWS edge locations. The best way to test whether Transfer Acceleration helps client request performance is to use the [Amazon S3 Transfer Acceleration Speed Comparison tool](#). Network configurations and conditions vary from time to time and from location to location. So you are charged only for transfers where Amazon S3 Transfer Acceleration can potentially improve your upload performance. For information about using Transfer Acceleration with different AWS SDKs, see [Enabling and using S3 Transfer Acceleration \(p. 139\)](#).

What is Amazon S3 on Outposts?

AWS Outposts is a fully managed service that offers the same AWS infrastructure, AWS services, APIs, and tools to virtually any data center, co-location space, or on-premises facility for a truly consistent hybrid experience. AWS Outposts is ideal for workloads that require low-latency access to on-premises systems, local data processing, data residency, and migration of applications with local system interdependencies. For more information, see [What is AWS Outposts?](#) in the *AWS Outposts User Guide*.

With Amazon S3 on Outposts, you can create S3 buckets on your AWS Outposts and easily store and retrieve objects on premises. S3 on Outposts provides a new storage class, OUTPOSTS, which uses the Amazon S3 APIs and is designed to store data durably and redundantly across multiple devices and servers on your AWS Outposts. You communicate with your Outposts bucket using an access point and endpoint connection over a virtual private cloud (VPC). You can use the same APIs and features on Outposts buckets as you do on Amazon S3, including access policies, encryption, and tagging. You can use S3 on Outposts through the AWS Management Console, AWS Command Line Interface (AWS CLI), AWS SDKs, or REST API.

- [How S3 on Outposts works \(p. 1241\)](#)
- [Features of S3 on Outposts \(p. 1243\)](#)
- [Related services \(p. 1244\)](#)
- [Accessing S3 on Outposts \(p. 1245\)](#)
- [Paying for S3 on Outposts \(p. 1245\)](#)
- [Next steps \(p. 1245\)](#)

How S3 on Outposts works

S3 on Outposts is an object storage service that stores data as objects within buckets on your Outpost. An *object* is a data file and any metadata that describes the file. A *bucket* is a container for objects.

To store your data in S3 on Outposts, you first create a bucket. When you create the bucket, you specify a bucket name and the Outpost that will hold the bucket. To access your S3 on Outposts bucket and perform object operations, you next create and configure an access point. You must also create an endpoint to route requests to your access point.

Access points simplify data access for any AWS service or customer application that stores data in S3. Access points are named network endpoints that are attached to buckets and can be used to perform object operations, such as `GetObject` and `PutObject`. Each access point has distinct permissions and network controls.

You can create and manage your S3 on Outposts buckets, access points, and endpoints by using the AWS Management Console, AWS CLI, AWS SDKs, or REST API. To upload and manage objects in your S3 on Outposts bucket, you can use the AWS CLI, AWS SDKs, or REST API.

Regions

During AWS Outposts provisioning, you or AWS creates a service link connection that connects your Outpost back to your chosen AWS Region or Outposts home Region for bucket operations and telemetry. An Outpost relies on connectivity to the parent AWS Region. The Outposts rack is not designed for

disconnected operations or environments with limited to no connectivity. For more information, see [Outpost connectivity to AWS Regions](#) in the *AWS Outposts User Guide*.

Buckets

A bucket is a container for objects stored in S3 on Outposts. You can store any number of objects in a bucket and can have up to 100 buckets per account per Outpost.

When you create a bucket, you enter a bucket name and choose the Outpost where the bucket will reside. After you create a bucket, you cannot change the bucket name or move the bucket to a different Outpost. Bucket names must follow [Amazon S3 bucket naming rules](#). In S3 on Outposts, bucket names are unique to an Outpost and AWS account. S3 on Outposts buckets require the `outpost-id`, `account-id`, and bucket name to identify them. The following example shows the Amazon Resource Name (ARN) format for S3 on Outposts buckets. The ARN is comprised of the Region your Outpost is homed to, your Outpost account, the Outpost ID, and the bucket name.

```
arn:aws:s3-outposts:region:account-id:outpost/outpost-id/bucket/bucket-name
```

Every object is contained in a bucket. You must use access points to access any object in an Outposts bucket. When you specify the bucket for object operations, you use the access point ARN, which includes the `outpost-id`, `account-id`, and access point name. The following example shows the ARN format for S3 on Outposts access points in object operations:

```
arn:aws:s3-outposts:region:account-id:outpost/outpost-id/accesspoint/accesspoint-name
```

For more information about buckets, see [Working with S3 on Outposts buckets \(p. 1261\)](#).

Objects

Objects are the fundamental entities stored in S3 on Outposts. Objects consist of object data and metadata. The metadata is a set of name-value pairs that describe the object. These pairs include some default metadata, such as the date last modified, and standard HTTP metadata, such as `Content-Type`. You can also specify custom metadata at the time that the object is stored. An object is uniquely identified within a bucket by a [key \(or name\) \(p. 5\)](#).

With Amazon S3 on Outposts, object data is always stored on the Outpost. When AWS installs an Outpost rack, your data stays local to your Outpost to meet data-residency requirements. Your objects never leave your Outpost and are not in an AWS Region. Because the AWS Management Console is hosted in-Region, you can't use the console to upload or manage objects in your Outpost. However, you can use the REST API, AWS Command Line Interface (AWS CLI), and AWS SDKs to upload and manage your objects through your access points.

Keys

An *object key (or key name)* is the unique identifier for an object within a bucket. Every object in a bucket has exactly one key. The combination of a bucket and object key uniquely identifies each object.

The following example shows the ARN format for S3 on Outposts objects, which includes the AWS Region code for the Region that the Outpost is homed to, AWS account ID, Outpost ID, bucket name, and object key:

```
arn:aws:s3-outposts:us-west-2:123456789012:outpost/op-01ac5d28a6a232904/bucket/DOC-EXAMPLE-BUCKET1/object/myobject
```

For more information about object keys, see [Working with S3 on Outposts objects \(p. 1286\)](#).

Storage class and encryption

S3 on Outposts provides a new storage class, S3 Outposts (OUTPOSTS). The S3 Outposts storage class is available only for objects stored in buckets on AWS Outposts. If you try to use other S3 storage classes with S3 on Outposts, S3 on Outposts returns the `InvalidStorageClass` error.

By default, objects stored in the S3 Outposts (OUTPOSTS) storage class are encrypted using server-side encryption with Amazon S3 managed encryption keys (SSE-S3). For more information, see [Data encryption in S3 on Outposts \(p. 1310\)](#).

Bucket policy

A bucket policy is a resource-based AWS Identity and Access Management (IAM) policy that you can use to grant access permissions to your bucket and the objects in it. Only the bucket owner can associate a policy with a bucket. The permissions attached to the bucket apply to all of the objects in the bucket that are owned by the bucket owner. Bucket policies are limited to 20 KB in size.

Bucket policies use JSON-based IAM policy language that is standard across AWS. You can use bucket policies to add or deny permissions for the objects in a bucket. Bucket policies allow or deny requests based on the elements in the policy. These elements can include the requester, S3 on Outposts actions, resources, and aspects or conditions of the request (for example, the IP address used to make the request). For example, you can create a bucket policy that grants cross-account permissions to upload objects to an S3 on Outposts bucket while ensuring that the bucket owner has full control of the uploaded objects. For more information, see [Bucket policy examples \(p. 490\)](#).

In your bucket policy, you can use wildcard characters (*) in ARNs and other values to grant permissions to a subset of objects. For example, you can control access to groups of objects that begin with a common [prefix](#) or end with a given extension, such as .html.

S3 on Outposts access points

S3 on Outposts access points are named network endpoints with dedicated access policies that describe how data can be accessed using that endpoint. Access points simplify managing data access at scale for shared datasets in S3 on Outposts. Access points are attached to buckets that you can use to perform S3 object operations, such as `GetObject` and `PutObject`.

Access points have distinct permissions and network controls that S3 on Outposts applies for any request that is made through that access point. Each access point enforces a customized access point policy that works in conjunction with the bucket policy that is attached to the underlying bucket.

For more information, see [Accessing your S3 on Outposts buckets and objects \(p. 1260\)](#).

Features of S3 on Outposts

Access management

S3 on Outposts provides features for auditing and managing access to your buckets and objects. By default, S3 on Outposts buckets and the objects in them are private. You have access only to the S3 on Outposts resources that you create.

To grant granular resource permissions that support your specific use case or to audit the permissions of your S3 on Outposts resources, you can use the following features.

- [S3 Block Public Access](#) – Block public access to buckets and objects. For buckets on Outposts, Block Public Access is always enabled by default.

- [AWS Identity and Access Management \(IAM\)](#) – Create IAM users for your AWS account to manage access to your S3 on Outposts resources. For example, you can use IAM with S3 on Outposts to control the type of access a user or group of users has to a bucket.
- [S3 on Outposts access points](#) – Manage data access for shared datasets in S3 on Outposts. Access points are named network endpoints with dedicated access policies. Access points are attached to buckets and can be used to perform object operations, such as `GetObject` and `PutObject`.
- [Bucket policies](#) – Use IAM-based policy language to configure resource-based permissions for your S3 buckets and the objects in them.
- [AWS Resource Access Manager \(AWS RAM\)](#) – Securely share your S3 on Outposts capacity across AWS accounts, within your organization or organizational units (OUs) in AWS Organizations.

Storage logging and monitoring

S3 on Outposts provides logging and monitoring tools that you can use to monitor and control how your S3 on Outposts resources are being used. For more information, see [Monitoring tools](#).

- [Amazon CloudWatch metrics for S3 on Outposts](#) – Track the operational health of your resources and understand your capacity availability.
- [Amazon CloudWatch Events events for S3 on Outposts](#) – Create a rule for any S3 on Outposts API event to receive notifications through all supported CloudWatch Events targets, including Amazon Simple Queue Service (Amazon SQS), Amazon Simple Notification Service (Amazon SNS), and AWS Lambda.
- [AWS CloudTrail logs for S3 on Outposts](#) – Record actions taken by a user, a role, or an AWS service in S3 on Outposts. CloudTrail logs provide you with detailed API tracking for S3 bucket-level and object-level operations.

Strong consistency

S3 on Outposts provides strong read-after-write consistency for PUT and DELETE requests of objects in your S3 on Outposts bucket in all AWS Regions. This behavior applies to both writes of new objects and to PUT requests that overwrite existing objects and to DELETE requests. In addition, S3 on Outposts object tags and object metadata (for example, the HEAD object) are strongly consistent. For more information, see [Amazon S3 data consistency model \(p. 6\)](#).

Related services

After you load your data into S3 on Outposts, you can use it with other AWS services. The following are the services that you might use most frequently:

- [Amazon Elastic Compute Cloud \(Amazon EC2\)](#) – Provides secure and scalable computing capacity in the AWS Cloud. Using Amazon EC2 lessens your need to invest in hardware up front, so you can develop and deploy applications faster. You can use Amazon EC2 to launch as many or as few virtual servers as you need, configure security and networking, and manage storage.
- [Amazon Elastic Block Store \(Amazon EBS\) on Outposts](#) – Use Amazon EBS local snapshots on Outposts to store snapshots of volumes on an Outpost locally in S3 on Outposts.
- [Amazon Relational Database Service \(Amazon RDS\) on Outposts](#) – Use Amazon RDS local backups to store your Amazon RDS backups locally on your Outpost.
- [AWS DataSync](#) – Automate transferring data between your Outposts and AWS Regions, choosing what to transfer, when to transfer, and how much network bandwidth to use. S3 on Outposts is integrated with AWS DataSync. For on-premises applications that require high-throughput local processing, S3

on Outposts provides on-premises object storage to minimize data transfers and buffer from network variations, while providing you the ability to easily transfer data between Outposts and AWS Regions.

Accessing S3 on Outposts

You can work with S3 on Outposts in any of the following ways:

AWS Management Console

The console is a web-based user interface for managing S3 on Outposts and AWS resources. If you've signed up for an AWS account, you can access S3 on Outposts by signing into the AWS Management Console and choosing **S3** from the AWS Management Console home page. Then, choose **Outposts buckets** from the left navigation pane.

AWS Command Line Interface

You can use the AWS command line tools to issue commands or build scripts at your system's command line to perform AWS (including S3) tasks.

The [AWS Command Line Interface \(AWS CLI\)](#) provides commands for a broad set of AWS services. The AWS CLI is supported on Windows, macOS, and Linux. To get started, see the [AWS Command Line Interface User Guide](#). For more information about the commands that you can use with S3 on Outposts, see `s3api`, `s3control`, and `s3outposts` in the [AWS CLI Command Reference](#).

AWS SDKs

AWS provides SDKs (software development kits) that consist of libraries and sample code for various programming languages and platforms (Java, Python, Ruby, .NET, iOS, Android, and so on). The AWS SDKs provide a convenient way to create programmatic access to S3 on Outposts and AWS. Because S3 on Outposts uses the same SDKs as Amazon S3, S3 on Outposts provides a consistent experience using the same S3 APIs, automation, and tools.

S3 on Outposts is a REST service. You can send requests to S3 on Outposts by using the AWS SDK libraries, which wrap the underlying REST API and simplify your programming tasks. For example, the SDKs take care of tasks such as calculating signatures, cryptographically signing requests, managing errors, and retrying requests automatically. For information about the AWS SDKs, including how to download and install them, see [Tools to Build on AWS](#).

Paying for S3 on Outposts

You can purchase a variety of AWS Outposts rack configurations featuring a combination of Amazon EC2 instance types, Amazon EBS General Purpose solid state drive (SSD) volumes (gp2), and S3 on Outposts. Pricing includes delivery, installation, infrastructure service maintenance, and software patches and upgrades.

For more information, see [AWS Outposts rack pricing](#).

Next steps

For more information about working with S3 on Outposts, see the following topics:

- [Setting up your Outpost \(p. 1246\)](#)
- [How is Amazon S3 on Outposts different from Amazon S3? \(p. 1247\)](#)
- [Getting started with Amazon S3 on Outposts \(p. 1249\)](#)
- [Networking for S3 on Outposts \(p. 1260\)](#)
- [Working with S3 on Outposts buckets \(p. 1261\)](#)
- [Working with S3 on Outposts objects \(p. 1286\)](#)
- [Security in S3 on Outposts \(p. 1310\)](#)
- [Managing S3 on Outposts storage \(p. 1317\)](#)
- [Developing with Amazon S3 on Outposts \(p. 1323\)](#)

Setting up your Outpost

To get started with Amazon S3 on Outposts, you will need an Outpost with Amazon S3 capacity deployed at your facility. For information about options for ordering an Outpost and S3 capacity, see [AWS Outposts](#). For specifications and to see how S3 on Outposts is different than Amazon S3, see [How is Amazon S3 on Outposts different from Amazon S3? \(p. 1247\)](#)

For more information, see the following topics.

Topics

- [Order a new Outpost \(p. 1246\)](#)
- [Add Amazon S3 storage to an existing Outpost \(p. 1246\)](#)

Order a new Outpost

If you need to order a new Outpost with S3 capacity, see [AWS Outposts rack pricing](#) to understand the capacity options for Amazon Elastic Compute Cloud (Amazon EC2), Amazon Elastic Block Store (Amazon EBS), and Amazon S3.

After you select your configuration, follow the steps in [Create an Outpost and order Outpost capacity](#) in the *AWS Outposts User Guide*.

Add Amazon S3 storage to an existing Outpost

If AWS Outposts is already deployed at your site, depending on your current Outpost configuration and storage capacity, you might be able to add Amazon S3 storage to an existing Outpost, or you might need to work with your AWS account team to add additional hardware to support Amazon S3 on Outposts.

To add Amazon S3 capacity to an existing Outpost

1. From the AWS Outposts owner account, sign in to the AWS Management Console and open the AWS Outposts console at <https://console.aws.amazon.com/outposts/>.
2. Choose the Outpost ID that you would like to add capacity to.
3. Choose **Actions**, and then choose **Increase capacity**.
4. Choose **Request increase**.

Note

An AWS representative will contact you within three business days. The increase in capacity might require AWS to install additional equipment at your site.

How is Amazon S3 on Outposts different from Amazon S3?

Amazon S3 on Outposts delivers object storage to your on-premises AWS Outposts environment and helps you to meet local processing, data residency, and demanding performance needs by keeping data close to on-premises applications. Using Amazon S3 APIs and features, S3 on Outposts makes it easy to store, secure, tag, report on, and control access to the data on your Outposts and extend AWS infrastructure to your in-premises facility for a consistent hybrid experience.

For more information about how S3 on Outposts is unique, see the following topics.

Topics

- [S3 on Outposts specifications \(p. 1247\)](#)
- [API operations supported by S3 on Outposts \(p. 1247\)](#)
- [Amazon S3 features not supported by S3 on Outposts \(p. 1247\)](#)
- [S3 on Outposts network requirements \(p. 1248\)](#)

S3 on Outposts specifications

- The maximum Outposts bucket size is 50 TB.
- The maximum number of Outposts buckets is 100 per AWS account.
- Outposts buckets can be accessed only by using access points and endpoints.
- The maximum number of access points per Outposts bucket is 10.
- Access point policies are limited to 20 KB in size.
- The Outpost owner can manage access within your organization in AWS Organizations by using AWS Resource Access Manager. All accounts that need access to the Outpost must be within the same organization as the owner account in AWS Organizations.
- The S3 on Outposts bucket owner account is always the owner of all objects in the bucket.
- Only the S3 on Outposts bucket owner account can perform operations on the bucket.
- Object size limitations are consistent with Amazon S3.
- All objects stored on S3 on Outposts are stored in the OUTPOSTS storage class.
- By default, all objects stored in the OUTPOSTS storage class are stored by using server-side encryption with Amazon S3 managed encryption keys (SSE-S3). You can also explicitly choose to store objects by using server-side encryption with customer-provided encryption keys (SSE-C).
- If there is not enough space to store an object on your Outpost, the API returns an insufficient capacity exception (ICE).

API operations supported by S3 on Outposts

For a list of API operations supported by S3 on Outposts, see [Amazon S3 on Outposts API operations \(p. 1323\)](#).

Amazon S3 features not supported by S3 on Outposts

The following Amazon S3 features are currently not supported by Amazon S3 on Outposts. Any attempts to use them are rejected.

- Access control lists (ACLs)
- Access point alias names
- Cross-origin resource sharing (CORS)
- S3 Batch Operations
- S3 Inventory reports
- Changing the default bucket encryption
- Public buckets
- Multi-factor authentication (MFA) delete
- S3 Lifecycle transitions (aside from object deletion and stopping incomplete multipart uploads)
- S3 Object Lock legal hold
- Object Lock retention
- S3 Versioning
- Server-side encryption with AWS Key Management Service (AWS KMS) keys (SSE-KMS)
- S3 Replication
- S3 Replication Time Control (S3 RTC)
- Amazon CloudWatch request metrics
- Metrics configuration
- Transfer Acceleration
- S3 Event Notifications
- Requester Pays buckets
- S3 Select
- AWS Lambda events
- Server access logging
- HTTP POST requests
- SOAP
- Website access

S3 on Outposts network requirements

- To route requests to an S3 on Outposts access point, you must create and configure an S3 on Outposts endpoint. The following limits apply to endpoints for S3 on Outposts:
 - Each virtual private cloud (VPC) on an Outpost can have one associated endpoint, and you can have up to 100 endpoints per Outpost.
 - You can map multiple access points to the same endpoint.
 - You can add endpoints only to VPCs with CIDR blocks in the subspaces of the following CIDR ranges:
 - 10.0.0.0/8
 - 172.16.0.0/12
 - 192.168.0.0/16
- You can create endpoints to an Outpost only from VPCs that have non-overlapping CIDR blocks.
- You can create an endpoint only from within its Outposts subnet.
- The subnet that you use to create an endpoint must contain four IP addresses for S3 on Outposts to use.
- If you specify the customer-owned IP address pool (CoIP pool), it must contain four IP addresses for S3 on Outposts to use.

- You can create only one endpoint per Outpost per VPC.

Getting started with Amazon S3 on Outposts

With Amazon S3 on Outposts, you can create S3 buckets on your AWS Outposts and easily store and retrieve objects on premises for applications that require local data access, local data processing, and data residency. S3 on Outposts provides a new storage class, S3 Outposts (OUTPOSTS), which uses the Amazon S3 APIs, and is designed to store data durably and redundantly across multiple devices and servers on your AWS Outposts. You communicate with your Outpost bucket by using an access point and endpoint connection over a virtual private cloud (VPC). You can use the same APIs and features on Outpost buckets as you do on Amazon S3 buckets, including access policies, encryption, and tagging. You can use S3 on Outposts through the AWS Management Console, AWS Command Line Interface (AWS CLI), AWS SDKs, or REST API.

With Amazon S3 on Outposts, you can use the Amazon S3 APIs and features, such as object storage, access policies, encryption, and tagging, on AWS Outposts as you do on Amazon S3. For information about S3 on Outposts, see [What is Amazon S3 on Outposts? \(p. 1241\)](#)

Topics

- [Setting up IAM with S3 on Outposts \(p. 1249\)](#)
- [Getting started by using the AWS Management Console \(p. 1254\)](#)
- [Getting started by using the AWS CLI and SDK for Java \(p. 1255\)](#)

Setting up IAM with S3 on Outposts

AWS Identity and Access Management (IAM) is an AWS service that helps an administrator securely control access to AWS resources. IAM administrators control who can be authenticated (signed in) and authorized (have permissions) to use AWS resources. IAM enables you to create users and groups under your AWS account. You control the permissions that users have to perform tasks with AWS resources. You can use IAM for no additional charge.

By default, IAM users don't have permissions for Amazon S3 on Outposts resources and operations. To allow IAM users to manage S3 on Outposts resources, you must do the following:

- Create an IAM policy that explicitly grants IAM users or groups permissions.
- Attach the policy to the IAM users or groups that require those permissions.

In addition to IAM policies, S3 on Outposts supports both bucket and access point policies. Bucket policies and access point policies are [resource-based policies](#) that are attached to the S3 on Outposts resource.

- A bucket policy is attached to the bucket and allows or denies requests to the bucket and the objects in it based on the elements in the policy.
- In contrast, an access point policy is attached to the access point and allows or denies requests to the access point.

The access point policy works with the bucket policy that is attached to the underlying S3 on Outposts bucket. For an application or user to access objects in an S3 on Outposts bucket through an S3 on Outposts access point, both the access point policy and the bucket policy must permit the request.

Restrictions that you include in an access point policy apply only to requests made through that access point. For example, if an access point is attached to a bucket, you can't use the access point policy to

allow or deny requests that are made directly to the bucket. However, restrictions that you apply to a bucket policy can allow or deny requests made directly to the bucket or through the access point.

In an IAM policy or a resource-based policy, you define which S3 on Outposts actions are allowed or denied. S3 on Outposts actions correspond to specific S3 on Outposts API operations. S3 on Outposts actions use the `s3-outposts:` namespace prefix. Requests made to the S3 on Outposts control API in an AWS Region and requests made to the object API endpoints on the Outpost are authenticated by using IAM and authorized against the `s3-outposts:` namespace prefix. To work with S3 on Outposts, configure your IAM users and authorize them against the `s3-outposts:` IAM namespace.

For more information, see [Actions, resources, and condition keys for Amazon S3 on Outposts](#) in the [Service Authorization Reference](#).

Note

- Access control lists (ACLs) are not supported by S3 on Outposts.
- S3 on Outposts defaults to the bucket owner as object owner to help ensure that the owner of a bucket can't be prevented from accessing or deleting objects.
- S3 on Outposts always has S3 Block Public Access enabled to help ensure that objects can never have public access.

For more information about setting up IAM for S3 on Outposts, see the following topics.

Topics

- [Principals for S3 on Outposts policies \(p. 1250\)](#)
- [Resource ARNs for S3 on Outposts \(p. 1250\)](#)
- [Example policies for S3 on Outposts \(p. 1251\)](#)
- [Permissions for S3 on Outposts endpoints \(p. 1252\)](#)

Principals for S3 on Outposts policies

When you create a resource-based policy to grant access to your S3 on Outposts bucket, you must use the `Principal` element to specify the person or application that can make a request for an action or operation on that resource. For S3 on Outposts policies, you can use one of the following principals:

- An AWS account
- An IAM user
- An IAM role
- All principals, by specifying a wildcard character (*) in a policy that uses a `Condition` element to limit access to a specific IP range

Important

You can't write a policy for an S3 on Outposts bucket that uses a wildcard character (*) in the `Principal` element unless the policy also includes a `Condition` that limits access to a specific IP address range. This restriction helps ensure that there is no public access to your S3 on Outposts bucket. For an example, see [Example policies for S3 on Outposts \(p. 1251\)](#).

For more information about the `Principal` element, see [AWS JSON policy elements: Principal](#) in the [IAM User Guide](#).

Resource ARNs for S3 on Outposts

Amazon Resource Names (ARNs) for S3 on Outposts contain the Outpost ID in addition to the AWS Region that the Outpost is homed to, the AWS account ID, and the resource name. To access and perform

actions on your Outposts buckets and objects, you must use one of the ARN formats shown in the following table.

The *partition* value in the ARN refers to a group of AWS Regions. Each AWS account is scoped to one partition. The following are the supported partitions:

- aws – AWS Regions
- aws-us-gov – AWS GovCloud (US) Regions

S3 on Outposts ARN formats

Amazon S3 on Outposts ARN	ARN format	Example
Bucket ARN	arn: <i>partition</i> :s3-outposts: <i>region</i> : <i>account_id</i> :outpost/ <i>outpost_id</i> /bucket/ <i>bucket_name</i>	arn:aws:s3-outposts:us-west-2:123456789012:outpost/op-01ac5d28a6a232904/bucket/DOC-EXAMPLE-BUCKET1
Access point ARN	arn: <i>partition</i> :s3-outposts: <i>region</i> : <i>account_id</i> :outpost/ <i>outpost_id</i> /accesspoint/ <i>accesspoint_name</i>	arn:aws:s3-outposts:us-west-2:123456789012:outpost/op-01ac5d28a6a232904/accesspoint/ <i>access-point-name</i>
Object ARN	arn: <i>partition</i> :s3-outposts: <i>region</i> : <i>account_id</i> :outpost/ <i>outpost_id</i> /bucket/ <i>bucket_name</i> /object/ <i>object_key</i>	arn:aws:s3-outposts:us-west-2:123456789012:outpost/op-01ac5d28a6a232904/bucket/DOC-EXAMPLE-BUCKET1/object/myobject
S3 on Outposts access point object ARN (used in policies)	arn: <i>partition</i> :s3-outposts: <i>region</i> : <i>account_id</i> :outpost/ <i>outpost_id</i> /accesspoint/ <i>accesspoint_name</i> /object/ <i>object_key</i>	arn:aws:s3-outposts:us-west-2:123456789012:outpost/op-01ac5d28a6a232904/accesspoint/ <i>access-point-name</i> /object/ <i>myobject</i>
S3 on Outposts ARN	arn: <i>partition</i> :s3-outposts: <i>region</i> : <i>account_id</i> :outpost/ <i>outpost_id</i>	arn:aws:s3-outposts:us-west-2:123456789012:outpost/op-01ac5d28a6a232904

Example policies for S3 on Outposts

Example : S3 on Outposts bucket policy with an AWS account principal

The following bucket policy uses an AWS account principal to grant access to an S3 on Outposts bucket. To use this bucket policy, replace the *user input placeholders* with your own information.

```
{
    "Version": "2012-10-17",
```

```

"Id": "ExampleBucketPolicy1",
"Statement": [
    {
        "Sid": "statement1",
        "Effect": "Allow",
        "Principal": {
            "AWS": "123456789012"
        },
        "Action": "s3-outposts:*",
        "Resource": "arn:aws:s3-outposts:region:123456789012:outpost/op-01ac5d28a6a232904/bucket/example-outpost-bucket"
    }
]
}

```

Example : S3 on Outposts bucket policy with a wildcard principal (*) and condition key to limit access to a specific IP address range

The following bucket policy uses a wildcard principal (*) with the `aws:SourceIp` condition to limit access to a specific IP address range. To use this bucket policy, replace the *user input placeholders* with your own information.

```

{
    "Version": "2012-10-17",
    "Id": "ExampleBucketPolicy2",
    "Statement": [
        {
            "Sid": "statement1",
            "Effect": "Allow",
            "Principal": { "AWS" : "*" },
            "Action": "s3-outposts:*",
            "Resource": "arn:aws:s3-outposts:region:123456789012:outpost/op-01ac5d28a6a232904/bucket/example-outpost-bucket",
            "Condition" : {
                "IpAddress" : {
                    "aws:SourceIp": "192.0.2.0/24"
                },
                "NotIpAddress" : {
                    "aws:SourceIp": "198.51.100.0/24"
                }
            }
        }
    ]
}

```

Permissions for S3 on Outposts endpoints

S3 on Outposts requires its own permissions in IAM to manage S3 on Outposts endpoint actions.

Note

- For endpoints that use the customer-owned IP address pool (CoIP pool) access type, you also must have permissions to work with IP addresses from your CoIP pool, as described in the following table.
- For shared accounts that access S3 on Outposts by using AWS Resource Access Manager, users in these shared accounts can't create their own endpoints on a shared subnet. If a user in a shared account wants to manage their own endpoints, the shared account must create its own subnet on the Outpost. For more information, see [the section called "Sharing S3 on Outposts" \(p. 1320\)](#).

S3 on Outposts endpoint-related IAM permissions

Action	IAM permissions
CreateEndpoint	<pre>s3-outposts:CreateEndpoint ec2:CreateNetworkInterface ec2:DescribeNetworkInterfaces ec2:DescribeVpcs ec2:DescribeSecurityGroups ec2:DescribeSubnets ec2:CreateTags</pre> <p>For endpoints that are using the on-premises customer-owned IP address pool (CoIP pool) access type, the following additional permissions are required:</p> <pre>s3-outposts:CreateEndpoint ec2:DescribeCoipPools ec2:GetCoipPoolUsage ec2:AllocateAddress ec2:AssociateAddress ec2:DescribeAddresses ec2:DescribeLocalGatewayRouteTableVpcAssociations</pre>
DeleteEndpoint	<pre>s3-outposts:DeleteEndpoint ec2:DeleteNetworkInterface ec2:DescribeNetworkInterfaces</pre> <p>For endpoints that are using the on-premises customer-owned IP address pool (CoIP pool) access type, the following additional permissions are required:</p> <pre>s3-outposts:DeleteEndpoint ec2:DisassociateAddress ec2:DescribeAddresses ec2:ReleaseAddress</pre>
ListEndpoints	s3-outposts>ListEndpoints

Note

You can use resource tags in an IAM policy to manage permissions.

Getting started by using the AWS Management Console

With Amazon S3 on Outposts, you can create S3 buckets on your AWS Outposts and easily store and retrieve objects on premises for applications that require local data access, local data processing, and data residency. S3 on Outposts provides a new storage class, S3 Outposts (OUTPOSTS), which uses the Amazon S3 APIs, and is designed to store data durably and redundantly across multiple devices and servers on your AWS Outposts. You communicate with your Outpost bucket by using an access point and endpoint connection over a virtual private cloud (VPC). You can use the same APIs and features on Outpost buckets as you do on Amazon S3 buckets, including access policies, encryption, and tagging. You can use S3 on Outposts through the AWS Management Console, AWS Command Line Interface (AWS CLI), AWS SDKs, or REST API. For more information, see [What is Amazon S3 on Outposts? \(p. 1241\)](#)

To get started with S3 on Outposts by using the console, see the following topics. To get started by using the AWS CLI or AWS SDK for Java, see [Getting started by using the AWS CLI and SDK for Java \(p. 1255\)](#).

Topics

- [Create a bucket, an access point, and an endpoint \(p. 1254\)](#)
- [Next steps \(p. 1255\)](#)

Create a bucket, an access point, and an endpoint

The following procedure shows you how to create your first bucket in S3 on Outposts. When you create a bucket using the console, you also create an access point and an endpoint associated with the bucket so that you can immediately begin storing objects in your bucket.

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 **Outposts buckets**.
3. Choose **Create Outposts bucket**.
4. For **Bucket name**, enter a Domain Name System (DNS)-compliant name for your bucket.

The bucket name must:

- Be unique within the AWS account, the Outpost, and the AWS Region the Outpost is homed to.
- Be 3–63 characters long.
- Not contain uppercase characters.
- Start with a lowercase letter or number.

After you create the bucket, you can't change its name. For information about naming buckets, see [Bucket naming rules \(p. 118\)](#).

Important

Avoid including sensitive information such as account numbers in the bucket name. The bucket name is visible in the URLs that point to the objects in the bucket.

5. For **Outpost**, choose the Outpost where you want the bucket to reside.
6. (Optional) Add any **optional tags** that you would like to associate with the Outposts bucket. You can use tags to track criteria for individual projects or groups of projects, or to label your buckets by using cost-allocation tags.

By default, all objects stored in your Outposts bucket are stored by using server-side encryption with Amazon S3 managed encryption keys (SSE-S3). You can also explicitly choose to store objects by using server-side encryption with customer-provided encryption keys (SSE-C). To change the encryption type, you must use the REST API, AWS Command Line Interface (AWS CLI), or AWS SDKs.

7. In the **Outposts access point settings** section, enter the access point name.

S3 on Outposts access points simplify managing data access at scale for shared datasets in S3 on Outposts. Access points are named network endpoints that are attached to Outposts buckets that you can use to perform S3 object operations. For more information, see [Access points \(p. 1261\)](#).

Access point names must be unique within the account for this Region and Outpost, and comply with the [Access points restrictions and limitations \(p. 318\)](#).

8. Choose the **VPC** for this Amazon S3 on Outposts access point.

If you don't have a VPC, choose **Create VPC**. For more information, see [Creating access points restricted to a virtual private cloud \(p. 308\)](#).

A virtual private cloud (VPC) enables you to launch AWS resources into a virtual network that you define. This virtual network closely resembles a traditional network that you would operate in your own data center, with the benefits of using the scalable infrastructure of AWS.

9. (Optional for an existing VPC) Choose an **Endpoint subnet** for your endpoint.

A subnet is a range of IP addresses in your VPC. If you don't have the subnet that you want, choose **Create subnet**. For more information, see [Networking for S3 on Outposts \(p. 1260\)](#).

10. (Optional for an existing VPC) Choose an **Endpoint security group** for your endpoint.

A **security group** acts as a virtual firewall to control inbound and outbound traffic.

11. (Optional for an existing VPC) Choose the **Endpoint access type**:

- **Private** – To be used with the VPC.
- **Customer owned IP** – To be used with a customer-owned IP address pool (CoIP pool) from within your on-premises network.

12. (Optional) Specify the **Outpost access point policy**. The console automatically displays the **Amazon Resource Name (ARN)** for the access point, which you can use in the policy.

13. Choose **Create Outposts bucket**.

Note

It can take up to 5 minutes for your Outposts endpoint to be created and your bucket to be ready to use. To configure additional bucket settings, choose **View details**.

Next steps

With Amazon S3 on Outposts, object data is always stored on the Outpost. When AWS installs an Outpost rack, your data stays local to your Outpost to meet data-residency requirements. Your objects never leave your Outpost and are not in an AWS Region. Because the AWS Management Console is hosted in-Region, you can't use the console to upload or manage objects in your Outpost. However, you can use the REST API, AWS Command Line Interface (AWS CLI), and AWS SDKs to upload and manage your objects through your access points.

After you create an S3 on Outposts bucket, access point, and endpoint, you can use the AWS CLI or SDK for Java to upload an object to your bucket. For more information, see [Step 4: Upload an object to an S3 on Outposts bucket \(p. 1258\)](#).

Getting started by using the AWS CLI and SDK for Java

With Amazon S3 on Outposts, you can create S3 buckets on your AWS Outposts and easily store and retrieve objects on premises for applications that require local data access, local data processing, and data residency. S3 on Outposts provides a new storage class, S3 Outposts (OUTPOSTS), which uses the

Amazon S3 APIs, and is designed to store data durably and redundantly across multiple devices and servers on your AWS Outposts. You communicate with your Outpost bucket by using an access point and endpoint connection over a virtual private cloud (VPC). You can use the same APIs and features on Outpost buckets as you do on Amazon S3 buckets, including access policies, encryption, and tagging. You can use S3 on Outposts through the AWS Management Console, AWS Command Line Interface (AWS CLI), AWS SDKs, or REST API. For more information, see [What is Amazon S3 on Outposts? \(p. 1241\)](#)

To get started with S3 on Outposts, you must create a bucket, an access point, and an endpoint. Then, you can upload objects to your bucket. The following examples show you how to get started with S3 on Outposts by using the AWS CLI and SDK for Java. To get started by using the console, see [Getting started by using the AWS Management Console \(p. 1254\)](#).

Topics

- [Step 1: Create a bucket \(p. 1256\)](#)
- [Step 2: Create an access point \(p. 1257\)](#)
- [Step 3: Create an endpoint \(p. 1257\)](#)
- [Step 4: Upload an object to an S3 on Outposts bucket \(p. 1258\)](#)

Step 1: Create a bucket

The following AWS CLI and SDK for Java examples show you how to create an S3 on Outposts bucket.

AWS CLI

Example

The following example creates an S3 on Outposts bucket (`s3-outposts:CreateBucket`) by using the AWS CLI. To run this command, replace the `user input placeholders` with your own information.

```
aws s3control create-bucket --bucket example-outpost-bucket --outpost-id op-01ac5d28a6a232904
```

SDK for Java

Example

The following example creates an S3 on Outposts bucket (`s3-outposts:CreateBucket`) by using the SDK for Java.

```
import com.amazonaws.services.s3control.model.*;

public String createBucket(String bucketName) {

    CreateBucketRequest reqCreateBucket = new CreateBucketRequest()
        .withBucket(bucketName)
        .withOutpostId(OutpostId)
        .withCreateBucketConfiguration(new CreateBucketConfiguration());

    CreateBucketResult respCreateBucket =
        s3ControlClient.createBucket(reqCreateBucket);
    System.out.printf("CreateBucket Response: %s%n", respCreateBucket.toString());

    return respCreateBucket.getBucketArn();
}
```

Step 2: Create an access point

To access your Amazon S3 on Outposts bucket, you must create and configure an access point. These examples show you how to create an access point by using the AWS CLI and the SDK for Java.

Access points simplify managing data access at scale for shared datasets in Amazon S3. Access points are named network endpoints that are attached to buckets that you can use to perform Amazon S3 object operations, such as `GetObject` and `PutObject`. With S3 on Outposts, you must use access points to access any object in an Outposts bucket. Access points support only virtual-host-style addressing.

AWS CLI

Example

The following AWS CLI example creates an access point for an Outposts bucket. To run this command, replace the *user input placeholders* with your own information.

```
aws s3control create-access-point --account-id 123456789012
  --name example-outposts-access-point --bucket "arn:aws:s3-
outposts:region:123456789012:outpost/op-01ac5d28a6a232904/bucket/example-outpost-
bucket" --vpc-configuration VpcId=example-vpc-12345
```

SDK for Java

Example

The following SDK for Java example creates an access point for an Outposts bucket. To use this example, replace the *user input placeholders* with your own information.

```
import com.amazonaws.services.s3control.model.*;

public String createAccessPoint(String bucketArn, String accessPointName) {

    CreateAccessPointRequest reqCreateAP = new CreateAccessPointRequest()
        .withAccountId(AccountId)
        .withBucket(bucketArn)
        .withName(accessPointName)
        .withVpcConfiguration(new VpcConfiguration().withVpcId("vpc-12345"));

    CreateAccessPointResult respCreateAP =
        s3ControlClient.createAccessPoint(reqCreateAP);
    System.out.printf("CreateAccessPoint Response: %s%n", respCreateAP.toString());

    return respCreateAP.getAccessPointArn();
}
```

Step 3: Create an endpoint

To route requests to an Amazon S3 on Outposts access point, you must create and configure an S3 on Outposts endpoint. Each virtual private cloud (VPC) on your Outpost can have one associated endpoint. For more information about endpoint quotas, see [S3 on Outposts network requirements \(p. 1248\)](#). You must create these endpoints to be able to access your Outposts buckets and perform object operations. For more information, see [Endpoints \(p. 1262\)](#).

These examples show you how to create an endpoint by using the AWS CLI and the SDK for Java. For more information about the permissions required to create and manage endpoints, see [Permissions for S3 on Outposts endpoints \(p. 1252\)](#).

AWS CLI

Example

The following AWS CLI example creates an endpoint for an Outpost by using the VPC resource access type. The VPC is derived from the subnet. To run this command, replace the *user input placeholders* with your own information.

```
aws s3outposts create-endpoint --outpost-id op-01ac5d28a6a232904 --subnet-id
subnet-8c7a57c5 --security-group-id sg-ab19e0d1
```

The following AWS CLI example creates an endpoint for an Outpost by using the customer-owned IP address pool (CoIP pool) access type. To run this command, replace the *user input placeholders* with your own information.

```
aws s3outposts create-endpoint --outpost-id op-01ac5d28a6a232904 --subnet-id
subnet-8c7a57c5 --security-group-id sg-ab19e0d1 --access-type CustomerOwnedIp --
customer-owned-ipv4-pool ipv4pool-coip-12345678901234567
```

SDK for Java

Example

The following SDK for Java example creates an endpoint for an Outpost. To use this example, replace the *user input placeholders* with your own information.

```
import com.amazonaws.services.s3outposts.AmazonS3Outposts;
import com.amazonaws.services.s3outposts.AmazonS3OutpostsClientBuilder;
import com.amazonaws.services.s3outposts.model.CreateEndpointRequest;
import com.amazonaws.services.s3outposts.model.CreateEndpointResult;

public void createEndpoint() {
    AmazonS3Outposts s3OutpostsClient = AmazonS3OutpostsClientBuilder
        .standard().build();

    CreateEndpointRequest createEndpointRequest = new CreateEndpointRequest()
        .withOutpostId("op-0d79779cef3c30a40")
        .withSubnetId("subnet-8c7a57c5")
        .withSecurityGroupId("sg-ab19e0d1")
        .withAccessType("CustomerOwnedIp")
        .withCustomerOwnedIpv4Pool("ipv4pool-coip-12345678901234567");
    // Use .withAccessType and .withCustomerOwnedIpv4Pool only when the access type is
    // customer-owned IP address pool (CoIP pool)
    CreateEndpointResult createEndpointResult =
    s3OutpostsClient.createEndpoint(createEndpointRequest);
    System.out.println("Endpoint is created and its ARN is " +
    createEndpointResult.getEndpointArn());
}
```

Step 4: Upload an object to an S3 on Outposts bucket

Objects are the fundamental entities stored in Amazon S3 on Outposts. Every object is contained in a bucket. You must use access points to access any object in an Outpost bucket. When you specify the bucket for object operations, you use the access point Amazon Resource Name (ARN), which includes the AWS Region code for the Region that the Outpost is homed to, the AWS account ID, the Outpost ID, and the access point name. The following example shows the ARN format for S3 on Outposts access points in object operations:

```
arn:aws:s3-outposts:region:account-id:outpost/outpost-id/accesspoint/accesspoint-name
```

For more information about S3 on Outposts ARNs, see [Resource ARNs for S3 on Outposts \(p. 1250\)](#).

With Amazon S3 on Outposts, object data is always stored on the Outpost. When AWS installs an Outpost rack, your data stays local to your Outpost to meet data-residency requirements. Your objects never leave your Outpost and are not in an AWS Region. Because the AWS Management Console is hosted in-Region, you can't use the console to upload or manage objects in your Outpost. However, you can use the REST API, AWS Command Line Interface (AWS CLI), and AWS SDKs to upload and manage your objects through your access points.

The following AWS CLI and AWS SDK for Java examples show you how to upload an object to an S3 on Outposts bucket by using an access point.

AWS CLI

Example

The following example puts an object named `sample-object.xml` into an S3 on Outposts bucket (`s3-outposts:PutObject`) by using the AWS CLI. To use this command, replace each `user input placeholder` with your own information. For more information about this command, see [put-object](#) in the [AWS CLI Reference](#).

```
aws s3api put-object --bucket arn:aws:s3-
outposts:Region:123456789012:outpost/op-01ac5d28a6a232904/accesspoint/example-outposts-
access-point --key sample-object.xml --body sample-object.xml
```

SDK for Java

Example

The following example puts an object into an S3 on Outposts bucket by using the SDK for Java. To use this example, replace each `user input placeholder` with your own information. For more information, see [Uploading objects \(p. 158\)](#).

```
import com.amazonaws.AmazonServiceException;
import com.amazonaws.SdkClientException;
import com.amazonaws.services.s3.AmazonS3;
import com.amazonaws.services.s3.AmazonS3ClientBuilder;
import com.amazonaws.services.s3.model.ObjectMetadata;
import com.amazonaws.services.s3.model.PutObjectRequest;

import java.io.File;

public class PutObject {
    public static void main(String[] args) {
        String accessPointArn = "*** access point ARN ***";
        String stringObjKeyName = "*** String object key name ***";
        String fileObjKeyName = "*** File object key name ***";
        String fileName = "*** Path to file to upload ***";

        try {
            // This code expects that you have AWS credentials set up per:
            // https://docs.aws.amazon.com/sdk-for-java/v1/developer-guide/setup-
            credentials.html
            AmazonS3 s3Client = AmazonS3ClientBuilder.standard()
                .enableUseArnRegion()
                .build();

            // Upload a text string as a new object.
        }
    }
}
```

```
s3Client.putObject(accessPointArn, stringObjKeyName, "Uploaded String
Object");

        // Upload a file as a new object with ContentType and title specified.
        PutObjectRequest request = new PutObjectRequest(accessPointArn,
fileObjKeyName, new File(fileName));
        ObjectMetadata metadata = new ObjectMetadata();
        metadata.setContentType("plain/text");
        metadata.addUserMetadata("title", "someTitle");
        request.setMetadata(metadata);
        s3Client.putObject(request);
    } catch (AmazonServiceException e) {
        // The call was transmitted successfully, but Amazon S3 couldn't process
        // it, so it returned an error response.
        e.printStackTrace();
    } catch (SdkClientException e) {
        // Amazon S3 couldn't be contacted for a response, or the client
        // couldn't parse the response from Amazon S3.
        e.printStackTrace();
    }
}
```

Networking for S3 on Outposts

You can use Amazon S3 on Outposts to store and retrieve objects on-premises for applications that require local data access, data processing, and data residency. This section describes the networking requirements for accessing S3 on Outposts.

Topics

- [Choosing your networking access type \(p. 1260\)](#)
- [Accessing your S3 on Outposts buckets and objects \(p. 1260\)](#)
- [Cross-account elastic network interfaces \(p. 1261\)](#)

Choosing your networking access type

You can access S3 on Outposts from within a VPC or from your on-premises network. You communicate with your Outpost bucket by using an access point and endpoint connection. This connection keeps traffic between your VPC and your S3 on Outposts buckets within the AWS network. When you create an endpoint, you must specify the endpoint access type as either `Private` (for VPC routing) or `CustomerOwnedIp` (for a customer-owned IP address pool [CoIP pool]).

- `Private` (for VPC routing) – If you don't specify the access type, S3 on Outposts uses `Private` by default. With the `Private` access type, instances in your VPC don't require public IP addresses to communicate with resources in your Outpost. You can work with S3 on Outposts from within a VPC. This type of endpoint is not accessible from your on-premises network.
- `CustomerOwnedIp` (for CoIP pool) – If you don't default to the `Private` access type and choose `CustomerOwnedIp`, you must specify an IP address range. You can use this access type to work with S3 on Outposts from both your on-premises network and within a VPC. When accessing S3 on Outposts within a VPC, your traffic is limited to the bandwidth of the local gateway.

Accessing your S3 on Outposts buckets and objects

To access your S3 on Outposts buckets and objects, you must have the following:

- An access point for the VPC.
- An endpoint for the same VPC.
- An active connection between your Outpost and your AWS Region. For more information about how to connect your Outpost to a Region, see [Outpost connectivity to AWS Regions](#) in the *AWS Outposts User Guide*.

For more information about accessing buckets and objects in S3 on Outposts, see [Working with S3 on Outposts buckets \(p. 1261\)](#) and [Working with S3 on Outposts objects \(p. 1286\)](#).

Cross-account elastic network interfaces

S3 on Outposts endpoints are named resources with Amazon Resource Names (ARNs). When these endpoints are created, AWS Outposts sets up multiple cross-account elastic network interfaces. S3 on Outposts cross-account elastic network interfaces are like other network interfaces with one exception: S3 on Outposts associates the cross-account elastic network interfaces to Amazon EC2 instances.

The S3 on Outposts Domain Name System (DNS) load balances your requests over the cross-account elastic network interface. S3 on Outposts creates the cross-account elastic network interface in your AWS account that is visible from the **Network interfaces** pane of the Amazon EC2 console.

For endpoints that use the CoIP pool access type, S3 on Outposts allocates and associates IP addresses with the cross-account elastic network interface from the configured CoIP pool.

Working with S3 on Outposts buckets

With Amazon S3 on Outposts, you can create S3 buckets on your AWS Outposts and easily store and retrieve objects on premises for applications that require local data access, local data processing, and data residency. S3 on Outposts provides a new storage class, S3 Outposts (`OUTPOSTS`), which uses the Amazon S3 APIs, and is designed to store data durably and redundantly across multiple devices and servers on your AWS Outposts. You can use the same APIs and features on Outpost buckets as you do on Amazon S3, including access policies, encryption, and tagging. For more information, see [What is Amazon S3 on Outposts? \(p. 1241\)](#)

You communicate with your Outpost buckets by using an access point and endpoint connection over a virtual private cloud (VPC). To access your S3 on Outposts buckets and objects, you must have an access point for the VPC and an endpoint for the same VPC. For more information, see [Networking for S3 on Outposts \(p. 1260\)](#).

Buckets

In S3 on Outposts, bucket names are unique to an Outpost and require the AWS Region code for the Region the Outpost is homed to, AWS account ID, Outpost ID, and the bucket name to identify them.

```
arn:aws:s3-outposts:region:account-id:outpost/outpost-id/bucket/bucket-name
```

For more information, see [Resource ARNs for S3 on Outposts \(p. 1250\)](#).

Access points

Amazon S3 on Outposts supports virtual private cloud (VPC)-only access points as the only means to access your Outposts buckets.

Access points simplify managing data access at scale for shared datasets in Amazon S3. Access points are named network endpoints that are attached to buckets that you can use to perform Amazon S3 object operations, such as `GetObject` and `PutObject`. With S3 on Outposts, you must use access points to access any object in an Outposts bucket. Access points support only virtual-host-style addressing.

The following example shows the ARN format that you use for S3 on Outposts access points. The access point ARN includes the AWS Region code for the Region the Outpost is homed to, AWS account ID, Outpost ID, and access point name.

```
arn:aws:s3-outposts:region:account-id:outpost/outpost-id/accesspoint/accesspoint-name
```

Endpoints

To route requests to an S3 on Outposts access point, you must create and configure an S3 on Outposts endpoint. With S3 on Outposts endpoints, you can privately connect your VPC to your Outpost bucket. S3 on Outposts endpoints are virtual uniform resource identifiers (URIs) of the entry point to your S3 on Outposts bucket. They are horizontally scaled, redundant, and highly available VPC components.

Each virtual private cloud (VPC) on your Outpost can have one associated endpoint, and you can have up to 100 endpoints per Outpost. You must create these endpoints to be able to access your Outpost buckets and perform object operations. Creating these endpoints also enables the API model and behaviors to be the same by allowing the same operations to work in S3 and S3 on Outposts.

API operations on S3 on Outposts

To manage Outpost bucket API operations, S3 on Outposts hosts a separate endpoint that is distinct from the Amazon S3 endpoint. This endpoint is `s3-outposts.region.amazonaws.com`.

To use Amazon S3 API operations, you must sign the bucket and objects using the correct ARN format. You must pass ARNs to API operations so that Amazon S3 can determine whether the request is for Amazon S3 (`s3-control.region.amazonaws.com`) or for S3 on Outposts (`s3-outposts.region.amazonaws.com`). Based on the ARN format, S3 can then sign and route the request appropriately.

Whenever a request is sent to the Amazon S3 control plane, the SDK extracts the components from the ARN and includes the additional header `x-amz-outpost-id`, with the `outpost-id` value extracted from the ARN. The service name from the ARN is used to sign the request before it is routed to the S3 on Outposts endpoint. This behavior applies to all API operations handled by the `s3control` client.

The following table lists the extended API operations for Amazon S3 on Outposts and their changes relative to Amazon S3.

API	S3 on Outposts parameter value
CreateBucket	Bucket name as ARN, Outpost ID
ListRegionalBuckets	Outpost ID
DeleteBucket	Bucket name as ARN
DeleteBucketLifecycleConfiguration	Bucket name as ARN
GetBucketLifecycleConfiguration	Bucket name as ARN
PutBucketLifecycleConfiguration	Bucket name as ARN
GetBucketPolicy	Bucket name as ARN

API	S3 on Outposts parameter value
PutBucketPolicy	Bucket name as ARN
DeleteBucketPolicy	Bucket name as ARN
GetBucketTagging	Bucket name as ARN
PutBucketTagging	Bucket name as ARN
DeleteBucketTagging	Bucket name as ARN
CreateAccessPoint	Access point name as ARN
DeleteAccessPoint	Access point name as ARN
GetAccessPoint	Access point name as ARN
GetAccessPoint	Access point name as ARN
ListAccessPoints	Access point name as ARN
PutAccessPointPolicy	Access point name as ARN
GetAccessPointPolicy	Access point name as ARN
DeleteAccessPointPolicy	Access point name as ARN

Creating and managing S3 on Outposts buckets

For more information about creating and managing S3 on Outposts buckets, see the following topics.

Topics

- [Creating an S3 on Outposts bucket \(p. 1263\)](#)
- [Adding tags for S3 on Outposts buckets \(p. 1265\)](#)
- [Creating and managing a lifecycle configuration for your Amazon S3 on Outposts bucket \(p. 1266\)](#)
- [Managing access to an Amazon S3 on Outposts bucket using a bucket policy \(p. 1271\)](#)
- [Listing Amazon S3 on Outposts buckets \(p. 1274\)](#)
- [Getting an S3 on Outposts bucket by using the AWS CLI and the SDK for Java \(p. 1275\)](#)
- [Deleting your Amazon S3 on Outposts bucket \(p. 1276\)](#)
- [Working with Amazon S3 on Outposts access points \(p. 1277\)](#)
- [Working with Amazon S3 on Outposts endpoints \(p. 1283\)](#)

Creating an S3 on Outposts bucket

With Amazon S3 on Outposts, you can create S3 buckets on your AWS Outposts and easily store and retrieve objects on premises for applications that require local data access, local data processing, and data residency. S3 on Outposts provides a new storage class, S3 Outposts (`OUTPOSTS`), which uses the Amazon S3 APIs, and is designed to store data durably and redundantly across multiple devices and servers on your AWS Outposts. You communicate with your Outpost bucket by using an access point and endpoint connection over a virtual private cloud (VPC). You can use the same APIs and features on Outpost buckets as you do on Amazon S3 buckets, including access policies, encryption, and tagging. You can use S3 on Outposts through the AWS Management Console, AWS Command Line Interface (AWS CLI), AWS SDKs, or REST API. For more information, see [What is Amazon S3 on Outposts? \(p. 1241\)](#)

Note

The AWS account that creates the bucket owns it and is the only one that can commit actions to it. Buckets have configuration properties, such as Outpost, tag, default encryption, and access point settings. The access point settings include the virtual private cloud (VPC), the access point policy for accessing the objects in the bucket, and other metadata. For more information, see [S3 on Outposts specifications \(p. 1247\)](#).

The following examples show you how to create an S3 on Outposts bucket by using the AWS Management Console, AWS Command Line Interface (AWS CLI), and AWS SDK for Java.

Using the S3 console

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 **Outposts buckets**.
3. Choose **Create Outposts bucket**.
4. For **Bucket name**, enter a Domain Name System (DNS)-compliant name for your bucket.

The bucket name must:

- Be unique within the AWS account, the Outpost, and the AWS Region the Outpost is homed to.
- Be 3–63 characters long.
- Not contain uppercase characters.
- Start with a lowercase letter or number.

After you create the bucket, you can't change its name. For information about naming buckets, see [Bucket naming rules \(p. 118\)](#).

Important

Avoid including sensitive information such as account numbers in the bucket name. The bucket name is visible in the URLs that point to the objects in the bucket.

5. For **Outpost**, choose the Outpost where you want the bucket to reside.
6. (Optional) Add any **optional tags** that you would like to associate with the Outposts bucket. You can use tags to track criteria for individual projects or groups of projects, or to label your buckets by using cost-allocation tags.

By default, all objects stored in your Outposts bucket are stored by using server-side encryption with Amazon S3 managed encryption keys (SSE-S3). You can also explicitly choose to store objects by using server-side encryption with customer-provided encryption keys (SSE-C). To change the encryption type, you must use the REST API, AWS Command Line Interface (AWS CLI), or AWS SDKs.

7. In the **Outposts access point settings** section, enter the access point name.

S3 on Outposts access points simplify managing data access at scale for shared datasets in S3 on Outposts. Access points are named network endpoints that are attached to Outposts buckets that you can use to perform S3 object operations. For more information, see [Access points \(p. 1261\)](#).

Access point names must be unique within the account for this Region and Outpost, and comply with the [Access points restrictions and limitations \(p. 318\)](#).

8. Choose the **VPC** for this Amazon S3 on Outposts access point.

If you don't have a VPC, choose **Create VPC**. For more information, see [Creating access points restricted to a virtual private cloud \(p. 308\)](#).

A virtual private cloud (VPC) enables you to launch AWS resources into a virtual network that you define. This virtual network closely resembles a traditional network that you would operate in your own data center, with the benefits of using the scalable infrastructure of AWS.

9. (Optional for an existing VPC) Choose an **Endpoint subnet** for your endpoint.

A subnet is a range of IP addresses in your VPC. If you don't have the subnet that you want, choose **Create subnet**. For more information, see [Networking for S3 on Outposts \(p. 1260\)](#).

10. (Optional for an existing VPC) Choose an **Endpoint security group** for your endpoint.

A [security group](#) acts as a virtual firewall to control inbound and outbound traffic.

11. (Optional for an existing VPC) Choose the **Endpoint access type**:

- **Private** – To be used with the VPC.
- **Customer owned IP** – To be used with a customer-owned IP address pool (CoIP pool) from within your on-premises network.

12. (Optional) Specify the **Outpost access point policy**. The console automatically displays the **Amazon Resource Name (ARN)** for the access point, which you can use in the policy.

13. Choose **Create Outposts bucket**.

Note

It can take up to 5 minutes for your Outposts endpoint to be created and your bucket to be ready to use. To configure additional bucket settings, choose [View details](#).

Using the AWS CLI

Example

The following example creates an S3 on Outposts bucket (`s3-outposts:CreateBucket`) by using the AWS CLI. To run this command, replace the `user input placeholders` with your own information.

```
aws s3control create-bucket --bucket example-outpost-bucket --outpost-id op-01ac5d28a6a232904
```

Using the AWS SDK for Java

Example

The following example creates an S3 on Outposts bucket (`s3-outposts:CreateBucket`) by using the SDK for Java.

```
import com.amazonaws.services.s3control.model.*;  
  
public String createBucket(String bucketName) {  
  
    CreateBucketRequest reqCreateBucket = new CreateBucketRequest()  
        .withBucket(bucketName)  
        .withOutpostId(OutpostId)  
        .withCreateBucketConfiguration(new CreateBucketConfiguration());  
  
    CreateBucketResult respCreateBucket = s3ControlClient.createBucket(reqCreateBucket);  
    System.out.printf("CreateBucket Response: %s%n", respCreateBucket.toString());  
  
    return respCreateBucket.getBucketArn();  
}
```

Adding tags for S3 on Outposts buckets

You can add tags for your Amazon S3 on Outposts buckets to track storage costs and other criteria for individual projects or groups of projects.

Note

The AWS account that creates the bucket owns it and is the only one that can change its tags.

Using the S3 console

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 **Outposts buckets**.
3. Choose the Outposts bucket whose tags you want to edit.
4. Choose the **Properties** tab.
5. Under **Tags**, choose **Edit**.
6. Choose **Add new tag**, and enter the **Key** and optional **Value**.

Add any tags that you would like to associate with an Outposts bucket to track other criteria for individual projects or groups of projects.

7. Choose **Save changes**.

Using the AWS CLI

The following AWS CLI example applies a tagging configuration to an S3 on Outposts bucket by using a JSON document in the current folder that specifies tags (`tagging.json`). To use this example, replace each `user input placeholder` with your own information.

```
aws s3control put-bucket-tagging --account-id 123456789012 --bucket arn:aws:s3-outposts:region:123456789012:outpost/op-01ac5d28a6a232904/bucket/example-outpost-bucket --tagging file://tagging.json

tagging.json

{
    "TagSet": [
        {
            "Key": "organization",
            "Value": "marketing"
        }
    ]
}
```

The following AWS CLI example applies a tagging configuration to an S3 on Outposts bucket directly from the command line.

```
aws s3control put-bucket-tagging --account-id 123456789012 --bucket arn:aws:s3-outposts:region:123456789012:outpost/op-01ac5d28a6a232904/bucket/example-outpost-bucket --tagging 'TagSet=[{Key=organization,Value=marketing}]'
```

For more information about this command, see [put-bucket-tagging](#) in the *AWS CLI Reference*.

Creating and managing a lifecycle configuration for your Amazon S3 on Outposts bucket

Lifecycle rules for Amazon S3 on Outposts buckets are limited to object deletion. You can use lifecycle rules to define when to initiate object deletion based on age or date. You can create, enable, disable, or delete a lifecycle rule.

Note

The AWS account that creates the bucket owns it and is the only one that can create, enable, disable, or delete a lifecycle rule.

To create and manage the lifecycle configuration for your S3 on Outposts bucket, see the following topics.

Topics

- [Creating and managing a lifecycle rule by using the AWS Management Console \(p. 1267\)](#)
- [Creating and managing a lifecycle configuration by using the AWS CLI and SDK for Java \(p. 1268\)](#)

Creating and managing a lifecycle rule by using the AWS Management Console

Lifecycle rules for Amazon S3 on Outposts buckets are limited to object deletion. You can use lifecycle rules to define when to initiate object deletion based on age or date. You can create, enable, disable, or delete a lifecycle rule.

Note

The AWS account that creates the bucket owns it and is the only one that can create, enable, disable, or delete a lifecycle rule.

To create and manage a lifecycle rule for an S3 on Outposts bucket by using the AWS Management Console, see the following topics.

Topics

- [Creating a lifecycle rule \(p. 1267\)](#)
- [Enabling a lifecycle rule \(p. 1268\)](#)
- [Editing a lifecycle rule \(p. 1268\)](#)
- [Deleting a lifecycle rule \(p. 1268\)](#)

Creating a lifecycle rule

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 **Outposts buckets**.
3. Choose the Outposts bucket that you want to create a lifecycle rule for.
4. Choose the **Management** tab, and then choose **Create Lifecycle rule**.
5. In the **Lifecycle rule configuration** section:
 - a. Enter the **Lifecycle rule name**.
 - b. Choose **Rule scope**.

Important

If you want the rule to apply to specific objects, you must use a filter to identify those objects. Choose **Limit the scope to specific objects or tags**. To limit the scope, do the following:

- Add a prefix filter to limit the scope of this rule to a single prefix.
- Add tags to limit the scope of this rule to the key-value pairs that you add.

6. In the **Lifecycle rule trigger** section, choose the **rule trigger** based on a specific date or object's age.

Enabling a lifecycle rule

To enable or disable a bucket lifecycle rule

1. Open the Amazon S3 console at <https://console.aws.amazon.com/s3/>.
2. In the left navigation pane, choose **Outposts buckets**.
3. Choose the Outposts bucket that you want to enable or disable a lifecycle rule for.
4. Choose the **Management** tab, and then choose the **Lifecycle rule** that you want to enable or disable.
5. For **Action**, choose **Enable or disable rule**.

Editing a lifecycle rule

1. Open the Amazon S3 console at <https://console.aws.amazon.com/s3/>.
2. In the left navigation pane, choose **Outposts buckets**.
3. Choose the Outposts bucket that you want to edit a lifecycle rule for.
4. Choose the **Management** tab, and then choose the **Lifecycle rule** that you want to edit.
5. In the **Lifecycle rule configuration** section, do the following:
 - a. Update the **Lifecycle rule name**.
 - b. Update **Rule scope**.

Important

If you want the rule to apply to specific objects, you must use a filter to identify those objects. Choose **Limit the scope to specific objects or tags**. To limit the scope, do the following:

- Add a prefix filter to limit the scope of this rule to a single prefix.

- Add tags to limit the scope of this rule to the key-value pairs that you add.

6. In the **Lifecycle rule trigger** section, update the **rule trigger** based on a specific date or object's age.

Deleting a lifecycle rule

1. Open the Amazon S3 console at <https://console.aws.amazon.com/s3/>.
2. In the left navigation pane, choose **Outposts buckets**.
3. Choose the Outposts bucket that you want to delete a lifecycle rule for.
4. Choose the **Management** tab, and then choose the **Lifecycle rule** that you want to delete.
5. Choose **Delete**.

Creating and managing a lifecycle configuration by using the AWS CLI and SDK for Java

Lifecycle rules for Amazon S3 on Outposts buckets are limited to object deletion. You can use lifecycle rules to define when to initiate object deletion based on age or date. You can create, enable, disable, or delete a lifecycle rule.

Note

The AWS account that creates the bucket owns it and is the only one that can create, enable, disable, or delete a lifecycle rule.

To create and manage a lifecycle configuration for an S3 on Outposts bucket by using the AWS Command Line Interface (AWS CLI) and the AWS SDK for Java, see the following examples.

Topics

- [PUT a lifecycle configuration \(p. 1269\)](#)
- [GET the lifecycle configuration on an S3 on Outposts bucket \(p. 1270\)](#)

PUT a lifecycle configuration

AWS CLI

The following AWS CLI example puts a lifecycle configuration policy on an Outposts bucket. This policy specifies that all objects that have the flagged prefix (*myprefix*) and tags expire after 10 days. To use this example, replace each *user input placeholder* with your own information.

1. Save the lifecycle configuration policy to a JSON file. In this example, the file is named `lifecycle1.json`.

```
{  
    "Rules": [  
        {  
            "ID": "id-1",  
            "Filter": {  
                "And": {  
                    "Prefix": "myprefix",  
                    "Tags": [  
                        {  
                            "Value": "mytagvalue1",  
                            "Key": "mytagkey1"  
                        },  
                        {  
                            "Value": "mytagvalue2",  
                            "Key": "mytagkey2"  
                        }  
                    ]  
                }  
            },  
            "Status": "Enabled",  
            "Expiration": {  
                "Days": 10  
            }  
        }  
    ]  
}  
]S3OutpostsPutBucketLifecycleConfigurationCLI
```

2. Submit the JSON file as part of the `put-bucket-lifecycle-configuration` CLI command. To use this command, replace each *user input placeholder* with your own information. For more information about this command, see [put-bucket-lifecycle-configuration](#) in the [AWS CLI Reference](#).

```
aws s3control put-bucket-lifecycle-configuration --account-id 123456789012 --  
bucket arn:aws:s3-outposts:region:123456789012:outpost/op-01ac5d28a6a232904/  
bucket/example-outpost-bucket --lifecycle-configuration file://lifecycle1.json
```

SDK for Java

The following SDK for Java example puts a lifecycle configuration on an Outposts bucket. This lifecycle configuration specifies that all objects that have the flagged prefix (*myprefix*) and tags expire after 10 days. To use this example, replace each *user input placeholder* with your own information. For more information, see [PutBucketLifecycleConfiguration](#) in the [Amazon Simple Storage Service API Reference](#).

```
import com.amazonaws.services.s3control.model.*;

public void putBucketLifecycleConfiguration(String bucketArn) {

    S3Tag tag1 = new S3Tag().withKey("mytagkey1").withValue("mytagvalue1");
    S3Tag tag2 = new S3Tag().withKey("mytagkey2").withValue("mytagvalue2");

    LifecycleRuleFilter lifecycleRuleFilter = new LifecycleRuleFilter()
        .withAnd(new LifecycleRuleAndOperator()
            .withPrefix("myprefix")
            .withTags(tag1, tag2));

    LifecycleExpiration lifecycleExpiration = new LifecycleExpiration()
        .withExpiredObjectDeleteMarker(false)
        .withDays(10);

    LifecycleRule lifecycleRule = new LifecycleRule()
        .withStatus("Enabled")
        .withFilter(lifecycleRuleFilter)
        .withExpiration(lifecycleExpiration)
        .withID("id-1");

    LifecycleConfiguration lifecycleConfiguration = new LifecycleConfiguration()
        .withRules(lifecycleRule);

    PutBucketLifecycleConfigurationRequest reqPutBucketLifecycle = new
    PutBucketLifecycleConfigurationRequest()
        .withAccountId(AccountId)
        .withBucket(bucketArn)
        .withLifecycleConfiguration(lifecycleConfiguration);

    PutBucketLifecycleConfigurationResult respPutBucketLifecycle =
    s3ControlClient.putBucketLifecycleConfiguration(reqPutBucketLifecycle);
    System.out.printf("PutBucketLifecycleConfiguration Response: %s%n",
    respPutBucketLifecycle.toString());

}
```

GET the lifecycle configuration on an S3 on Outposts bucket

AWS CLI

The following AWS CLI example gets a lifecycle configuration on an Outposts bucket. To use this command, replace each *user input placeholder* with your own information. For more information about this command, see [get-bucket-lifecycle-configuration](#) in the *AWS CLI Reference*.

```
aws s3control get-bucket-lifecycle-configuration --account-id 123456789012 --bucket
arn:aws:s3-outposts:<your-region>:123456789012:outpost/op-01ac5d28a6a232904/
bucket/example-outpost-bucket
```

SDK for Java

The following SDK for Java example gets a lifecycle configuration for an Outposts bucket. For more information, see [GetBucketLifecycleConfiguration](#) in the *Amazon Simple Storage Service API Reference*.

```
import com.amazonaws.services.s3control.model.*;
```

```
public void getBucketLifecycleConfiguration(String bucketArn) {  
  
    GetBucketLifecycleConfigurationRequest reqGetBucketLifecycle = new  
    GetBucketLifecycleConfigurationRequest()  
        .withAccountId(AccountId)  
        .withBucket(bucketArn);  
  
    GetBucketLifecycleConfigurationResult respGetBucketLifecycle =  
    s3ControlClient.getBucketLifecycleConfiguration(reqGetBucketLifecycle);  
    System.out.printf("GetBucketLifecycleConfiguration Response: %s%n",  
    respGetBucketLifecycle.toString());  
  
}
```

Managing access to an Amazon S3 on Outposts bucket using a bucket policy

A bucket policy is a resource-based AWS Identity and Access Management (IAM) policy that you can use to grant access permissions to your bucket and the objects in it. Only the bucket owner can associate a policy with a bucket. The permissions attached to the bucket apply to all of the objects in the bucket that are owned by the bucket owner. Bucket policies are limited to 20 KB in size. For more information, see [Bucket policy \(p. 1243\)](#).

You can update your bucket policy to manage access to your Amazon S3 on Outposts bucket. For more information, see the following topics.

Topics

- [Adding or editing a bucket policy for an Amazon S3 on Outposts bucket \(p. 1271\)](#)
- [Viewing the bucket policy for your Amazon S3 on Outposts bucket \(p. 1273\)](#)
- [Deleting the bucket policy for your Amazon S3 on Outposts bucket \(p. 1273\)](#)

Adding or editing a bucket policy for an Amazon S3 on Outposts bucket

A bucket policy is a resource-based AWS Identity and Access Management (IAM) policy that you can use to grant access permissions to your bucket and the objects in it. Only the bucket owner can associate a policy with a bucket. The permissions attached to the bucket apply to all of the objects in the bucket that are owned by the bucket owner. Bucket policies are limited to 20 KB in size. For more information, see [Bucket policy \(p. 1243\)](#).

The following topics show you how to update your Amazon S3 on Outposts bucket policy by using the AWS Management Console, AWS Command Line Interface (AWS CLI), or AWS SDK for Java.

Using the S3 console

To create or edit a bucket policy

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 **Outposts buckets**.
3. Choose the Outposts bucket whose bucket policy you want to edit.
4. Choose the **Permissions** tab.

5. In the **Outposts bucket policy** section, to create or edit new policy, choose **Edit**.

You can now add or edit the S3 on Outposts bucket policy. For more information, see [Setting up IAM with S3 on Outposts \(p. 1249\)](#).

Using the AWS CLI

The following AWS CLI example puts a policy on an Outposts bucket.

1. Save the following bucket policy to a JSON file. In this example, the file is named `policy1.json`. Replace the `user input placeholders` with your own information.

```
{  
    "Version": "2012-10-17",  
    "Id": "testBucketPolicy",  
    "Statement": [  
        {  
            "Sid": "st1",  
            "Effect": "Allow",  
            "Principal": {  
                "AWS": "123456789012"  
            },  
            "Action": "s3-outposts:*",  
            "Resource": "arn:aws:s3-  
outposts:region:123456789012:outpost/op-01ac5d28a6a232904/bucket/example-outpost-  
bucket"  
        }  
    ]  
}
```

2. Submit the JSON file as part of the `put-bucket-policy` CLI command. To run this command, replace the `user input placeholders` with your own information.

```
aws s3control put-bucket-policy --account-id 123456789012 --bucket arn:aws:s3-  
outposts:region:123456789012:outpost/op-01ac5d28a6a232904/bucket/example-outpost-bucket  
--policy file://policy1.json
```

Using the AWS SDK for Java

The following SDK for Java example puts a policy on an Outposts bucket.

```
import com.amazonaws.services.s3control.model.*;  
  
public void putBucketPolicy(String bucketArn) {  
  
    String policy = "{\"Version\": \"2012-10-17\", \"Id\": \"testBucketPolicy\", \"Statement\":[{\"Sid\": \"st1\", \"Effect\": \"Allow\", \"Principal\": {\"AWS\": \"\" + AccountId + \"\"}, \"Action\": \"s3-outposts:*\", \"Resource\": \"\" + bucketArn + \"\"}]}";  
  
    PutBucketPolicyRequest reqPutBucketPolicy = new PutBucketPolicyRequest()  
        .withAccountId(AccountId)  
        .withBucket(bucketArn)  
        .withPolicy(policy);  
  
    PutBucketPolicyResult respPutBucketPolicy =  
        s3ControlClient.putBucketPolicy(reqPutBucketPolicy);  
    System.out.printf("PutBucketPolicy Response: %s%n", respPutBucketPolicy.toString());  
}
```

Viewing the bucket policy for your Amazon S3 on Outposts bucket

A bucket policy is a resource-based AWS Identity and Access Management (IAM) policy that you can use to grant access permissions to your bucket and the objects in it. Only the bucket owner can associate a policy with a bucket. The permissions attached to the bucket apply to all of the objects in the bucket that are owned by the bucket owner. Bucket policies are limited to 20 KB in size. For more information, see [Bucket policy \(p. 1243\)](#).

The following topics show you how to view your Amazon S3 on Outposts bucket policy by using the AWS Management Console, AWS Command Line Interface (AWS CLI), or AWS SDK for Java.

Using the S3 console

To create or edit a bucket policy

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 **Outposts buckets**.
3. Choose the Outposts bucket whose permission you want to edit.
4. Choose the **Permissions** tab.
5. In the **Outposts bucket policy** section, you can review your existing bucket policy. For more information, see [Setting up IAM with S3 on Outposts \(p. 1249\)](#).

Using the AWS CLI

The following AWS CLI example gets a policy for an Outposts bucket. To run this command, replace the *user input placeholders* with your own information.

```
aws s3control get-bucket-policy --account-id 123456789012 --bucket arn:aws:s3-outposts:region:123456789012:outpost/op-01ac5d28a6a232904/bucket/example-outpost-bucket
```

Using the AWS SDK for Java

The following SDK for Java example gets a policy for an Outposts bucket.

```
import com.amazonaws.services.s3control.model.*;  
  
public void getBucketPolicy(String bucketArn) {  
  
    GetBucketPolicyRequest reqGetBucketPolicy = new GetBucketPolicyRequest()  
        .withAccountId(AccountId)  
        .withBucket(bucketArn);  
  
    GetBucketPolicyResult respGetBucketPolicy =  
        s3ControlClient.getBucketPolicy(reqGetBucketPolicy);  
    System.out.printf("GetBucketPolicy Response: %s%n", respGetBucketPolicy.toString());  
}
```

Deleting the bucket policy for your Amazon S3 on Outposts bucket

A bucket policy is a resource-based AWS Identity and Access Management (IAM) policy that you can use to grant access permissions to your bucket and the objects in it. Only the bucket owner can associate a policy with a bucket. The permissions attached to the bucket apply to all of the objects in the bucket that

are owned by the bucket owner. Bucket policies are limited to 20 KB in size. For more information, see [Bucket policy \(p. 1243\)](#).

The following topics show you how to view your Amazon S3 on Outposts bucket policy by using the AWS Management Console or AWS Command Line Interface (AWS CLI).

Using the S3 console

To delete a bucket policy

1. Open the Amazon S3 console at <https://console.aws.amazon.com/s3/>.
2. In the left navigation pane, choose **Outposts buckets**.
3. Choose the Outposts bucket whose permission you want to edit.
4. Choose the **Permissions** tab.
5. In the **Outposts bucket policy** section, choose **Delete**.
6. Confirm the deletion.

Using the AWS CLI

The following example deletes the bucket policy for an S3 on Outposts bucket (`s3-outposts:DeleteBucket`) by using the AWS CLI. To run this command, replace the `user input placeholders` with your own information.

```
aws s3control delete-bucket-policy --account-id 123456789012 --bucket arn:aws:s3-outposts:region:123456789012:outpost/op-01ac5d28a6a232904/bucket/example-outpost-bucket
```

Listing Amazon S3 on Outposts buckets

With Amazon S3 on Outposts, you can create S3 buckets on your AWS Outposts and easily store and retrieve objects on premises for applications that require local data access, local data processing, and data residency. S3 on Outposts provides a new storage class, S3 Outposts (OUTPOSTS), which uses the Amazon S3 APIs, and is designed to store data durably and redundantly across multiple devices and servers on your AWS Outposts. You communicate with your Outpost bucket by using an access point and endpoint connection over a virtual private cloud (VPC). You can use the same APIs and features on Outpost buckets as you do on Amazon S3 buckets, including access policies, encryption, and tagging. You can use S3 on Outposts through the AWS Management Console, AWS Command Line Interface (AWS CLI), AWS SDKs, or REST API. For more information, see [What is Amazon S3 on Outposts? \(p. 1241\)](#)

For more information about working with buckets in S3 on Outposts, see [Working with S3 on Outposts buckets \(p. 1261\)](#).

The following examples show you how to return a list of your S3 on Outposts buckets by using the AWS Management Console, AWS CLI, and AWS SDK for Java.

Using the S3 console

1. Open the Amazon S3 console at <https://console.aws.amazon.com/s3/>.
2. In the left navigation pane, choose **Outposts buckets**.
3. Under **Outposts buckets**, review your list of S3 on Outposts buckets.

Using the AWS CLI

The following AWS CLI example gets a list of buckets in an Outpost. To use this command, replace each `user input placeholder` with your own information. For more information about this command, see [list-regional-buckets](#) in the [AWS CLI Reference](#).

```
aws s3control list-regional-buckets --account-id 123456789012 --outpost-id op-01ac5d28a6a232904
```

Using the AWS SDK for Java

The following SDK for Java example gets a list of buckets in an Outpost. For more information, see [ListRegionalBuckets](#) in the *Amazon Simple Storage Service API Reference*.

```
import com.amazonaws.services.s3control.model.*;  
  
public void listRegionalBuckets() {  
  
    ListRegionalBucketsRequest reqListBuckets = new ListRegionalBucketsRequest()  
        .withAccountId(AccountId)  
        .withOutpostId(OutpostId);  
  
    ListRegionalBucketsResult respListBuckets =  
    s3ControlClient.listRegionalBuckets(reqListBuckets);  
    System.out.printf("ListRegionalBuckets Response: %s%n", respListBuckets.toString());  
}
```

Getting an S3 on Outposts bucket by using the AWS CLI and the SDK for Java

With Amazon S3 on Outposts, you can create S3 buckets on your AWS Outposts and easily store and retrieve objects on premises for applications that require local data access, local data processing, and data residency. S3 on Outposts provides a new storage class, S3 Outposts (OUTPOSTS), which uses the Amazon S3 APIs, and is designed to store data durably and redundantly across multiple devices and servers on your AWS Outposts. You communicate with your Outpost bucket by using an access point and endpoint connection over a virtual private cloud (VPC). You can use the same APIs and features on Outpost buckets as you do on Amazon S3 buckets, including access policies, encryption, and tagging. You can use S3 on Outposts through the AWS Management Console, AWS Command Line Interface (AWS CLI), AWS SDKs, or REST API. For more information, see [What is Amazon S3 on Outposts? \(p. 1241\)](#)

The following examples show you how to get an S3 on Outposts bucket by using the AWS CLI and AWS SDK for Java.

Note

When you're working with Amazon S3 on Outposts through the AWS CLI or AWS SDKs, you provide the access point ARN for the Outpost in place of the bucket name. The access point ARN takes the following form, where *region* is the AWS Region code for the Region that the Outpost is homed to:

`arn:aws:s3-outposts:region:123456789012:outpost/op-01ac5d28a6a232904/accesspoint/example-outposts-access-point`

For more information about S3 on Outposts ARNs, see [Resource ARNs for S3 on Outposts \(p. 1250\)](#).

Using the AWS CLI

The following S3 on Outposts example gets a bucket by using the AWS CLI. To use this command, replace each *user input placeholder* with your own information. For more information about this command, see [get-bucket](#) in the *AWS CLI Reference*.

```
aws s3control get-bucket --account-id 123456789012 --bucket "arn:aws:s3-outposts:region:123456789012:outpost/op-01ac5d28a6a232904/bucket/example-outpost-bucket"
```

Using the AWS SDK for Java

The following S3 on Outposts example gets a bucket by using the SDK for Java. For more information, see [GetBucket](#) in the *Amazon Simple Storage Service API Reference*.

```
import com.amazonaws.services.s3control.model.*;  
  
public void getBucket(String bucketArn) {  
  
    GetBucketRequest reqGetBucket = new GetBucketRequest()  
        .withBucket(bucketArn)  
        .withAccountId(AccountId);  
  
    GetBucketResult respGetBucket = s3ControlClient.getBucket(reqGetBucket);  
    System.out.printf("GetBucket Response: %s%n", respGetBucket.toString());  
}
```

Deleting your Amazon S3 on Outposts bucket

With Amazon S3 on Outposts, you can create S3 buckets on your AWS Outposts and easily store and retrieve objects on premises for applications that require local data access, local data processing, and data residency. S3 on Outposts provides a new storage class, S3 Outposts (OUTPOSTS), which uses the Amazon S3 APIs, and is designed to store data durably and redundantly across multiple devices and servers on your AWS Outposts. You communicate with your Outpost bucket by using an access point and endpoint connection over a virtual private cloud (VPC). You can use the same APIs and features on Outpost buckets as you do on Amazon S3 buckets, including access policies, encryption, and tagging. You can use S3 on Outposts through the AWS Management Console, AWS Command Line Interface (AWS CLI), AWS SDKs, or REST API. For more information, see [What is Amazon S3 on Outposts? \(p. 1241\)](#)

For more information about working with buckets in S3 on Outposts, see [Working with S3 on Outposts buckets \(p. 1261\)](#).

The AWS account that creates the bucket owns it and is the only one that can delete it.

Note

- Outposts buckets must be empty before they can be deleted.

The Amazon S3 console doesn't support S3 on Outposts object actions. To delete objects in an S3 on Outposts bucket, you must use the REST API, AWS CLI, or AWS SDKs.

- Before you can delete an Outposts bucket, you must delete any Outposts access points for the bucket. For more information, see [Deleting an access point \(p. 1280\)](#).
- You cannot recover a bucket after it has been deleted.

The following examples show you how to delete an S3 on Outposts bucket by using the AWS Management Console and AWS Command Line Interface (AWS CLI).

Using the S3 console

- Sign in to the AWS Management Console and open the Amazon S3 console at <https://console.aws.amazon.com/s3/>.
- In the left navigation pane, choose **Outposts buckets**.
- Choose the bucket that you want to delete, and choose **Delete**.
- Confirm the deletion.

Using the AWS CLI

The following example deletes an S3 on Outposts bucket (`s3-outposts:DeleteBucket`) by using the AWS CLI. To run this command, replace the `user input placeholders` with your own information.

```
aws s3control delete-bucket --account-id 123456789012 --bucket arn:aws:s3-outposts:region:123456789012:outpost/op-01ac5d28a6a232904/bucket/example-outpost-bucket
```

Working with Amazon S3 on Outposts access points

To access your Amazon S3 on Outposts bucket, you must create and configure an access point.

Access points simplify managing data access at scale for shared datasets in Amazon S3. Access points are named network endpoints that are attached to buckets that you can use to perform Amazon S3 object operations, such as `GetObject` and `PutObject`. With S3 on Outposts, you must use access points to access any object in an Outposts bucket. Access points support only virtual-host-style addressing.

Note

The AWS account that creates the Outposts bucket owns it and is the only one that can assign access points to it.

The following sections describe how to create and manage access points for S3 on Outposts buckets.

Topics

- [Creating an S3 on Outposts access point \(p. 1277\)](#)
- [Viewing information about an access point configuration \(p. 1278\)](#)
- [View a list of your Amazon S3 on Outposts access points \(p. 1279\)](#)
- [Deleting an access point \(p. 1280\)](#)
- [Adding or editing an access point policy \(p. 1280\)](#)
- [Viewing an access point policy for an S3 on Outposts access point \(p. 1282\)](#)

Creating an S3 on Outposts access point

To access your Amazon S3 on Outposts bucket, you must create and configure an access point.

Access points simplify managing data access at scale for shared datasets in Amazon S3. Access points are named network endpoints that are attached to buckets that you can use to perform Amazon S3 object operations, such as `GetObject` and `PutObject`. With S3 on Outposts, you must use access points to access any object in an Outposts bucket. Access points support only virtual-host-style addressing.

The following examples show you how to create an S3 on Outposts access point by using the AWS Management Console, AWS Command Line Interface (AWS CLI), and AWS SDK for Java.

Note

The AWS account that creates the Outposts bucket owns it and is the only one that can assign access points to it.

Using the S3 console

1. Open the Amazon S3 console at <https://console.aws.amazon.com/s3/>.
2. In the left navigation pane, choose **Outposts buckets**.
3. Choose the Outposts bucket that you want to create an Outposts access point for.
4. Choose the **Outposts access points** tab.
5. In the **Outposts access points** section, choose **Create Outposts access point**.

6. In **Outposts access point settings**, enter a name for the access point, and then choose the virtual private cloud (VPC) for the access point.
7. If you want to add a policy for your access point, enter it in the **Outposts access point policy** section.

For more information, see [Setting up IAM with S3 on Outposts \(p. 1249\)](#).

Using the AWS CLI

Example

The following AWS CLI example creates an access point for an Outposts bucket. To run this command, replace the *user input placeholders* with your own information.

```
aws s3control create-access-point --account-id 123456789012 --name example-outposts-access-point --bucket "arn:aws:s3-outposts:region:123456789012:outpost/op-01ac5d28a6a232904/bucket/example-outpost-bucket" --vpc-configuration VpcId=example-vpc-12345
```

Using the AWS SDK for Java

Example

The following SDK for Java example creates an access point for an Outposts bucket. To use this example, replace the *user input placeholders* with your own information.

```
import com.amazonaws.services.s3control.model.*;  
  
public String createAccessPoint(String bucketArn, String accessPointName) {  
  
    CreateAccessPointRequest reqCreateAP = new CreateAccessPointRequest()  
        .withAccountId(AccountId)  
        .withBucket(bucketArn)  
        .withName(accessPointName)  
        .withVpcConfiguration(new VpcConfiguration().withVpcId("vpc-12345"));  
  
    CreateAccessPointResult respCreateAP = s3ControlClient.createAccessPoint(reqCreateAP);  
    System.out.printf("CreateAccessPoint Response: %s%n", respCreateAP.toString());  
  
    return respCreateAP.getAccessPointArn();  
}
```

Viewing information about an access point configuration

Access points simplify managing data access at scale for shared datasets in Amazon S3. Access points are named network endpoints that are attached to buckets that you can use to perform Amazon S3 object operations, such as `GetObject` and `PutObject`. With S3 on Outposts, you must use access points to access any object in an Outposts bucket. Access points support only virtual-host-style addressing.

The following topics show you how to return configuration information for an S3 on Outposts access point by using the AWS Management Console, AWS Command Line Interface (AWS CLI), and AWS SDK for Java.

Using the S3 console

1. Open the Amazon S3 console at <https://console.aws.amazon.com/s3/>.
2. In the left navigation pane, choose **Outposts access points**.

3. Choose the Outposts access point that you want to view configuration details for.
4. Under **Outposts access point overview**, review the access point configuration details.

Using the AWS CLI

The following AWS CLI example gets an access point for an Outposts bucket. Replace the *user input placeholders* with your own information.

```
aws s3control get-access-point --account-id 123456789012 --name arn:aws:s3-outposts:region:123456789012:outpost/op-01ac5d28a6a232904/accesspoint/example-outposts-access-point
```

Using the AWS SDK for Java

The following SDK for Java example gets an access point for an Outposts bucket.

```
import com.amazonaws.services.s3control.model.*;  
  
public void getAccessPoint(String accessPointArn) {  
  
    GetAccessPointRequest reqGetAP = new GetAccessPointRequest()  
        .withAccountId(AccountId)  
        .withName(accessPointArn);  
  
    GetAccessPointResult respGetAP = s3ControlClient.getAccessPoint(reqGetAP);  
    System.out.printf("GetAccessPoint Response: %s%n", respGetAP.toString());  
}
```

View a list of your Amazon S3 on Outposts access points

Access points simplify managing data access at scale for shared datasets in Amazon S3. Access points are named network endpoints that are attached to buckets that you can use to perform Amazon S3 object operations, such as `GetObject` and `PutObject`. With S3 on Outposts, you must use access points to access any object in an Outposts bucket. Access points support only virtual-host-style addressing.

The following topics show you how to return a list of your S3 on Outposts access points by using the AWS Management Console, AWS Command Line Interface (AWS CLI), and AWS SDK for Java.

Using the S3 console

1. Open the Amazon S3 console at <https://console.aws.amazon.com/s3/>.
2. In the left navigation pane, choose **Outposts access points**.
3. Under **Outposts access points**, review your list of S3 on Outposts access points.

Using the AWS CLI

The following AWS CLI example lists the access points for an Outposts bucket. To run this command, replace the *user input placeholders* with your own information.

```
aws s3control list-access-points --account-id 123456789012 --bucket arn:aws:s3-outposts:region:123456789012:outpost/op-01ac5d28a6a232904/bucket/example-outpost-bucket
```

Using the AWS SDK for Java

The following SDK for Java example lists the access points for an Outposts bucket.

```
import com.amazonaws.services.s3control.model.*;  
  
public void listAccessPoints(String bucketArn) {  
  
    ListAccessPointsRequest reqListAPs = new ListAccessPointsRequest()  
        .withAccountId(AccountId)  
        .withBucket(bucketArn);  
  
    ListAccessPointsResult respListAPs = s3ControlClient.listAccessPoints(reqListAPs);  
    System.out.printf("ListAccessPoints Response: %s%n", respListAPs.toString());  
}
```

Deleting an access point

Access points simplify managing data access at scale for shared datasets in Amazon S3. Access points are named network endpoints that are attached to buckets that you can use to perform Amazon S3 object operations, such as `GetObject` and `PutObject`. With S3 on Outposts, you must use access points to access any object in an Outposts bucket. Access points support only virtual-host-style addressing.

The following examples show you how to delete an access point by using the AWS Management Console and the AWS Command Line Interface (AWS CLI).

Using the S3 console

1. Open the Amazon S3 console at <https://console.aws.amazon.com/s3/>.
2. In the left navigation pane, choose **Outposts access points**.
3. In the **Outposts access points** section, choose the Outposts access point that you want to delete.
4. Choose **Delete**.
5. Confirm the deletion.

Using the AWS CLI

The following AWS CLI example deletes an Outposts access point. To run this command, replace the *user input placeholders* with your own information.

```
aws s3control delete-access-point --account-id 123456789012 --name arn:aws:s3-  
outposts:region:123456789012:outpost/op-01ac5d28a6a232904/accesspoint/example-outposts-  
access-point
```

Adding or editing an access point policy

Access points have distinct permissions and network controls that Amazon S3 on Outposts applies for any request that is made through that access point. Each access point enforces a customized access point policy that works in conjunction with the bucket policy that is attached to the underlying bucket. For more information, see [Access points \(p. 1261\)](#).

The following topics show you how to add or edit the access point policy for your S3 on Outposts access point by using the AWS Management Console, AWS Command Line Interface (AWS CLI), and AWS SDK for Java.

Using the S3 console

1. Open the Amazon S3 console at <https://console.aws.amazon.com/s3/>.
2. In the left navigation pane, choose **Outposts buckets**.

3. Choose the Outposts bucket that you want to edit the access point policy for.
4. Choose the **Outposts access points** tab.
5. In the **Outposts access points** section, choose the access point whose policy you want to edit, and choose **Edit policy**.
6. Add or edit the policy in the **Outposts access point policy** section. For more information, see [Setting up IAM with S3 on Outposts \(p. 1249\)](#).

Using the AWS CLI

The following AWS CLI example puts a policy on an Outposts access point.

1. Save the following access point policy to a JSON file. In this example, the file is named `appolicy1.json`. Replace the *user input placeholders* with your own information.

```
{  
    "Version": "2012-10-17",  
    "Id": "exampleAccessPointPolicy",  
    "Statement": [  
        {  
            "Sid": "st1",  
            "Effect": "Allow",  
            "Principal": {  
                "AWS": "123456789012"  
            },  
            "Action": "s3-outposts:*",  
            "Resource": "arn:aws:s3-  
outposts:region:123456789012:outpost/op-01ac5d28a6a232904/accesspoint/example-outposts-  
access-point"  
        }  
    ]  
}
```

2. Submit the JSON file as part of the `put-access-point-policy` CLI command. Replace the *user input placeholders* with your own information.

```
aws s3control put-access-point-policy --account-id 123456789012 --name arn:aws:s3-  
outposts:region:123456789012:outpost/op-01ac5d28a6a232904/accesspoint/example-outposts-  
access-point --policy file://appolicy1.json
```

Using the AWS SDK for Java

The following SDK for Java example puts a policy on an Outposts access point.

```
import com.amazonaws.services.s3control.model.*;  
  
public void putAccessPointPolicy(String accessPointArn) {  
  
    String policy = "{\"Version\": \"2012-10-17\", \"Id\": \"testAccessPointPolicy\",  
    \"Statement\": [{\"Sid\": \"st1\", \"Effect\": \"Allow\", \"Principal\": {\"AWS\": \"\" + AccountId  
    + "\"}, \"Action\": \"s3-outposts:*\", \"Resource\": \"\" + accessPointArn + "\"}]}";  
  
    PutAccessPointPolicyRequest reqPutAccessPointPolicy = new PutAccessPointPolicyRequest()  
        .withAccountId(AccountId)  
        .withName(accessPointArn)  
        .withPolicy(policy);  
  
    PutAccessPointPolicyResult respPutAccessPointPolicy =  
    s3ControlClient.putAccessPointPolicy(reqPutAccessPointPolicy);
```

```
    System.out.printf("PutAccessPointPolicy Response: %s%n",
respPutAccessPointPolicy.toString());
    printWriter.printf("PutAccessPointPolicy Response: %s%n",
respPutAccessPointPolicy.toString());
}
```

Viewing an access point policy for an S3 on Outposts access point

Access points have distinct permissions and network controls that Amazon S3 on Outposts applies for any request that is made through that access point. Each access point enforces a customized access point policy that works in conjunction with the bucket policy that is attached to the underlying bucket. For more information, see [Access points \(p. 1261\)](#).

For more information about working with access points in S3 on Outposts, see [Working with S3 on Outposts buckets \(p. 1261\)](#).

The following topics show you how to view your S3 on Outposts access point policy by using the AWS Management Console, AWS Command Line Interface (AWS CLI), and AWS SDK for Java.

Using the S3 console

1. Open the Amazon S3 console at <https://console.aws.amazon.com/s3/>.
2. In the left navigation pane, choose **Outposts access points**.
3. Choose the Outposts access point that you want to view the policy for.
4. On the **Permissions** tab, review the S3 on Outposts access point policy.
5. To edit the access point policy, see [Adding or editing an access point policy \(p. 1280\)](#).

Using the AWS CLI

The following AWS CLI example gets a policy for an Outposts access point. To run this command, replace the `user input placeholders` with your own information.

```
aws s3control get-access-point-policy --account-id 123456789012 --name arn:aws:s3-
outposts:region:123456789012:outpost/op-01ac5d28a6a232904/accesspoint/example-outposts-
access-point
```

Using the AWS SDK for Java

The following SDK for Java example gets a policy for an Outposts access point.

```
import com.amazonaws.services.s3control.model.*;

public void getAccessPointPolicy(String accessPointArn) {

    GetAccessPointPolicyRequest reqGetAccessPointPolicy = new GetAccessPointPolicyRequest()
        .withAccountId(AccountId)
        .withName(accessPointArn);

    GetAccessPointPolicyResult respGetAccessPointPolicy =
s3ControlClient.getAccessPointPolicy(reqGetAccessPointPolicy);
    System.out.printf("GetAccessPointPolicy Response: %s%n",
respGetAccessPointPolicy.toString());
    printWriter.printf("GetAccessPointPolicy Response: %s%n",
respGetAccessPointPolicy.toString());
```

}

Working with Amazon S3 on Outposts endpoints

To route requests to an Amazon S3 on Outposts access point, you must create and configure an S3 on Outposts endpoint. Each virtual private cloud (VPC) on your Outpost can have one associated endpoint. For more information about endpoint quotas, see [S3 on Outposts network requirements \(p. 1248\)](#). You must create these endpoints to be able to access your Outposts buckets and perform object operations. For more information, see [Endpoints \(p. 1262\)](#).

For more information about working with buckets on S3 on Outposts, see [Working with S3 on Outposts buckets \(p. 1261\)](#).

The following sections describe how to create and manage endpoints for S3 on Outposts.

Topics

- [Creating an endpoint on an Outpost \(p. 1283\)](#)
- [Viewing a list of your Amazon S3 on Outposts endpoints \(p. 1284\)](#)
- [Deleting an Amazon S3 on Outposts endpoint \(p. 1285\)](#)

Creating an endpoint on an Outpost

To route requests to an Amazon S3 on Outposts access point, you must create and configure an S3 on Outposts endpoint. Each virtual private cloud (VPC) on your Outpost can have one associated endpoint. For more information about endpoint quotas, see [S3 on Outposts network requirements \(p. 1248\)](#). You must create these endpoints to be able to access your Outposts buckets and perform object operations. For more information, see [Endpoints \(p. 1262\)](#).

Permissions

For more information about the permissions that are required to create an endpoint, see [Permissions for S3 on Outposts endpoints \(p. 1252\)](#).

The following examples show you how to create an S3 on Outposts endpoint by using the AWS Management Console, AWS Command Line Interface (AWS CLI), and AWS SDK for Java.

Using the S3 console

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 **Outposts access points**.
3. Choose the **Outposts endpoints** tab.
4. Choose **Create Outposts endpoint**.
5. Under **Outpost**, choose the Outpost to create this endpoint on.
6. Under **VPC**, choose a VPC that does not yet have an endpoint and that also complies with the rules for Outposts endpoints.

A virtual private cloud (VPC) enables you to launch AWS resources into a virtual network that you define. This virtual network closely resembles a traditional network that you would operate in your own data center, with the benefits of using the scalable infrastructure of AWS.

If you don't have a VPC, choose **Create VPC**. For more information, see [Creating access points restricted to a virtual private cloud \(p. 308\)](#).

7. Choose **Create Outposts endpoint**.

Using the AWS CLI

Example

The following AWS CLI example creates an endpoint for an Outpost by using the VPC resource access type. The VPC is derived from the subnet. To run this command, replace the *user input placeholders* with your own information.

```
aws s3outposts create-endpoint --outpost-id op-01ac5d28a6a232904 --subnet-id
    subnet-8c7a57c5 --security-group-id sg-ab19e0d1
```

The following AWS CLI example creates an endpoint for an Outpost by using the customer-owned IP address pool (CoIP pool) access type. To run this command, replace the *user input placeholders* with your own information.

```
aws s3outposts create-endpoint --outpost-id op-01ac5d28a6a232904 --subnet-id
    subnet-8c7a57c5 --security-group-id sg-ab19e0d1 --access-type CustomerOwnedIp --customer-
    owned-ipv4-pool ipv4pool-coip-12345678901234567
```

Using the AWS SDK for Java

Example

The following SDK for Java example creates an endpoint for an Outpost. To use this example, replace the *user input placeholders* with your own information.

```
import com.amazonaws.services.s3outposts.AmazonS3Outposts;
import com.amazonaws.services.s3outposts.AmazonS3OutpostsClientBuilder;
import com.amazonaws.services.s3outposts.model.CreateEndpointRequest;
import com.amazonaws.services.s3outposts.model.CreateEndpointResult;

public void createEndpoint() {
    AmazonS3Outposts s3OutpostsClient = AmazonS3OutpostsClientBuilder
        .standard().build();

    CreateEndpointRequest createEndpointRequest = new CreateEndpointRequest()
        .withOutpostId("op-0d79779cef3c30a40")
        .withSubnetId("subnet-8c7a57c5")
        .withSecurityGroupId("sg-ab19e0d1")
        .withAccessType("CustomerOwnedIp")
        .withCustomerOwnedIpv4Pool("ipv4pool-coip-12345678901234567");
    // Use .withAccessType and .withCustomerOwnedIpv4Pool only when the access type is
    // customer-owned IP address pool (CoIP pool)
    CreateEndpointResult createEndpointResult =
        s3OutpostsClient.createEndpoint(createEndpointRequest);
    System.out.println("Endpoint is created and its ARN is " +
        createEndpointResult.getEndpointArn());
}
```

Viewing a list of your Amazon S3 on Outposts endpoints

To route requests to an Amazon S3 on Outposts access point, you must create and configure an S3 on Outposts endpoint. Each virtual private cloud (VPC) on your Outpost can have one associated endpoint. For more information about endpoint quotas, see [S3 on Outposts network requirements \(p. 1248\)](#). You must create these endpoints to be able to access your Outposts buckets and perform object operations. For more information, see [Endpoints \(p. 1262\)](#).

The following examples show you how to return a list of your S3 on Outposts endpoints by using the AWS Management Console, AWS Command Line Interface (AWS CLI), and AWS SDK for Java.

Using the S3 console

1. Open the Amazon S3 console at <https://console.aws.amazon.com/s3/>.
2. In the left navigation pane, choose **Outposts access points**.
3. On the **Outposts access points** page, choose the **Outposts endpoints** tab.
4. Under **Outposts endpoints**, you can view a list of your S3 on Outposts endpoints.

Using the AWS CLI

The following AWS CLI example lists the endpoints for the AWS Outposts resources that are associated with your account. For more information about this command, see [list-endpoints](#) in the *AWS CLI Reference*.

```
aws s3outposts list-endpoints
```

Using the AWS SDK for Java

The following SDK for Java example lists the endpoints for an Outpost. For more information, see [ListEndpoints](#) in the *Amazon Simple Storage Service API Reference*.

```
import com.amazonaws.services.s3outposts.AmazonS3Outposts;
import com.amazonaws.services.s3outposts.AmazonS3OutpostsClientBuilder;
import com.amazonaws.services.s3outposts.model.ListEndpointsRequest;
import com.amazonaws.services.s3outposts.model.ListEndpointsResult;

public void listEndpoints() {
    AmazonS3Outposts s3OutpostsClient = AmazonS3OutpostsClientBuilder
        .standard().build();

    ListEndpointsRequest listEndpointsRequest = new ListEndpointsRequest();
    ListEndpointsResult listEndpointsResult =
        s3OutpostsClient.listEndpoints(listEndpointsRequest);
    System.out.println("List endpoints result is " + listEndpointsResult);
}
```

Deleting an Amazon S3 on Outposts endpoint

To route requests to an Amazon S3 on Outposts access point, you must create and configure an S3 on Outposts endpoint. Each virtual private cloud (VPC) on your Outpost can have one associated endpoint. For more information about endpoint quotas, see [S3 on Outposts network requirements \(p. 1248\)](#). You must create these endpoints to be able to access your Outposts buckets and perform object operations. For more information, see [Endpoints \(p. 1262\)](#).

The following examples show you how to delete your S3 on Outposts endpoints by using the AWS Management Console, AWS Command Line Interface (AWS CLI), and AWS SDK for Java.

Using the S3 console

1. Open the Amazon S3 console at <https://console.aws.amazon.com/s3/>.
2. In the left navigation pane, choose **Outposts access points**.
3. On the **Outposts access points** page, choose the **Outposts endpoints** tab.
4. Under **Outposts endpoints**, choose the endpoint that you want to delete, and choose **Delete**.

Using the AWS CLI

The following AWS CLI example deletes an endpoint for an Outpost. To run this command, replace the *user input placeholders* with your own information.

```
aws s3outposts delete-endpoint --endpoint-id example-endpoint-id --outpost-id op-01ac5d28a6a232904
```

Using the AWS SDK for Java

The following SDK for Java example deletes an endpoint for an Outpost. To use this example, replace the *user input placeholders* with your own information.

```
import com.amazonaws.Arn;
import com.amazonaws.services.s3outposts.AmazonS3Outposts;
import com.amazonaws.services.s3outposts.AmazonS3OutpostsClientBuilder;
import com.amazonaws.services.s3outposts.model.DeleteEndpointRequest;

public void deleteEndpoint(String endpointArnInput) {
    String outpostId = "op-01ac5d28a6a232904";
    AmazonS3Outposts s3OutpostsClient = AmazonS3OutpostsClientBuilder
        .standard().build();

    Arn endpointArn = Arn.fromString(endpointArnInput);
    String[] resourceParts = endpointArn.getResource().getResource().split("/");
    String endpointId = resourceParts[resourceParts.length - 1];
    DeleteEndpointRequest deleteEndpointRequest = new DeleteEndpointRequest()
        .withEndpointId(endpointId)
        .withOutpostId(outpostId);
    s3OutpostsClient.deleteEndpoint(deleteEndpointRequest);
    System.out.println("Endpoint with id " + endpointId + " is deleted.");
}
```

Working with S3 on Outposts objects

With Amazon S3 on Outposts, you can create S3 buckets on your AWS Outposts and easily store and retrieve objects on premises for applications that require local data access, local data processing, and data residency. S3 on Outposts provides a new storage class, S3 Outposts (OUTPOSTS), which uses the Amazon S3 APIs, and is designed to store data durably and redundantly across multiple devices and servers on your AWS Outposts. You communicate with your Outpost bucket by using an access point and endpoint connection over a virtual private cloud (VPC). You can use the same APIs and features on Outpost buckets as you do on Amazon S3 buckets, including access policies, encryption, and tagging. You can use S3 on Outposts through the AWS Management Console, AWS Command Line Interface (AWS CLI), AWS SDKs, or REST API.

Objects are the fundamental entities stored in Amazon S3 on Outposts. Every object is contained in a bucket. You must use access points to access any object in an Outpost bucket. When you specify the bucket for object operations, you use the access point Amazon Resource Name (ARN), which includes the AWS Region code for the Region that the Outpost is homed to, the AWS account ID, the Outpost ID, and the access point name. The following example shows the ARN format for S3 on Outposts access points in object operations:

```
arn:aws:s3-outposts:region:account-id:outpost/outpost-id/accesspoint/accesspoint-name
```

For more information about S3 on Outposts ARNs, see [Resource ARNs for S3 on Outposts \(p. 1250\)](#).

Object ARNs use the following format, which includes the AWS Region that the Outpost is homed to, AWS account ID, Outpost ID, bucket name, and object key:

```
arn:aws:s3-outposts:us-west-2:123456789012:outpost/op-01ac5d28a6a232904/bucket/DOC-EXAMPLE-  
BUCKET1/object/myobject
```

With Amazon S3 on Outposts, object data is always stored on the Outpost. When AWS installs an Outpost rack, your data stays local to your Outpost to meet data-residency requirements. Your objects never leave your Outpost and are not in an AWS Region. Because the AWS Management Console is hosted in-Region, you can't use the console to upload or manage objects in your Outpost. However, you can use the REST API, AWS Command Line Interface (AWS CLI), and AWS SDKs to upload and manage your objects through your access points.

To upload an object, see [Step 4: Upload an object to an S3 on Outposts bucket \(p. 1258\)](#). For other object actions, see the following topics.

Topics

- [Copying an object in an Amazon S3 on Outposts bucket using the AWS SDK for Java \(p. 1287\)](#)
- [Getting an object from an Amazon S3 on Outposts bucket \(p. 1288\)](#)
- [Listing the objects in an Amazon S3 on Outposts bucket \(p. 1290\)](#)
- [Deleting objects in Amazon S3 on Outposts buckets \(p. 1292\)](#)
- [Using HeadBucket to determine if an S3 on Outposts bucket exists and you have access permissions \(p. 1295\)](#)
- [Performing and managing a multipart upload with the SDK for Java \(p. 1296\)](#)
- [Using presigned URLs for S3 on Outposts \(p. 1301\)](#)

Copying an object in an Amazon S3 on Outposts bucket using the AWS SDK for Java

Objects are the fundamental entities stored in Amazon S3 on Outposts. Every object is contained in a bucket. You must use access points to access any object in an Outpost bucket. When you specify the bucket for object operations, you use the access point Amazon Resource Name (ARN), which includes the AWS Region code for the Region that the Outpost is homed to, the AWS account ID, the Outpost ID, and the access point name. The following example shows the ARN format for S3 on Outposts access points in object operations:

```
arn:aws:s3-outposts:region:account-id:outpost/outpost-id/accesspoint/accesspoint-name
```

For more information about S3 on Outposts ARNs, see [Resource ARNs for S3 on Outposts \(p. 1250\)](#).

With Amazon S3 on Outposts, object data is always stored on the Outpost. When AWS installs an Outpost rack, your data stays local to your Outpost to meet data-residency requirements. Your objects never leave your Outpost and are not in an AWS Region. Because the AWS Management Console is hosted in-Region, you can't use the console to upload or manage objects in your Outpost. However, you can use the REST API, AWS Command Line Interface (AWS CLI), and AWS SDKs to upload and manage your objects through your access points.

The following example shows you how to copy an object in an S3 on Outposts bucket by using the AWS SDK for Java.

Using the AWS SDK for Java

The following S3 on Outposts example copies an object into a new object in the same bucket by using the SDK for Java. To use this example, replace the `user input placeholders` with your own information.

```
import com.amazonaws.AmazonServiceException;
import com.amazonaws.SdkClientException;
import com.amazonaws.services.s3.AmazonS3;
import com.amazonaws.services.s3.AmazonS3ClientBuilder;
import com.amazonaws.services.s3.model.CopyObjectRequest;

public class CopyObject {
    public static void main(String[] args) {
        String accessPointArn = "*** access point ARN ***";
        String sourceKey = "*** Source object key ***";
        String destinationKey = "*** Destination object key ***";

        try {
            // This code expects that you have AWS credentials set up per:
            // https://docs.aws.amazon.com/sdk-for-java/v1/developer-guide/setup-credentials.html
            AmazonS3 s3Client = AmazonS3ClientBuilder.standard()
                .enableUseArnRegion()
                .build();

            // Copy the object into a new object in the same bucket.
            CopyObjectRequest copyObjectRequest = new CopyObjectRequest(accessPointArn,
                sourceKey, accessPointArn, destinationKey);
            s3Client.copyObject(copyObjectRequest);
        } catch (AmazonServiceException e) {
            // The call was transmitted successfully, but Amazon S3 couldn't process
            // it, so it returned an error response.
            e.printStackTrace();
        } catch (SdkClientException e) {
            // Amazon S3 couldn't be contacted for a response, or the client
            // couldn't parse the response from Amazon S3.
            e.printStackTrace();
        }
    }
}
```

Getting an object from an Amazon S3 on Outposts bucket

Objects are the fundamental entities stored in Amazon S3 on Outposts. Every object is contained in a bucket. You must use access points to access any object in an Outpost bucket. When you specify the bucket for object operations, you use the access point Amazon Resource Name (ARN), which includes the AWS Region code for the Region that the Outpost is homed to, the AWS account ID, the Outpost ID, and the access point name. The following example shows the ARN format for S3 on Outposts access points in object operations:

```
arn:aws:s3-outposts:region:account-id:outpost/outpost-id/accesspoint/accesspoint-name
```

For more information about S3 on Outposts ARNs, see [Resource ARNs for S3 on Outposts \(p. 1250\)](#).

With Amazon S3 on Outposts, object data is always stored on the Outpost. When AWS installs an Outpost rack, your data stays local to your Outpost to meet data-residency requirements. Your objects never leave your Outpost and are not in an AWS Region. Because the AWS Management Console is hosted in-Region, you can't use the console to upload or manage objects in your Outpost. However, you can use the REST API, AWS Command Line Interface (AWS CLI), and AWS SDKs to upload and manage your objects through your access points.

The following examples show you how to download (get) an object by using the AWS Command Line Interface (AWS CLI) and AWS SDK for Java.

Using the AWS CLI

The following example gets an object named `sample-object.xml` from an S3 on Outposts bucket (`s3-outposts:GetObject`) by using the AWS CLI. To use this command, replace each *user input placeholder* with your own information. For more information about this command, see [get-object](#) in the [AWS CLI Reference](#).

```
aws s3api get-object --bucket arn:aws:s3-
outposts:region:123456789012:outpost/op-01ac5d28a6a232904/accesspoint/example-outposts-
access-point --key testkey sample-object.xml
```

Using the AWS SDK for Java

The following S3 on Outposts example gets an object by using the SDK for Java. To use this example, replace each *user input placeholder* with your own information. For more information, see [GetObject](#) in the [Amazon Simple Storage Service API Reference](#).

```
import com.amazonaws.AmazonServiceException;
import com.amazonaws.SdkClientException;
import com.amazonaws.services.s3.AmazonS3;
import com.amazonaws.services.s3.AmazonS3ClientBuilder;
import com.amazonaws.services.s3.model.GetObjectRequest;
import com.amazonaws.services.s3.model.ResponseHeaderOverrides;
import com.amazonaws.services.s3.model.S3Object;

import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStream;
import java.io.InputStreamReader;

public class GetObject {
    public static void main(String[] args) throws IOException {
        String accessPointArn = "*** access point ARN ***";
        String key = "*** Object key ***";

        S3Object fullObject = null, objectPortion = null, headerOverrideObject = null;
        try {
            // This code expects that you have AWS credentials set up per:
            // https://docs.aws.amazon.com/sdk-for-java/v1/developer-guide/setup-
credentials.html
            AmazonS3 s3Client = AmazonS3ClientBuilder.standard()
                .enableUseArnRegion()
                .build();

            // Get an object and print its contents.
            System.out.println("Downloading an object");
            fullObject = s3Client.getObject(new GetObjectRequest(accessPointArn, key));
            System.out.println("Content-Type: " +
                fullObject.getObjectMetadata().getContentType());
            System.out.println("Content: ");
            displayTextInputStream(fullObject.getObjectContent());

            // Get a range of bytes from an object and print the bytes.
            GetObjectRequest rangeObjectRequest = new GetObjectRequest(accessPointArn, key)
                .withRange(0, 9);
            objectPortion = s3Client.getObject(rangeObjectRequest);
            System.out.println("Printing bytes retrieved.");
            displayTextInputStream(objectPortion.getObjectContent());

            // Get an entire object, overriding the specified response headers, and print
            // the object's content.
            ResponseHeaderOverrides headerOverrides = new ResponseHeaderOverrides()
                .withCacheControl("No-cache")
        }
    }

    private void displayTextInputStream(InputStream inputStream) {
        BufferedReader reader = new BufferedReader(new InputStreamReader(inputStream));
        String line;
        while ((line = reader.readLine()) != null) {
            System.out.println(line);
        }
    }
}
```

```
.withContentDisposition("attachment; filename=example.txt");
GetObjectRequest getObjectRequestHeaderOverride = new
GetObjectRequest(accessPointArn, key)
    .withResponseHeaders(headerOverrides);
headerOverrideObject = s3Client.getObject(getObjectRequestHeaderOverride);
displayTextInputStream(headerOverrideObject.getObjectContent());
} catch (AmazonServiceException e) {
    // The call was transmitted successfully, but Amazon S3 couldn't process
    // it, so it returned an error response.
    e.printStackTrace();
} catch (SdkClientException e) {
    // Amazon S3 couldn't be contacted for a response, or the client
    // couldn't parse the response from Amazon S3.
    e.printStackTrace();
} finally {
    // To ensure that the network connection doesn't remain open, close any open
input streams.
    if (fullObject != null) {
        fullObject.close();
    }
    if (objectPortion != null) {
        objectPortion.close();
    }
    if (headerOverrideObject != null) {
        headerOverrideObject.close();
    }
}
}

private static void displayTextInputStream(InputStream input) throws IOException {
    // Read the text input stream one line at a time and display each line.
    BufferedReader reader = new BufferedReader(new InputStreamReader(input));
    String line = null;
    while ((line = reader.readLine()) != null) {
        System.out.println(line);
    }
    System.out.println();
}
}
```

Listing the objects in an Amazon S3 on Outposts bucket

Objects are the fundamental entities stored in Amazon S3 on Outposts. Every object is contained in a bucket. You must use access points to access any object in an Outpost bucket. When you specify the bucket for object operations, you use the access point Amazon Resource Name (ARN), which includes the AWS Region code for the Region that the Outpost is homed to, the AWS account ID, the Outpost ID, and the access point name. The following example shows the ARN format for S3 on Outposts access points in object operations:

```
arn:aws:s3-outposts:region:account-id:outpost/outpost-id/accesspoint/accesspoint-name
```

For more information about S3 on Outposts ARNs, see [Resource ARNs for S3 on Outposts \(p. 1250\)](#).

Note

With Amazon S3 on Outposts, object data is always stored on the Outpost. When AWS installs an Outpost rack, your data stays local to your Outpost to meet data-residency requirements. Your objects never leave your Outpost and are not in an AWS Region. Because the AWS Management Console is hosted in-Region, you can't use the console to upload or manage objects in your Outpost. However, you can use the REST API, AWS Command Line Interface (AWS CLI), and AWS SDKs to upload and manage your objects through your access points.

The following examples show you how to list the objects in an S3 on Outposts bucket using the AWS CLI and AWS SDK for Java.

Using the AWS CLI

The following example lists the objects in an S3 on Outposts bucket (`s3-outposts>ListObjectsV2`) by using the AWS CLI. To use this command, replace each *user input placeholder* with your own information. For more information about this command, see [list-objects-v2](#) in the *AWS CLI Reference*.

```
aws s3api list-objects-v2 --bucket arn:aws:s3-
outposts:region:123456789012:outpost/op-01ac5d28a6a232904/accesspoint/example-outposts-
access-point
```

Note

When using this action with Amazon S3 on Outposts through the AWS SDKs, you provide the Outposts access point ARN in place of the bucket name, in the following form: `arn:aws:s3-outposts:region:123456789012:outpost/op-01ac5d28a6a232904/accesspoint/example-Outposts-Access-Point`. For more information about S3 on Outposts ARNs, see [Resource ARNs for S3 on Outposts \(p. 1250\)](#).

Using the AWS SDK for Java

The following S3 on Outposts example lists objects in a bucket by using the SDK for Java. To use this example, replace each *user input placeholder* with your own information.

Important

This example uses [ListObjectsV2](#), which is the latest revision of the `ListObjects` API operation. We recommend that you use this revised API operation for application development. For backward compatibility, Amazon S3 continues to support the prior version of this API operation.

```
import com.amazonaws.AmazonServiceException;
import com.amazonaws.SdkClientException;
import com.amazonaws.services.s3.AmazonS3;
import com.amazonaws.services.s3.AmazonS3ClientBuilder;
import com.amazonaws.services.s3.model.ListObjectsV2Request;
import com.amazonaws.services.s3.model.ListObjectsV2Result;
import com.amazonaws.services.s3.model.S3ObjectSummary;

public class ListObjectsV2 {

    public static void main(String[] args) {
        String accessPointArn = "*** access point ARN ***";

        try {
            // This code expects that you have AWS credentials set up per:
            // https://docs.aws.amazon.com/sdk-for-java/v1/developer-guide/setup-
            credentials.html
            AmazonS3 s3Client = AmazonS3ClientBuilder.standard()
                .enableUseArnRegion()
                .build();

            System.out.println("Listing objects");

            // maxKeys is set to 2 to demonstrate the use of
            // ListObjectsV2Result.getNextContinuationToken()
            ListObjectsV2Request req = new
            ListObjectsV2Request().withBucketName(accessPointArn).withMaxKeys(2);
            ListObjectsV2Result result;

            do {
                result = s3Client.listObjectsV2(req);
```

```
        for (S3ObjectSummary objectSummary : result.getObjectSummaries()) {
            System.out.printf(" - %s (size: %d)\n", objectSummary.getKey(),
                objectSummary.getSize());
        }
        // If there are more than maxKeys keys in the bucket, get a continuation
        token
        // and list the next objects.
        String token = result.getNextContinuationToken();
        System.out.println("Next Continuation Token: " + token);
        req.setContinuationToken(token);
    } while (result.isTruncated());
} catch (AmazonServiceException e) {
    // The call was transmitted successfully, but Amazon S3 couldn't process
    // it, so it returned an error response.
    e.printStackTrace();
} catch (SdkClientException e) {
    // Amazon S3 couldn't be contacted for a response, or the client
    // couldn't parse the response from Amazon S3.
    e.printStackTrace();
}
}
```

Deleting objects in Amazon S3 on Outposts buckets

Objects are the fundamental entities stored in Amazon S3 on Outposts. Every object is contained in a bucket. You must use access points to access any object in an Outpost bucket. When you specify the bucket for object operations, you use the access point Amazon Resource Name (ARN), which includes the AWS Region code for the Region that the Outpost is homed to, the AWS account ID, the Outpost ID, and the access point name. The following example shows the ARN format for S3 on Outposts access points in object operations:

```
arn:aws:s3-outposts:region:account-id:outpost/outpost-id/accesspoint/accesspoint-name
```

For more information about S3 on Outposts ARNs, see [Resource ARNs for S3 on Outposts \(p. 1250\)](#).

With Amazon S3 on Outposts, object data is always stored on the Outpost. When AWS installs an Outpost rack, your data stays local to your Outpost to meet data-residency requirements. Your objects never leave your Outpost and are not in an AWS Region. Because the AWS Management Console is hosted in-Region, you can't use the console to upload or manage objects in your Outpost. However, you can use the REST API, AWS Command Line Interface (AWS CLI), and AWS SDKs to upload and manage your objects through your access points.

The following examples show you how to delete a single object or multiple objects in an S3 on Outposts bucket by using the AWS Command Line Interface (AWS CLI) and AWS SDK for Java.

Using the AWS CLI

The following examples show you how to delete a single object or multiple objects from an S3 on Outposts bucket.

delete-object

The following example deletes an object named `sample-object.xml` from an S3 on Outposts bucket (`s3-outposts:DeleteObject`) by using the AWS CLI. To use this command, replace each `user input placeholder` with your own information. For more information about this command, see `delete-object` in the [AWS CLI Reference](#).

```
aws s3api delete-object --bucket arn:aws:s3-
outposts:region:123456789012:outpost/op-01ac5d28a6a232904/accesspoint/example-outposts-
access-point --key sample-object.xml
```

delete-objects

The following example deletes two objects named `sample-object.xml` and `test1.txt` from an S3 on Outposts bucket (`s3-outposts:DeleteObject`) by using the AWS CLI. To use this command, replace each *user input placeholder* with your own information. For more information about this command, see [delete-objects](#) in the *AWS CLI Reference*.

```
aws s3api delete-objects --bucket arn:aws:s3-
outposts:region:123456789012:outpost/op-01ac5d28a6a232904/accesspoint/example-outposts-
access-point --delete file://delete.json

delete.json
{
    "Objects": [
        {
            "Key": "test1.txt"
        },
        {
            "Key": "sample-object.xml"
        }
    ],
    "Quiet": false
}
```

Using the AWS SDK for Java

The following examples show you how to delete a single object or multiple objects from an S3 on Outposts bucket.

DeleteObject

The following S3 on Outposts example deletes an object in a bucket by using the SDK for Java. To use this example, specify the access point ARN for the Outpost and the key name for the object that you want to delete. For more information, see [DeleteObject](#) in the *Amazon Simple Storage Service API Reference*.

```
import com.amazonaws.AmazonServiceException;
import com.amazonaws.SdkClientException;
import com.amazonaws.services.s3.AmazonS3;
import com.amazonaws.services.s3.AmazonS3ClientBuilder;
import com.amazonaws.services.s3.model.DeleteObjectRequest;

public class DeleteObject {
    public static void main(String[] args) {
        String accessPointArn = "*** access point ARN ***";
        String keyName = "*** key name ***";

        try {
            // This code expects that you have AWS credentials set up per:
            // https://docs.aws.amazon.com/sdk-for-java/v1/developer-guide/setup-
credentials.html
            AmazonS3 s3Client = AmazonS3ClientBuilder.standard()
                .enableUseArnRegion()
                .build();

            s3Client.deleteObject(new DeleteObjectRequest(accessPointArn, keyName));
        } catch (AmazonServiceException e) {
```

```
// The call was transmitted successfully, but Amazon S3 couldn't process
// it, so it returned an error response.
e.printStackTrace();
} catch (SdkClientException e) {
    // Amazon S3 couldn't be contacted for a response, or the client
    // couldn't parse the response from Amazon S3.
    e.printStackTrace();
}
}
```

DeleteObjects

The following S3 on Outposts example uploads and then deletes objects in a bucket by using the SDK for Java. To use this example, specify the access point ARN for the Outpost. For more information, see [DeleteObjects](#) in the *Amazon Simple Storage Service API Reference*.

```
import com.amazonaws.AmazonServiceException;
import com.amazonaws.SdkClientException;
import com.amazonaws.services.s3.AmazonS3;
import com.amazonaws.services.s3.AmazonS3ClientBuilder;
import com.amazonaws.services.s3.model.DeleteObjectsRequest;
import com.amazonaws.services.s3.model.DeleteObjectsRequest.KeyVersion;
import com.amazonaws.services.s3.model.DeleteObjectsResult;

import java.util.ArrayList;

public class DeleteObjects {

    public static void main(String[] args) {
        String accessPointArn = "arn:aws:s3-
outposts:region:123456789012:outpost/op-01ac5d28a6a232904/accesspoint/example-outposts-
access-point";

        try {
            // This code expects that you have AWS credentials set up per:
            // https://docs.aws.amazon.com/sdk-for-java/v1/developer-guide/setup-
credentials.html
            AmazonS3 s3Client = AmazonS3ClientBuilder.standard()
                .enableUseArnRegion()
                .build();

            // Upload three sample objects.
            ArrayList<KeyVersion> keys = new ArrayList<KeyVersion>();
            for (int i = 0; i < 3; i++) {
                String keyName = "delete object example " + i;
                s3Client.putObject(accessPointArn, keyName, "Object number " + i + " to
be deleted.");
                keys.add(new KeyVersion(keyName));
            }
            System.out.println(keys.size() + " objects successfully created.");

            // Delete the sample objects.
            DeleteObjectsRequest multiObjectDeleteRequest = new
DeleteObjectsRequest(accessPointArn)
                .withKeys(keys)
                .withQuiet(false);

            // Verify that the objects were deleted successfully.
            DeleteObjectsResult delObjRes =
s3Client.deleteObjects(multiObjectDeleteRequest);
            int successfulDeletes = delObjRes.getDeletedObjects().size();
            System.out.println(successfulDeletes + " objects successfully deleted.");
        }
    }
}
```

```
        } catch (AmazonServiceException e) {
            // The call was transmitted successfully, but Amazon S3 couldn't process
            // it, so it returned an error response.
            e.printStackTrace();
        } catch (SdkClientException e) {
            // Amazon S3 couldn't be contacted for a response, or the client
            // couldn't parse the response from Amazon S3.
            e.printStackTrace();
        }
    }
}
```

Using HeadBucket to determine if an S3 on Outposts bucket exists and you have access permissions

Objects are the fundamental entities stored in Amazon S3 on Outposts. Every object is contained in a bucket. You must use access points to access any object in an Outpost bucket. When you specify the bucket for object operations, you use the access point Amazon Resource Name (ARN), which includes the AWS Region code for the Region that the Outpost is homed to, the AWS account ID, the Outpost ID, and the access point name. The following example shows the ARN format for S3 on Outposts access points in object operations:

```
arn:aws:s3-outposts:region:account-id:outpost/outpost-id/accesspoint/accesspoint-name
```

For more information about S3 on Outposts ARNs, see [Resource ARNs for S3 on Outposts \(p. 1250\)](#).

Note

With Amazon S3 on Outposts, object data is always stored on the Outpost. When AWS installs an Outpost rack, your data stays local to your Outpost to meet data-residency requirements. Your objects never leave your Outpost and are not in an AWS Region. Because the AWS Management Console is hosted in-Region, you can't use the console to upload or manage objects in your Outpost. However, you can use the REST API, AWS Command Line Interface (AWS CLI), and AWS SDKs to upload and manage your objects through your access points.

The following AWS Command Line Interface (AWS CLI) and AWS SDK for Java examples show you how to use the HeadBucket API operation to determine if an Amazon S3 on Outposts bucket exists and whether you have permission to access it. For more information, see [HeadBucket](#) in the *Amazon Simple Storage Service API Reference*.

Using the AWS CLI

The following S3 on Outposts AWS CLI example uses the head-bucket command to determine if a bucket exists and you have permissions to access it. To use this command, replace each **user input placeholder** with your own information. For more information about this command, see [head-bucket](#) in the *AWS CLI Reference*.

```
aws s3api head-bucket --bucket arn:aws:s3-
outposts:region:123456789012:outpost/op-01ac5d28a6a232904/accesspoint/example-outposts-
access-point
```

Using the AWS SDK for Java

The following S3 on Outposts example shows how to determine if a bucket exists and if you have permission to access it. To use this example, specify the access point ARN for the Outpost. For more information, see [HeadBucket](#) in the *Amazon Simple Storage Service API Reference*.

```
import com.amazonaws.AmazonServiceException;
```

```
import com.amazonaws.SdkClientException;
import com.amazonaws.services.s3.AmazonS3;
import com.amazonaws.services.s3.AmazonS3ClientBuilder;
import com.amazonaws.services.s3.model.HeadBucketRequest;

public class HeadBucket {
    public static void main(String[] args) {
        String accessPointArn = "*** access point ARN ***";

        try {
            // This code expects that you have AWS credentials set up per:
            // https://docs.aws.amazon.com/sdk-for-java/v1/developer-guide/setup-
            credentials.html
            AmazonS3 s3Client = AmazonS3ClientBuilder.standard()
                .enableUseArnRegion()
                .build();

            s3Client.headBucket(new HeadBucketRequest(accessPointArn));
        } catch (AmazonServiceException e) {
            // The call was transmitted successfully, but Amazon S3 couldn't process
            // it, so it returned an error response.
            e.printStackTrace();
        } catch (SdkClientException e) {
            // Amazon S3 couldn't be contacted for a response, or the client
            // couldn't parse the response from Amazon S3.
            e.printStackTrace();
        }
    }
}
```

Performing and managing a multipart upload with the SDK for Java

With Amazon S3 on Outposts, you can create S3 buckets on your AWS Outposts resources and store and retrieve objects on-premises for applications that require local data access, local data processing, and data residency. You can use S3 on Outposts through the AWS Management Console, AWS Command Line Interface (AWS CLI), AWS SDKs, or REST API. For more information, see [What is Amazon S3 on Outposts? \(p. 1241\)](#)

The following examples show how you can use S3 on Outposts with the AWS SDK for Java to perform and manage a multipart upload.

Topics

- [Perform a multipart upload of an object in an S3 on Outposts bucket \(p. 1296\)](#)
- [Copy a large object in an S3 on Outposts bucket by using multipart upload \(p. 1298\)](#)
- [List parts of an object in an S3 on Outposts bucket \(p. 1299\)](#)
- [Retrieve a list of in-progress multipart uploads in an S3 on Outposts bucket \(p. 1300\)](#)

Perform a multipart upload of an object in an S3 on Outposts bucket

The following S3 on Outposts example initiates, uploads, and finishes a multipart upload of an object to a bucket by using the SDK for Java. To use this example, replace each *user input placeholder* with your own information. For more information, see [Uploading an object using multipart upload \(p. 174\)](#).

```
import com.amazonaws.AmazonServiceException;
```

```

import com.amazonaws.SdkClientException;
import com.amazonaws.services.s3.AmazonS3;
import com.amazonaws.services.s3.AmazonS3ClientBuilder;
import com.amazonaws.services.s3.model.*;

import java.util.ArrayList;
import java.util.List;

public class MultipartUploadCopy {
    public static void main(String[] args) {
        String accessPointArn = "*** Source access point ARN ***";
        String sourceObjectKey = "*** Source object key ***";
        String destObjectKey = "*** Target object key ***";

        try {
            // This code expects that you have AWS credentials set up per:
            // https://docs.aws.amazon.com/sdk-for-java/v1/developer-guide/setup-
credentials.html
            AmazonS3 s3Client = AmazonS3ClientBuilder.standard()
                .enableUseArnRegion()
                .build();

            // Initiate the multipart upload.
            InitiateMultipartUploadRequest initRequest = new
InitiateMultipartUploadRequest(accessPointArn, destObjectKey);
            InitiateMultipartUploadResult initResult =
s3Client.initiateMultipartUpload(initRequest);

            // Get the object size to track the end of the copy operation.
            GetObjectMetadataRequest metadataRequest = new
GetObjectMetadataRequest(accessPointArn, sourceObjectKey);
            ObjectMetadata metadataResult = s3Client.getObjectMetadata(metadataRequest);
            long objectSize = metadataResult.getContentLength();

            // Copy the object using 5 MB parts.
            long partSize = 5 * 1024 * 1024;
            long bytePosition = 0;
            int partNum = 1;
            List<CopyPartResult> copyResponses = new ArrayList<CopyPartResult>();
            while (bytePosition < objectSize) {
                // The last part might be smaller than partSize, so check to make sure
                // that lastByte isn't beyond the end of the object.
                long lastByte = Math.min(bytePosition + partSize - 1, objectSize - 1);

                // Copy this part.
                CopyPartRequest copyRequest = new CopyPartRequest()
                    .withSourceBucketName(accessPointArn)
                    .withSourceKey(sourceObjectKey)
                    .withDestinationBucketName(accessPointArn)
                    .withDestinationKey(destObjectKey)
                    .withUploadId(initResult.getUploadId())
                    .withFirstByte(bytePosition)
                    .withLastByte(lastByte)
                    .withPartNumber(partNum++);
                copyResponses.add(s3Client.copyPart(copyRequest));
                bytePosition += partSize;
            }

            // Complete the upload request to concatenate all uploaded parts and make the
copied object available.
            CompleteMultipartUploadRequest completeRequest = new
CompleteMultipartUploadRequest(
                accessPointArn,
                destObjectKey,
                initResult.getUploadId(),
                getETags(copyResponses));
        }
    }
}

```

```

        s3Client.completeMultipartUpload(completeRequest);
        System.out.println("Multipart copy complete.");
    } catch (AmazonServiceException e) {
        // The call was transmitted successfully, but Amazon S3 couldn't process
        // it, so it returned an error response.
        e.printStackTrace();
    } catch (SdkClientException e) {
        // Amazon S3 couldn't be contacted for a response, or the client
        // couldn't parse the response from Amazon S3.
        e.printStackTrace();
    }
}

// This is a helper function to construct a list of ETags.
private static List<PartETag> getETags(List<CopyPartResult> responses) {
    List<PartETag> etags = new ArrayList<PartETag>();
    for (CopyPartResult response : responses) {
        etags.add(new PartETag(response.getPartNumber(), response.getETag()));
    }
    return etags;
}

```

Copy a large object in an S3 on Outposts bucket by using multipart upload

The following S3 on Outposts example uses the SDK for Java to copy an object in a bucket. To use this example, replace each *user input placeholder* with your own information. This example is adapted from [Copying an object using multipart upload \(p. 196\)](#).

```

import com.amazonaws.AmazonServiceException;
import com.amazonaws.SdkClientException;
import com.amazonaws.services.s3.AmazonS3;
import com.amazonaws.services.s3.AmazonS3ClientBuilder;
import com.amazonaws.services.s3.model.*;

import java.util.ArrayList;
import java.util.List;

public class MultipartUploadCopy {
    public static void main(String[] args) {
        String accessPointArn = "*** Source access point ARN ***";
        String sourceObjectKey = "*** Source object key ***";
        String destObjectKey = "*** Target object key ***";

        try {
            // This code expects that you have AWS credentials set up per:
            // https://docs.aws.amazon.com/sdk-for-java/v1/developer-guide/setup-credentials.html
            AmazonS3 s3Client = AmazonS3ClientBuilder.standard()
                .enableUseArnRegion()
                .build();

            // Initiate the multipart upload.
            InitiateMultipartUploadRequest initRequest = new
            InitiateMultipartUploadRequest(accessPointArn, destObjectKey);
            InitiateMultipartUploadResult initResult =
            s3Client.initiateMultipartUpload(initRequest);

            // Get the object size to track the end of the copy operation.
            GetObjectMetadataRequest metadataRequest = new
            GetObjectMetadataRequest(accessPointArn, sourceObjectKey);
            ObjectMetadata metadataResult = s3Client.getObjectMetadata(metadataRequest);
            long objectSize = metadataResult.getContentLength();
        }
    }
}

```

```

// Copy the object using 5 MB parts.
long partSize = 5 * 1024 * 1024;
long bytePosition = 0;
int partNum = 1;
List<CopyPartResult> copyResponses = new ArrayList<CopyPartResult>();
while (bytePosition < objectSize) {
    // The last part might be smaller than partSize, so check to make sure
    // that lastByte isn't beyond the end of the object.
    long lastByte = Math.min(bytePosition + partSize - 1, objectSize - 1);

    // Copy this part.
    CopyPartRequest copyRequest = new CopyPartRequest()
        .withSourceBucketName(accessPointArn)
        .withSourceKey(sourceObjectKey)
        .withDestinationBucketName(accessPointArn)
        .withDestinationKey(destObjectKey)
        .withUploadId(initResult.getUploadId())
        .withFirstByte(bytePosition)
        .withLastByte(lastByte)
        .withPartNumber(partNum++);
    copyResponses.add(s3Client.copyPart(copyRequest));
    bytePosition += partSize;
}

// Complete the upload request to concatenate all uploaded parts and make the
copied object available.
CompleteMultipartUploadRequest completeRequest = new
CompleteMultipartUploadRequest(
    accessPointArn,
    destObjectKey,
    initResult.getUploadId(),
    getETags(copyResponses));
s3Client.completeMultipartUpload(completeRequest);
System.out.println("Multipart copy complete.");
} catch (AmazonServiceException e) {
    // The call was transmitted successfully, but Amazon S3 couldn't process
    // it, so it returned an error response.
    e.printStackTrace();
} catch (SdkClientException e) {
    // Amazon S3 couldn't be contacted for a response, or the client
    // couldn't parse the response from Amazon S3.
    e.printStackTrace();
}
}

// This is a helper function to construct a list of ETags.
private static List<PartETag> getETags(List<CopyPartResult> responses) {
    List<PartETag> etags = new ArrayList<PartETag>();
    for (CopyPartResult response : responses) {
        etags.add(new PartETag(response.getPartNumber(), response.getETag()));
    }
    return etags;
}
}

```

List parts of an object in an S3 on Outposts bucket

The following S3 on Outposts example lists the parts of an object in a bucket by using the SDK for Java. To use this example, replace each **user input placeholder** with your own information.

```

import com.amazonaws.AmazonServiceException;
import com.amazonaws.SdkClientException;
import com.amazonaws.services.s3.AmazonS3;

```

```

import com.amazonaws.services.s3.AmazonS3ClientBuilder;
import com.amazonaws.services.s3.model.*;

import java.util.List;

public class ListParts {
    public static void main(String[] args) {
        String accessPointArn = "*** access point ARN ***";
        String keyName = "*** Key name ***";
        String uploadId = "*** Upload ID ***";

        try {
            // This code expects that you have AWS credentials set up per:
            // https://docs.aws.amazon.com/sdk-for-java/v1/developer-guide/setup-
            credentials.html
            AmazonS3 s3Client = AmazonS3ClientBuilder.standard()
                .enableUseArnRegion()
                .build();

            ListPartsRequest listPartsRequest = new ListPartsRequest(accessPointArn,
keyName, uploadId);
            PartListing partListing = s3Client.listParts(listPartsRequest);
            List<PartSummary> partSummaries = partListing.getParts();

            System.out.println(partSummaries.size() + " multipart upload parts");
            for (PartSummary p : partSummaries) {
                System.out.println("Upload part: Part number = \'" + p.getPartNumber() +
"\", ETag = " + p.getETag());
            }

        } catch (AmazonServiceException e) {
            // The call was transmitted successfully, but Amazon S3 couldn't process
            // it, so it returned an error response.
            e.printStackTrace();
        } catch (SdkClientException e) {
            // Amazon S3 couldn't be contacted for a response, or the client
            // couldn't parse the response from Amazon S3.
            e.printStackTrace();
        }
    }
}

```

Retrieve a list of in-progress multipart uploads in an S3 on Outposts bucket

The following S3 on Outposts example shows how to retrieve a list of the in-progress multipart uploads from an Outposts bucket by using the SDK for Java. To use this example, replace each *user input placeholder* with your own information. This example is adapted from the [Listing multipart uploads \(p. 188\)](#) example for Amazon S3.

```

import com.amazonaws.AmazonServiceException;
import com.amazonaws.SdkClientException;
import com.amazonaws.services.s3.AmazonS3;
import com.amazonaws.services.s3.AmazonS3ClientBuilder;
import com.amazonaws.services.s3.model.ListMultipartUploadsRequest;
import com.amazonaws.services.s3.model.MultipartUpload;
import com.amazonaws.services.s3.model.MultipartUploadListing;

import java.util.List;

public class ListMultipartUploads {
    public static void main(String[] args) {

```

```
String accessPointArn = "*** access point ARN ***";

try {
    // This code expects that you have AWS credentials set up per:
    // https://docs.aws.amazon.com/sdk-for-java/v1/developer-guide/setup-
credentials.html
    AmazonS3 s3Client = AmazonS3ClientBuilder.standard()
        .enableUseArnRegion()
        .build();

    // Retrieve a list of all in-progress multipart uploads.
    ListMultipartUploadsRequest allMultipartUploadsRequest = new
ListMultipartUploadsRequest(accessPointArn);
    MultipartUploadListing multipartUploadListing =
s3Client.listMultipartUploads(allMultipartUploadsRequest);
    List<MultipartUpload> uploads = multipartUploadListing.getMultipartUploads();

    // Display information about all in-progress multipart uploads.
    System.out.println(uploads.size() + " multipart upload(s) in progress.");
    for (MultipartUpload u : uploads) {
        System.out.println("Upload in progress: Key = \"" + u.getKey() + "\", id =
" + u.getUploadId());
    }
} catch (AmazonServiceException e) {
    // The call was transmitted successfully, but Amazon S3 couldn't process
    // it, so it returned an error response.
    e.printStackTrace();
} catch (SdkClientException e) {
    // Amazon S3 couldn't be contacted for a response, or the client
    // couldn't parse the response from Amazon S3.
    e.printStackTrace();
}
}
```

Using presigned URLs for S3 on Outposts

To grant time-limited access to objects that are stored locally on an Outpost without updating your bucket policy, you can use a presigned URL. With presigned URLs, you as the bucket owner can share objects with individuals in your virtual private cloud (VPC) or grant them the ability to upload or delete objects.

When you create a presigned URL by using the AWS SDKs or the AWS Command Line Interface (AWS CLI), you associate the URL with a specific action. You also grant time-limited access to the presigned URL by choosing a custom expiration time that can be as low as 1 second and as high as 7 days. When you share the presigned URL, the individual in the VPC can perform the action embedded in the URL as if they were the original signing user. When the URL reaches its expiration time, the URL expires and no longer works.

Limiting presigned URL capabilities

The capabilities of a presigned URL are limited by the permissions of the user who created it. In essence, presigned URLs are bearer tokens that grant access to those who possess them. As such, we recommend that you protect them appropriately.

AWS Signature Version 4 (SigV4)

To enforce specific behavior when presigned URL requests are authenticated by using AWS Signature Version 4 (SigV4), you can use condition keys in bucket policies and access point policies. For example, you can create a bucket policy that uses the `s3-outposts:signatureAge` condition to deny any Amazon S3 on Outposts presigned URL request on objects in the `example-outpost-bucket`

bucket if the signature is more than 10 minutes old. To use this example, replace the *user input placeholders* with your own information.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "Deny a presigned URL request if the signature is more than 10 minutes old",
            "Effect": "Deny",
            "Principal": {"AWS":"44445556666"},
            "Action": "s3-outposts:*",
            "Resource": "arn:aws:s3-outposts:us-east-1:111122223333:outpost/op-01ac5d28a6a232904/bucket/example-outpost-bucket/object/*",
            "Condition": {
                "NumericGreaterThan": {"s3-outposts:signatureAge": 600000},
                "StringEquals": {"s3-outposts:authType": "REST-QUERY-STRING"}
            }
        }
    ]
}
```

For a list of condition keys and additional example policies that you can use to enforce specific behavior when presigned URL requests are authenticated by using Signature Version 4, see [AWS Signature Version 4 \(SigV4\) authentication-specific policy keys \(p. 1315\)](#).

Network path restriction

If you want to restrict the use of presigned URLs and all S3 on Outposts access to particular network paths, you can write policies that require a particular network path. To set the restriction on the IAM principal that makes the call, you can use identity-based AWS Identity and Access Management (IAM) policies (for example, IAM user, group, or role policies). To set the restriction on the S3 on Outposts resource, you can use resource-based policies (for example, bucket and access point policies).

A network-path restriction on the IAM principal requires the user of those credentials to make requests from the specified network. A restriction on the bucket or access point requires that all requests to that resource originate from the specified network. These restrictions also apply outside of the presigned URL scenario.

The IAM global condition that you use depends on the type of endpoint. If you are using the public endpoint for S3 on Outposts, use `aws:SourceIp`. If you are using a VPC endpoint for S3 on Outposts, use `aws:SourceVpc` or `aws:SourceVpce`.

The following IAM policy statement requires the principal to access AWS only from the specified network range. With this policy statement, all access must originate from that range. This includes the case of someone who's using a presigned URL for S3 on Outposts. To use this example, replace the *user input placeholders* with your own information.

```
{
    "Sid": "NetworkRestrictionForIAMPPrincipal",
    "Effect": "Deny",
    "Action": "*",
    "Resource": "*",
    "Condition": {
        "NotIpAddressIfExists": {"aws:SourceIp": "IP-address-range" },
        "BoolIfExists": {"aws:ViaAWSService": "false" }
    }
}
```

For an example bucket policy that uses the `aws:SourceIP` AWS global condition key to restrict access to an S3 on Outposts bucket to a specific network range, see [Setting up IAM with S3 on Outposts \(p. 1249\)](#).

Who can create a presigned URL

Anyone with valid security credentials can create a presigned URL. But for a user in the VPC to successfully access an object, the presigned URL must be created by someone who has permission to perform the operation that the presigned URL is based upon.

You can use the following credentials to create a presigned URL:

- **IAM instance profile** – Valid up to 6 hours.
- **AWS Security Token Service** – Valid up to 36 hours when signed with permanent credentials, such as the credentials of the AWS account root user or an IAM user.
- **IAM user** – Valid up to 7 days when you're using AWS Signature Version 4.

To create a presigned URL that's valid for up to 7 days, first delegate IAM user credentials (the access key and secret key) to the SDK that you're using. Then, generate a presigned URL by using AWS Signature Version 4.

Note

- If you created a presigned URL by using a temporary token, the URL expires when the token expires, even if you created the URL with a later expiration time.
- Because presigned URLs grant access to your S3 on Outposts buckets to whoever has the URL, we recommend that you protect them appropriately. For more information about protecting presigned URLs, see [Limiting presigned URL capabilities \(p. 1301\)](#).

When does S3 on Outposts check the expiration date and time of a presigned URL?

At the time of the HTTP request, S3 on Outposts checks the expiration date and time of a signed URL. For example, if a client begins to download a large file immediately before the expiration time, the download continues even if the expiration time passes during the download. However, if the connection drops and the client tries to restart the download after the expiration time passes, the download fails.

For more information about using a presigned URL to share or upload objects, see the following topics.

Topics

- [Sharing objects by using presigned URLs \(p. 1303\)](#)
- [Generating a presigned URL to upload an object to an S3 on Outposts bucket \(p. 1307\)](#)

Sharing objects by using presigned URLs

To grant time-limited access to objects that are stored locally on an Outpost without updating your bucket policy, you can use a presigned URL. With presigned URLs, you as the bucket owner can share objects with individuals in your virtual private cloud (VPC) or grant them the ability to upload or delete objects.

When you create a presigned URL by using the AWS SDKs or the AWS Command Line Interface (AWS CLI), you associate the URL with a specific action. You also grant time-limited access to the presigned URL by choosing a custom expiration time that can be as low as 1 second and as high as 7 days. When you share the presigned URL, the individual in the VPC can perform the action embedded in the URL as if they were the original signing user. When the URL reaches its expiration time, the URL expires and no longer works.

When you create a presigned URL, you must provide your security credentials, and then specify the following:

- An access point Amazon Resource Name (ARN) for the Amazon S3 on Outposts bucket
- An object key
- An HTTP method (GET for downloading objects)
- An expiration date and time

A presigned URL is valid only for the specified duration. That is, you must start the action that's allowed by the URL before the expiration date and time. You can use a presigned URL multiple times, up to the expiration date and time. If you created a presigned URL by using a temporary token, then the URL expires when the token expires, even if you created the URL with a later expiration time.

Users in the virtual private cloud (VPC) who have access to the presigned URL can access the object. For example, if you have a video in your bucket and both the bucket and the object are private, you can share the video with others by generating a presigned URL. Because presigned URLs grant access to your S3 on Outposts buckets to whoever has the URL, we recommend that you protect these URLs appropriately. For more details about protecting presigned URLs, see [Limiting presigned URL capabilities \(p. 1301\)](#).

Anyone with valid security credentials can create a presigned URL. However, the presigned URL must be created by someone who has permission to perform the operation that the presigned URL is based upon. For more information, see [Who can create a presigned URL \(p. 1303\)](#).

You can generate a presigned URL to share an object in an S3 on Outposts bucket by using the AWS SDKs and the AWS CLI. For more information, see the following examples.

Using the AWS SDKs

You can use the AWS SDKs to generate a presigned URL that you can give to others so that they can retrieve an object.

Note

When you use the AWS SDKs to generate a presigned URL, the maximum expiration time for a presigned URL is 7 days from the time of creation.

Java

Example

The following example generates a presigned URL that you can give to others so that they can retrieve an object from an S3 on Outposts bucket. For more information, see [Using presigned URLs for S3 on Outposts \(p. 1301\)](#). To use this example, replace the `user input placeholders` with your own information.

For instructions on creating and testing a working sample, see [Testing the Amazon S3 Java Code Examples \(p. 1192\)](#).

```
import com.amazonaws.AmazonServiceException;
import com.amazonaws.HttpMethod;
import com.amazonaws.SdkClientException;
import com.amazonaws.auth.profile.ProfileCredentialsProvider;
import com.amazonaws.regions.Regions;
import com.amazonaws.services.s3.AmazonS3;
import com.amazonaws.services.s3.AmazonS3ClientBuilder;
import com.amazonaws.services.s3.model.GeneratePresignedUrlRequest;

import java.io.IOException;
import java.net.URL;
import java.time.Instant;
```

```
public class GeneratePresignedURL {  
  
    public static void main(String[] args) throws IOException {  
        Regions clientRegion = Regions.DEFAULT_REGION;  
        String accessPointArn = "*** access point ARN ***";  
        String objectKey = "*** object key ***";  
  
        try {  
            AmazonS3 s3Client = AmazonS3ClientBuilder.standard()  
                .withRegion(clientRegion)  
                .withCredentials(new ProfileCredentialsProvider())  
                .build();  
  
            // Set the presigned URL to expire after one hour.  
            java.util.Date expiration = new java.util.Date();  
            long expTimeMillis = Instant.now().toEpochMilli();  
            expTimeMillis += 1000 * 60 * 60;  
            expiration.setTime(expTimeMillis);  
  
            // Generate the presigned URL.  
            System.out.println("Generating pre-signed URL.");  
            GeneratePresignedUrlRequest generatePresignedUrlRequest =  
                new GeneratePresignedUrlRequest(accessPointArn, objectKey)  
                    .withMethod(HttpMethod.GET)  
                    .withExpiration(expiration);  
            URL url = s3Client.generatePresignedUrl(generatePresignedUrlRequest);  
  
            System.out.println("Pre-Signed URL: " + url.toString());  
        } catch (AmazonServiceException e) {  
            // The call was transmitted successfully, but Amazon S3 couldn't process  
            // it, so it returned an error response.  
            e.printStackTrace();  
        } catch (SdkClientException e) {  
            // Amazon S3 couldn't be contacted for a response, or the client  
            // couldn't parse the response from Amazon S3.  
            e.printStackTrace();  
        }  
    }  
}
```

.NET

Example

The following example generates a presigned URL that you can give to others so that they can retrieve an object from an S3 on Outposts bucket. For more information, see [Using presigned URLs for S3 on Outposts \(p. 1301\)](#). To use this example, replace the *user input placeholders* with your own information.

For instructions on creating and testing a working sample, see [Running the Amazon S3 .NET Code Examples \(p. 1193\)](#).

```
using Amazon;  
using Amazon.S3;  
using Amazon.S3.Model;  
using System;  
  
namespace Amazon.DocSamples.S3  
{  
    class GenPresignedURLTest  
    {  
        private const string accessPointArn = "*** access point ARN ***";  
        private const string objectKey = "*** object key ***";
```

```
// Specify how long the presigned URL lasts, in hours.  
private const double timeoutDuration = 12;  
// Specify your bucket Region (an example Region is shown).  
private static readonly RegionEndpoint bucketRegion = RegionEndpoint.USWest2;  
private static IAmazonS3 s3Client;  
  
public static void Main()  
{  
    s3Client = new AmazonS3Client(bucketRegion);  
    string urlString = GeneratePreSignedURL(timeoutDuration);  
}  
static string GeneratePreSignedURL(double duration)  
{  
    string urlString = "";  
    try  
    {  
        GetPreSignedUrlRequest request1 = new GetPreSignedUrlRequest  
        {  
            BucketName = accessPointArn,  
            Key = objectKey,  
            Expires = DateTime.UtcNow.AddHours(duration)  
        };  
        urlString = s3Client.GetPreSignedURL(request1);  
    }  
    catch (AmazonS3Exception e)  
    {  
        Console.WriteLine("Error encountered on server. Message:'{0}' when  
writing an object", e.Message);  
    }  
    catch (Exception e)  
    {  
        Console.WriteLine("Unknown encountered on server. Message:'{0}' when  
writing an object", e.Message);  
    }  
    return urlString;  
}  
}  
}
```

Python

The following example generates a presigned URL to share an object by using the SDK for Python (Boto3). For example, use a Boto3 client and the `generate_presigned_url` function to generate a presigned URL that allows you to GET an object.

```
import boto3  
url = boto3.client('s3').generate_presigned_url(  
    ClientMethod='get_object',  
    Params={'Bucket': 'ACCESS_POINT_ARN', 'Key': 'OBJECT_KEY'},  
    ExpiresIn=3600)
```

For more information about using the SDK for Python (Boto3) to generate a presigned URL, see [Python in the AWS SDK for Python \(Boto\) API Reference](#).

Using the AWS CLI

The following example AWS CLI command generates a presigned URL for an S3 on Outposts bucket. To use this example, replace the `user input placeholders` with your own information.

Note

When you use the AWS CLI to generate a presigned URL, the maximum expiration time for a presigned URL is 7 days from the time of creation.

```
aws s3 presign s3://arn:aws:s3-outposts:us-  
east-1:111122223333:outpost/op-01ac5d28a6a232904/accesspoint/example-outpost-access-  
point/mydoc.txt --expires-in 604800
```

For more information, see [presign](#) in the *AWS CLI Command Reference*.

Generating a presigned URL to upload an object to an S3 on Outposts bucket

To grant time-limited access to objects that are stored locally on an Outpost without updating your bucket policy, you can use a presigned URL. With presigned URLs, you as the bucket owner can share objects with individuals in your virtual private cloud (VPC) or grant them the ability to upload or delete objects.

When you create a presigned URL by using the AWS SDKs or the AWS Command Line Interface (AWS CLI), you associate the URL with a specific action. You also grant time-limited access to the presigned URL by choosing a custom expiration time that can be as low as 1 second and as high as 7 days. When you share the presigned URL, the individual in the VPC can perform the action embedded in the URL as if they were the original signing user. When the URL reaches its expiration time, the URL expires and no longer works.

When you create a presigned URL, you must provide your security credentials, and then specify the following:

- An access point Amazon Resource Name (ARN) for the Amazon S3 on Outposts bucket
- An object key
- An HTTP method (`PUT` for uploading objects)
- An expiration date and time

A presigned URL is valid only for the specified duration. That is, you must start the action that's allowed by the URL before the expiration date and time. You can use a presigned URL multiple times, up to the expiration date and time. If you created a presigned URL by using a temporary token, then the URL expires when the token expires, even if you created the URL with a later expiration time.

If the action allowed by a presigned URL consists of multiple steps, such as a multipart upload, you must start all steps before the expiration time. If S3 on Outposts tries to start a step with an expired URL, you receive an error.

Users in the virtual private cloud (VPC) who have access to the presigned URL can upload objects. For example, a user in the VPC who has access to the presigned URL can upload an object to your bucket. Because presigned URLs grant access to your S3 on Outposts bucket to any user in the VPC who has access to the presigned URL, we recommend that you protect these URLs appropriately. For more details about protecting presigned URLs, see [Limiting presigned URL capabilities \(p. 1301\)](#).

Anyone with valid security credentials can create a presigned URL. However, the presigned URL must be created by someone who has permission to perform the operation that the presigned URL is based upon. For more information, see [Who can create a presigned URL \(p. 1303\)](#).

Using the AWS SDKs to generate a presigned URL for an S3 on Outposts object operation

Java

SDK for Java 2.x

This example shows how to generate a presigned URL that you can use to upload an object to an S3 on Outposts bucket for a limited time. For more information, see [Using presigned URLs for S3 on Outposts \(p. 1301\)](#).

```
public static void signBucket(S3Presigner presigner, String
outpostAccessPointArn, String keyName) {

    try {
        PutObjectRequest objectRequest = PutObjectRequest.builder()
            .bucket(accessPointArn)
            .key(keyName)
            .contentType("text/plain")
            .build();

        PutObjectPresignRequest presignRequest =
PutObjectPresignRequest.builder()
            .signatureDuration(Duration.ofMinutes(10))
            .putObjectRequest(objectRequest)
            .build();

        PresignedPutObjectRequest presignedRequest =
presigner.presignPutObject(presignRequest);

        String myURL = presignedRequest.url().toString();
        System.out.println("Presigned URL to upload a file to: " +myURL);
        System.out.println("Which HTTP method must be used when uploading a
file: " +
                presignedRequest.httpRequest().method());

        // Upload content to the S3 on Outposts bucket by using this URL.
        URL url = presignedRequest.url();

        // Create the connection and use it to upload the new object by using
the presigned URL.
        HttpURLConnection connection = (HttpURLConnection)
url.openConnection();
        connection.setDoOutput(true);
        connection.setRequestProperty("Content-Type", "text/plain");
        connection.setRequestMethod("PUT");
        OutputStreamWriter out = new
OutputStreamWriter(connection.getOutputStream());
        out.write("This text was uploaded as an object by using a presigned
URL.");
        out.close();

        connection.getResponseCode();
        System.out.println("HTTP response code is " +
connection.getResponseCode());

    } catch (S3Exception e) {
        e.printStackTrace();
    } catch (IOException e) {
        e.printStackTrace();
    }
}
```

Python

SDK for Python (Boto3)

This example shows how to generate a presigned URL that can perform an S3 on Outposts action for a limited time. For more information, see [Using presigned URLs for S3 on Outposts \(p. 1301\)](#). To make a request with the URL, use the Requests package.

```
import argparse
```

```
import logging
import boto3
from botocore.exceptions import ClientError
import requests

logger = logging.getLogger(__name__)

def generate_presigned_url(s3_client, client_method, method_parameters,
                           expires_in):
    """
    Generate a presigned S3 on Outposts URL that can be used to perform an action.

    :param s3_client: A Boto3 Amazon S3 client.
    :param client_method: The name of the client method that the URL performs.
    :param method_parameters: The parameters of the specified client method.
    :param expires_in: The number of seconds that the presigned URL is valid for.
    :return: The presigned URL.
    """
    try:
        url = s3_client.generate_presigned_url(
            ClientMethod=client_method,
            Params=method_parameters,
            ExpiresIn=expires_in
        )
        logger.info("Got presigned URL: %s", url)
    except ClientError:
        logger.exception(
            "Couldn't get a presigned URL for client method '%s'.", client_method)
        raise
    return url

def usage_demo():
    logging.basicConfig(level=logging.INFO, format='%(levelname)s: %(message)s')

    print('*'*88)
    print("Welcome to the Amazon S3 on Outposts presigned URL demo.")
    print('*'*88)

    parser = argparse.ArgumentParser()
    parser.add_argument('accessPointArn', help="The name of the S3 on Outposts access point ARN.")
    parser.add_argument(
        'key', help="For a GET operation, the key of the object in S3 on Outposts. For a "
                    "PUT operation, the name of a file to upload.")
    parser.add_argument(
        'action', choices=('get', 'put'), help="The action to perform.")
    args = parser.parse_args()

    s3_client = boto3.client('s3')
    client_action = 'get_object' if args.action == 'get' else 'put_object'
    url = generate_presigned_url(
        s3_client, client_action, {'Bucket': args.accessPointArn, 'Key': args.key},
        1000)

    print("Using the Requests package to send a request to the URL.")
    response = None
    if args.action == 'get':
        response = requests.get(url)
    elif args.action == 'put':
        print("Putting data to the URL.")
        try:
            with open(args.key, 'r') as object_file:
                object_text = object_file.read()
```

```
response = requests.put(url, data=object_text)
except FileNotFoundError:
    print(f"Couldn't find {args.key}. For a PUT operation, the key must be
the "
         f"name of a file that exists on your computer.")

if response is not None:
    print("Got response:")
    print(f"Status: {response.status_code}")
    print(response.text)

print('*'*88)

if __name__ == '__main__':
    usage_demo()
```

Security in S3 on Outposts

Cloud security at AWS is the highest priority. As an AWS customer, you benefit from data centers and network architectures that are built to meet the requirements of the most security-sensitive organizations.

Security is a shared responsibility between AWS and you. The [shared responsibility model](#) describes this as security of the cloud and security *in* the cloud:

- **Security of the cloud** – AWS is responsible for protecting the infrastructure that runs AWS services in the AWS Cloud. AWS also provides you with services that you can use securely. Third-party auditors regularly test and verify the effectiveness of our security as part of the [AWS Compliance Programs](#). To learn about the compliance programs that apply to Amazon S3 on Outposts, see [AWS Services in Scope by Compliance Program](#).
- **Security in the cloud** – Your responsibility is determined by the AWS service that you use. You are also responsible for other factors including the sensitivity of your data, your company's requirements, and applicable laws and regulations.

This documentation helps you understand how to apply the shared responsibility model when using S3 on Outposts. The following topics show you how to configure S3 on Outposts to meet your security and compliance objectives. You also learn how to use other AWS services that help you to monitor and secure your S3 on Outposts resources.

Topics

- [Data encryption in S3 on Outposts \(p. 1310\)](#)
- [AWS PrivateLink for S3 on Outposts \(p. 1311\)](#)
- [AWS Signature Version 4 \(SigV4\) authentication-specific policy keys \(p. 1315\)](#)

Data encryption in S3 on Outposts

By default, all data stored in Amazon S3 on Outposts is encrypted by using server-side encryption with Amazon S3 managed encryption keys (SSE-S3). For more information, see [Protecting data using server-side encryption with Amazon S3-managed encryption keys \(SSE-S3\) \(p. 355\)](#).

You can optionally use server-side encryption with customer-provided encryption keys (SSE-C). To use SSE-C, specify an encryption key as part of your object API requests. Server-side encryption encrypts

only the object data, not the object metadata. For more information, see [Protecting data using server-side encryption with customer-provided encryption keys \(SSE-C\) \(p. 366\)](#).

Note

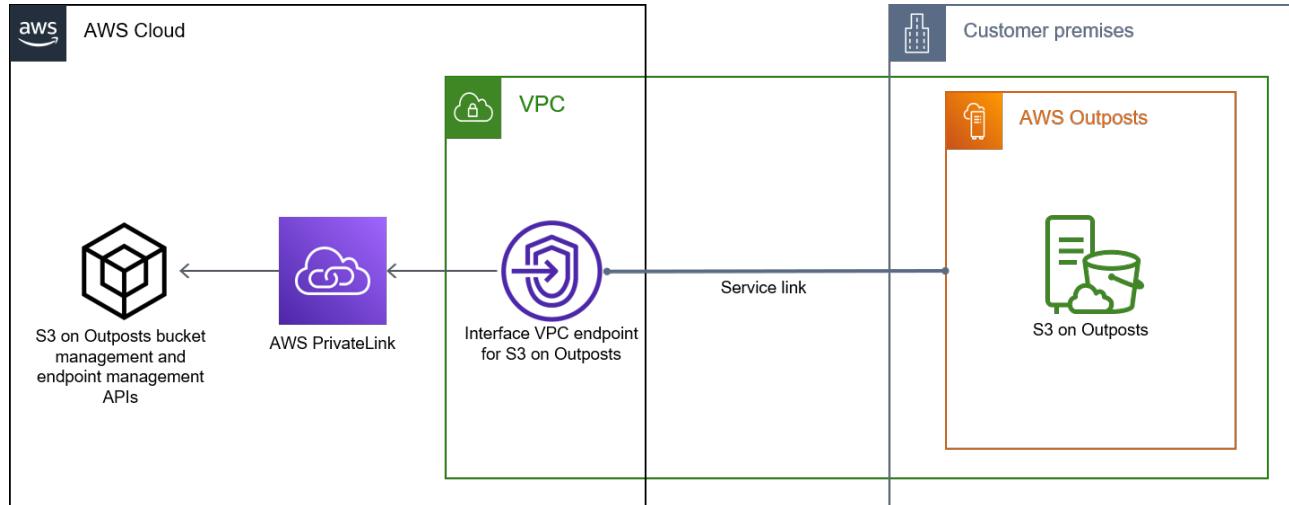
S3 on Outposts doesn't support server-side encryption with AWS Key Management Service (AWS KMS) keys (SSE-KMS).

AWS PrivateLink for S3 on Outposts

With AWS PrivateLink for Amazon S3 on Outposts, you can provision *interface VPC endpoints* in your virtual private cloud (VPC) to manage your S3 on Outposts deployment through the S3 on Outposts [bucket management \(p. 1324\)](#) and [endpoint management \(p. 1325\)](#) APIs. Interface VPC endpoints are directly accessible from applications that are deployed in your VPC or on premises over your virtual private network (VPN) or AWS Direct Connect. You can access the bucket management and endpoint management APIs through AWS PrivateLink. AWS PrivateLink doesn't support [data transfer \(p. 1323\)](#) API operations, such as GET, PUT, and similar APIs. These operations are transferred privately through the S3 on Outposts endpoint and access point configuration. For more information, see [Networking for S3 on Outposts \(p. 1260\)](#).

Interface endpoints are represented by one or more elastic network interfaces (ENIs) that are assigned private IP addresses from subnets in your VPC. Requests that are made to interface endpoints for S3 on Outposts are automatically routed to S3 on Outposts bucket management and endpoint management APIs on the Amazon network. You can also access interface endpoints in your VPC from on-premises applications through AWS Direct Connect or AWS Virtual Private Network (AWS VPN). For more information about how to connect your VPC with your on-premises network, see the [AWS Direct Connect User Guide](#) and the [AWS Site-to-Site VPN User Guide](#).

Interface endpoints route requests for S3 on Outposts bucket management and endpoint management APIs over the Amazon network and through AWS PrivateLink, as illustrated in the following diagram.



For general information about interface endpoints, see [Interface VPC endpoints \(AWS PrivateLink\)](#) in the [AWS PrivateLink Guide](#).

Topics

- [Restrictions and limitations \(p. 1312\)](#)
- [Accessing S3 on Outposts interface endpoints \(p. 1312\)](#)
- [Updating an on-premises DNS configuration \(p. 1313\)](#)
- [Creating a VPC endpoint for S3 on Outposts \(p. 1313\)](#)

- [Creating bucket policies and VPC endpoint policies for S3 on Outposts \(p. 1313\)](#)

Restrictions and limitations

When you access S3 on Outposts bucket management and endpoint management APIs through AWS PrivateLink, VPC limitations apply. For more information, see [Interface endpoint properties and limitations](#) and [AWS PrivateLink quotas](#) in the *AWS PrivateLink Guide*.

In addition, AWS PrivateLink doesn't support the following:

- [Federal Information Processing Standard \(FIPS\) endpoints](#)
- [S3 on Outposts data transfer APIs \(p. 1323\)](#), for example, GET, PUT, and similar object API operations.

Accessing S3 on Outposts interface endpoints

To access S3 on Outposts bucket management and endpoint management APIs using AWS PrivateLink, you *must* update your applications to use endpoint-specific DNS names. When you create an interface endpoint, AWS PrivateLink generates two types of endpoint-specific S3 on Outposts names: *Regional* and *zonal*.

- **Regional DNS names** – include a unique VPC endpoint ID, a service identifier, the AWS Region, and `vpce.amazonaws.com`, for example, `vpce-1a2b3c4d-5e6f.s3-outposts.us-east-1.vpce.amazonaws.com`.
- **Zonal DNS names** – include a unique VPC endpoint ID, the Availability Zone, a service identifier, the AWS Region, and `vpce.amazonaws.com`, for example, `vpce-1a2b3c4d-5e6f-us-east-1a.s3-outposts.us-east-1.vpce.amazonaws.com`. You might use this option if your architecture isolates Availability Zones. For example, you could use zonal DNS names for fault containment or to reduce Regional data transfer costs.

Endpoint-specific S3 on Outposts DNS names can be resolved from the S3 on Outposts public DNS domain. S3 on Outposts interface endpoints also support the private DNS feature of interface endpoints. For more information about [Private DNS for interface endpoints](#), see the *AWS PrivateLink Guide*.

AWS CLI examples

Use the `--region` and `--endpoint-url` parameters to access bucket management and endpoint management APIs through S3 on Outposts interface endpoints.

Example : Use the endpoint URL to list buckets with the S3 control API

In the following example, replace the Region `us-east-1`, VPC endpoint URL `vpce-1a2b3c4d-5e6f.s3.us-east-1.vpce.amazonaws.com`, and account ID `111122223333` with appropriate information.

```
aws s3control list-regional-buckets --region us-east-1 --endpoint-url https://vpce-1a2b3c4d-5e6f.s3-outposts.us-east-1.vpce.amazonaws.com --account-id 111122223333
```

AWS SDK examples

Update your SDKs to the latest version, and configure your clients to use an endpoint URL for accessing the S3 control API for S3 on Outposts interface endpoints. For more information, see [AWS SDK examples for AWS PrivateLink](#).

SDK for Python (Boto3)

Example : Use an endpoint URL to access the S3 control API

In the following example, replace the Region `us-east-1` and VPC endpoint URL `vpce-1a2b3c4d-5e6f.s3-outposts.us-east-1.vpce.amazonaws.com` with appropriate information.

```
control_client = session.client(  
    service_name='s3control',  
    region_name='us-east-1',  
    endpoint_url='https://vpce-1a2b3c4d-5e6f.s3-outposts.us-east-1.vpce.amazonaws.com'  
)
```

SDK for Java 2.x

Example : Use an endpoint URL to access the S3 control API

In the following example, replace the VPC endpoint URL `vpce-1a2b3c4d-5e6f.s3-outposts.us-east-1.vpce.amazonaws.com` and the Region `Region.US_EAST_1` with appropriate information.

```
// control client  
Region region = Region.US_EAST_1;  
S3ControlClient s3ControlClient = S3ControlClient.builder().region(region)  
  
.endpointOverride(URI.create("https://vpce-1a2b3c4d-5e6f.s3-outposts.us-  
east-1.vpce.amazonaws.com"))  
.build()
```

Updating an on-premises DNS configuration

When using endpoint-specific DNS names to access the interface endpoints for S3 on Outposts bucket management and endpoint management APIs, you don't have to update your on-premises DNS resolver. You can resolve the endpoint-specific DNS name with the private IP address of the interface endpoint from the public S3 on Outposts DNS domain.

Creating a VPC endpoint for S3 on Outposts

To create a VPC interface endpoint for S3 on Outposts, see [Create a VPC endpoint in the AWS PrivateLink Guide](#).

Creating bucket policies and VPC endpoint policies for S3 on Outposts

You can attach an endpoint policy to your VPC endpoint that controls access to S3 on Outposts. You can also use the `aws:sourceVpce` condition in S3 on Outposts bucket policies to restrict access to specific buckets from a specific VPC endpoint. With VPC endpoint policies, you can control access to S3 on Outposts bucket management APIs and endpoint management APIs. With bucket policies, you can control access to the S3 on Outposts bucket management APIs. However, you can't manage access to object actions for S3 on Outposts using `aws:sourceVpce`.

Access policies for S3 on Outposts specify the following information:

- The AWS Identity and Access Management (IAM) principal for which actions are allowed or denied.
- The S3 control actions that are allowed or denied.

- The S3 on Outposts resources on which actions are allowed or denied.

The following examples show policies that restrict access to a bucket or to an endpoint. For more information about VPC connectivity, see [Network-to-VPC connectivity options](#) in the AWS whitepaper [Amazon Virtual Private Cloud Connectivity Options](#).

Important

- When you apply the example policies for VPC endpoints described in this section, you might block your access to the bucket without intending to do so. Bucket permissions that limit bucket access to connections originating from your VPC endpoint can block all connections to the bucket. For information about how to fix this issue, see [My bucket policy has the wrong VPC or VPC endpoint ID. How can I fix the policy so that I can access the bucket?](#) in the AWS Support Knowledge Center.
- Before using the following example bucket policies, replace the VPC endpoint ID with an appropriate value for your use case. Otherwise, you won't be able to access your bucket.
- If your policy only allows access to an S3 on Outposts bucket from a specific VPC endpoint, it disables console access for that bucket because console requests don't originate from the specified VPC endpoint.

Topics

- [Example: Restricting access to a specific bucket from a VPC endpoint \(p. 393\)](#)
- [Example: Denying access from a specific VPC endpoint in an S3 on Outposts bucket policy \(p. 1314\)](#)

Example: Restricting access to a specific bucket from a VPC endpoint

You can create an endpoint policy that restricts access to specific S3 on Outposts buckets only. The following policy restricts access for the GetBucketPolicy action only to the `example-outpost-bucket`. To use this policy, replace the example values with your own.

```
{  
    "Version": "2012-10-17",  
    "Id": "Policy1415115909151",  
    "Statement": [  
        { "Sid": "Access-to-specific-bucket-only",  
          "Principal": {"AWS": "111122223333"},  
          "Action": "s3-outposts:GetBucketPolicy",  
          "Effect": "Allow",  
          "Resource": "arn:aws:s3-outposts:region:123456789012:outpost/op-01ac5d28a6a232904/bucket/example-outpost-bucket"  
        }  
    ]  
}
```

Example: Denying access from a specific VPC endpoint in an S3 on Outposts bucket policy

The following S3 on Outposts bucket policy denies access to GetBucketPolicy on the `example-outpost-bucket` bucket through the `vpce-1a2b3c4d` VPC endpoint.

The `aws:sourceVpce` condition specifies the endpoint and does not require an Amazon Resource Name (ARN) for the VPC endpoint resource, only the endpoint ID. To use this policy, replace the example values with your own.

```
{
```

```

    "Version": "2012-10-17",
    "Id": "Policy1415115909152",
    "Statement": [
        {
            "Sid": "Deny-access-to-specific-VPCE",
            "Principal": {"AWS":"111122223333"},
            "Action": "s3-outposts:GetBucketPolicy",
            "Effect": "Deny",
            "Resource": "arn:aws:s3-
outposts:region:123456789012:outpost/op-01ac5d28a6a232904/bucket/example-outpost-bucket",
            "Condition": {
                "StringEquals": {"aws:sourceVpce": "vpce-1a2b3c4d"}
            }
        }
    ]
}

```

AWS Signature Version 4 (SigV4) authentication-specific policy keys

The following table shows the condition keys related to AWS Signature Version 4 (SigV4) authentication that you can use with Amazon S3 on Outposts. In a bucket policy, you can add these conditions to enforce specific behavior when requests are authenticated by using Signature Version 4. For example policies, see [Bucket policy examples that use Signature Version 4-related condition keys \(p. 1316\)](#). For more information about authenticating requests using Signature Version 4, see [Authenticating requests \(AWS Signature Version 4\)](#) in the *Amazon Simple Storage Service API Reference*

Applicable keys for `s3-outposts:*` actions or any of the S3 on Outposts actions

Applicable keys	Description
<code>s3-outposts:authType</code>	S3 on Outposts supports various methods of authentication. To restrict incoming requests to use a specific authentication method, you can use this optional condition key. For example, you can use this condition key to allow only the HTTP Authorization header to be used in request authentication. Valid values: REST-HEADER REST-QUERY-STRING
<code>s3-outposts:signatureAge</code>	The length of time, in milliseconds, that a signature is valid in an authenticated request. This condition works only for presigned URLs. In Signature Version 4, the signing key is valid for up to seven days. Therefore, the signatures are also valid for up to seven days. For more information, see Introduction to signing requests in the <i>Amazon Simple Storage Service API Reference</i> . You can use this condition to further limit the signature age. Example value: 600000
<code>s3-outposts:x-amz-content-sha256</code>	You can use this condition key to disallow unsigned content in your bucket.

Applicable keys	Description
	<p>When you use Signature Version 4, for requests that use the Authorization header, you add the <code>x-amz-content-sha256</code> header in the signature calculation and then set its value to the hash payload.</p> <p>You can use this condition key in your bucket policy to deny any uploads where the payloads are not signed. For example:</p> <ul style="list-style-type: none"> Deny uploads that use the Authorization header to authenticate requests but don't sign the payload. For more information, see Transferring payload in a single chunk in the <i>Amazon Simple Storage Service API Reference</i>. Deny uploads that use presigned URLs. Presigned URLs always have an <code>UNSIGNED_PAYLOAD</code>. For more information, see Authenticating requests and Authentication methods in the <i>Amazon Simple Storage Service API Reference</i>. <p>Valid value: <code>UNSIGNED-PAYLOAD</code></p>

Bucket policy examples that use Signature Version 4-related condition keys

To use the following examples, replace the `user input placeholders` with your own information.

Example : s3-outposts:signatureAge

The following bucket policy denies any S3 on Outposts presigned URL request on objects in `example-outpost-bucket` if the signature is more than 10 minutes old.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "Deny a presigned URL request if the signature is more than 10 minutes old",
            "Effect": "Deny",
            "Principal": {"AWS":"444455556666"},
            "Action": "s3-outposts:*",
            "Resource": "arn:aws:s3-outposts:us-east-1:111122223333:outpost/op-01ac5d28a6a232904/bucket/example-outpost-bucket/object/*",
            "Condition": {
                "NumericGreaterThan": {"s3-outposts:signatureAge": 600000},
                "StringEquals": {"s3-outposts:authType": "REST-QUERY-STRING"}
            }
        }
    ]
}
```

Example : s3-outposts:authType

The following bucket policy allows only requests that use the Authorization header for request authentication. Any presigned URL requests will be denied since presigned URLs use query parameters to provide request and authentication information. For more information, see [Authentication methods](#) in the *Amazon Simple Storage Service API Reference*.

```
{
```

```

    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "Allow only requests that use the Authorization header for request authentication. Deny presigned URL requests.",
            "Effect": "Deny",
            "Principal": {"AWS":"111122223333"},
            "Action": "s3-outposts:*",
            "Resource": "arn:aws:s3-outposts:us-
east-1:111122223333:outpost/op-01ac5d28a6a232904/bucket/example-outpost-bucket/object/*",
            "Condition": {
                "StringNotEquals": {
                    "s3-outposts:authType": "REST-HEADER"
                }
            }
        }
    ]
}

```

Example : s3-outposts:x-amz-content-sha256

The following bucket policy denies any uploads with unsigned payloads, such as uploads that are using presigned URLs. For more information, see [Authenticating requests](#) and [Authentication methods](#) in the *Amazon Simple Storage Service API Reference*.

```

{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "Deny uploads with unsigned payloads.",
            "Effect": "Deny",
            "Principal": {"AWS":"111122223333"},
            "Action": "s3-outposts:*",
            "Resource": "arn:aws:s3-outposts:us-
east-1:111122223333:outpost/op-01ac5d28a6a232904/bucket/example-outpost-bucket/object/*",
            "Condition": {
                "StringEquals": {
                    "s3-outposts:x-amz-content-sha256": "UNSIGNED-PAYLOAD"
                }
            }
        }
    ]
}

```

Managing S3 on Outposts storage

With Amazon S3 on Outposts, you can create S3 buckets on your AWS Outposts and easily store and retrieve objects on premises for applications that require local data access, local data processing, and data residency. S3 on Outposts provides a new storage class, S3 Outposts (OUTPOSTS), which uses the Amazon S3 APIs, and is designed to store data durably and redundantly across multiple devices and servers on your AWS Outposts. You communicate with your Outpost bucket by using an access point and endpoint connection over a virtual private cloud (VPC). You can use the same APIs and features on Outpost buckets as you do on Amazon S3 buckets, including access policies, encryption, and tagging. You can use S3 on Outposts through the AWS Management Console, AWS Command Line Interface (AWS CLI), AWS SDKs, or REST API. For more information, see [What is Amazon S3 on Outposts? \(p. 1241\)](#)

For more information about managing, monitoring, and sharing your Amazon S3 on Outposts storage capacity, see the following topics.

Topics

- [Managing S3 on Outposts capacity with Amazon CloudWatch metrics \(p. 1318\)](#)
- [Receiving S3 on Outposts event notifications using Amazon CloudWatch Events \(p. 1319\)](#)
- [Managing S3 on Outposts capacity with AWS CloudTrail logs \(p. 1319\)](#)
- [Sharing S3 on Outposts by using AWS RAM \(p. 1320\)](#)
- [Other AWS services that use S3 on Outposts \(p. 1322\)](#)

Managing S3 on Outposts capacity with Amazon CloudWatch metrics

If there is not enough space to store an object on your Outpost, the API returns an insufficient capacity exemption (ICE). To avoid this, you can create CloudWatch alerts that tell you when storage utilization exceeds a certain threshold. For more information, see [Amazon S3 on Outposts metrics in CloudWatch \(p. 1008\)](#).

You can use this method to free up space by explicitly deleting data, using a lifecycle expiration policy, or copying data from your Amazon S3 on Outposts bucket to an S3 bucket in an AWS Region by using AWS DataSync. For more information about using DataSync, see [Getting Started with AWS DataSync](#) in the [AWS DataSync User Guide](#).

CloudWatch metrics

The `S3Outposts` namespace includes the following metrics for Amazon S3 on Outposts buckets. You can monitor the total number of S3 on Outposts bytes provisioned, the total free bytes available for objects, and the total size of all objects for a given bucket.

Note

S3 on Outposts supports only the following metrics, and no other Amazon S3 metrics. Because S3 on Outposts has fixed capacity, you can create CloudWatch alerts that alert you when your storage utilization exceeds a certain threshold.

Metric	Description
<code>OutpostTotalBytes</code>	The total provisioned capacity in bytes for an Outpost. Units: Bytes Period: 5 minutes
<code>OutpostFreeBytes</code>	The count of free bytes available on an Outpost to store customer data. Units: Bytes Period: 5 minutes
<code>BucketUsedBytes</code>	The total size of all objects for the given bucket. Units: Counts Period: 5 minutes
<code>AccountUsedBytes</code>	The total size of all objects for the specified Outposts account. Units: Bytes Period: 5 minutes

Receiving S3 on Outposts event notifications using Amazon CloudWatch Events

You can use CloudWatch Events to create a rule for any Amazon S3 on Outposts API event to get notified through all supported CloudWatch targets, including Amazon Simple Queue Service (Amazon SQS), Amazon Simple Notification Service (Amazon SNS), and AWS Lambda. For more information, see the list of [AWS services that can be targets for CloudWatch Events](#) in the *Amazon CloudWatch Events User Guide*. To choose a target service to work with your S3 on Outposts, see [Creating a CloudWatch Events rule that triggers on an AWS API call using AWS CloudTrail](#) in the *Amazon CloudWatch Events User Guide*.

Note

For S3 on Outposts object operations, AWS API call events sent by CloudTrail will only match your rules if you have trails (optionally with event selectors) configured to receive those events. For more information, see [Working with CloudTrail log files](#) in the *AWS CloudTrail User Guide*.

Example

The following is a sample rule for the `DeleteObject` operation. To use this sample rule, replace `DOC-EXAMPLE-BUCKET1` with the name of your S3 on Outposts bucket.

```
{  
    "source": [  
        "aws.s3-outposts"  
    ],  
    "detail-type": [  
        "AWS API call through CloudTrail"  
    ],  
    "detail": {  
        "eventSource": [  
            "s3-outposts.amazonaws.com"  
        ],  
        "eventName": [  
            "DeleteObject"  
        ],  
        "requestParameters": {  
            "bucketName": [  
                "DOC-EXAMPLE-BUCKET1"  
            ]  
        }  
    }  
}
```

Managing S3 on Outposts capacity with AWS CloudTrail logs

Your Amazon S3 on Outposts management events are available through AWS CloudTrail logs. For more information, see [Logging Amazon S3 API calls using AWS CloudTrail \(p. 962\)](#).

In addition, you can optionally enable logging for data events in CloudTrail. For more information, see [Enable logging for objects in a bucket using the console \(p. 971\)](#).

Configuring your S3 on Outposts bucket for CloudTrail logging using the S3 console

The following procedure shows you how to configure your S3 on Outposts bucket to emit to AWS CloudTrail logs.

Note

The AWS account that creates the bucket owns it and is the only one that can configure Amazon S3 data events to be sent to AWS CloudTrail.

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 **Outposts buckets**.
3. Choose the Outposts bucket whose data events you want to log by using CloudTrail.
4. Go to the section **AWS CloudTrail data events**, and choose **Configure in CloudTrail**.

For more information, see [Enabling CloudTrail event logging for S3 buckets and objects \(p. 971\)](#).

Sharing S3 on Outposts by using AWS RAM

Amazon S3 on Outposts supports sharing S3 capacity across multiple accounts within an organization by using AWS Resource Access Manager ([AWS RAM](#)). With S3 on Outposts sharing, you can allow others to create and manage buckets, endpoints, and access points on your Outpost.

This topic demonstrates how to use AWS RAM to share S3 on Outposts and related resources with another AWS account in your AWS organization.

Prerequisites

- The Outpost owner account has an organization configured in AWS Organizations. For more information, see [Creating an organization](#) in the *AWS Organizations User Guide*.
- The organization includes the AWS account that you want to share your S3 on Outposts capacity with. For more information, see [Sending invitations to AWS accounts](#) in the *AWS Organizations User Guide*.
- Select one of the following options that you want to share. The second resource (either **Subnets** or **Outposts**) must be selected so that endpoints are also accessible. Endpoints are a networking requirement in order to access data stored in S3 on Outposts.

Option 1	Option 2
S3 on Outposts Allows the user to create buckets on your Outposts and access points and to add objects to those buckets.	S3 on Outposts Allows the user to create buckets on your Outposts and access points and to add objects to those buckets.
Subnets Allows the user to use your virtual private cloud (VPC) and the endpoints that are associated with your subnet.	Outposts Allows the user to see S3 capacity charts and the AWS Outposts console home page. Also allows users to create subnets on shared Outposts and create endpoints.

Procedure

1. Sign in to the AWS Management Console by using the AWS account that owns the Outpost, and then open the AWS RAM console at <https://console.aws.amazon.com/ram>.
2. Make sure that you have enabled sharing with AWS Organizations in AWS RAM. For information, see [Enable resource sharing within AWS Organizations](#) in the *AWS RAM User Guide*.

3. Use either Option 1 or Option 2 in the [prerequisites \(p. 1320\)](#) to create a resource share. If you have multiple S3 on Outposts resources, select the Amazon Resource Names (ARNs) of the resources that you want to share. To enable endpoints, share either your subnet or Outpost.

For more information about how to create a resource share, see [Create a resource share in the AWS RAM User Guide](#).

4. The AWS account that you shared your resources with should now be able to use S3 on Outposts. Depending on the option that you selected in the [prerequisites \(p. 1320\)](#), provide the following information to the account user:

Option 1	Option 2
The Outpost ID	The Outpost ID
The VPC ID	
The subnet ID	
The security group ID	

Note

The user can confirm that the resources have been shared with them by using the AWS RAM console, the AWS Command Line Interface (AWS CLI), AWS SDKs, or REST API. The user can view their existing resource shares by using the [get-resource-shares](#) CLI command.

Usage examples

After you have shared your S3 on Outposts resources with another account, that account can manage buckets and objects on your Outpost. If you shared the **Subnets** resource, then that account can use the endpoint that you created. The following examples demonstrate how a user can use the AWS CLI to interact with your Outpost after you share these resources.

Example : Create a bucket

The following example creates a bucket named `DOC-EXAMPLE-BUCKET1` on the Outpost `op-01ac5d28a6a232904`. Before using this command, replace each `user input placeholder` with the appropriate values for your use case.

```
aws s3control create-bucket --bucket DOC-EXAMPLE-BUCKET1 --outpost-id op-01ac5d28a6a232904
```

For more information about this command, see [create-bucket](#) in the *AWS CLI Reference*.

Example : Create an access point

The following example creates an access point on an Outpost by using the example parameters in the following table. Before using this command, replace these `user input placeholder` values and the AWS Region code with the appropriate values for your use case.

Parameter	Value
Account ID	<code>111122223333</code>
Access point name	<code>example-outpost-access-point</code>
Outpost ID	<code>op-01ac5d28a6a232904</code>

Parameter	Value
Outpost bucket name	DOC-EXAMPLE-BUCKET1
VPC ID	vpc-1a2b3c4d5e6f7g8h9

Note

The Account ID parameter must be the AWS account ID of the bucket owner, which is the shared user.

```
aws s3control create-access-point --account-id 111122223333 --name example-outpost-access-point \
--bucket arn:aws:s3-outposts:us-east-1:111122223333:outpost/op-01ac5d28a6a232904/bucket/DOC-EXAMPLE-BUCKET1 \
--vpc-configuration VpcId=vpc-1a2b3c4d5e6f7g8h9
```

For more information about this command, see [create-access-point](#) in the *AWS CLI Reference*.

Example : Upload an object

The following example uploads the file **my_image.jpg** from the user's local file system to an object named **images/my_image.jpg** through the access point **example-outpost-access-point** on the Outpost **op-01ac5d28a6a232904**, owned by the AWS account **111122223333**. Before using this command, replace these **user input placeholder** values and the AWS Region code with the appropriate values for your use case.

```
aws s3api put-object --bucket arn:aws:s3-outposts:us-east-1:111122223333:outpost/op-01ac5d28a6a232904/accesspoint/example-outpost-access-point \
--body my_image.jpg --key images/my_image.jpg
```

For more information about this command, see [put-object](#) in the *AWS CLI Reference*.

Note

If this operation results in a Resource not found error or is unresponsive, your VPC might not have a shared endpoint.

To check whether there is a shared endpoint, use the [list-shared-endpoints](#) AWS CLI command. If there is no shared endpoint, work with the Outpost owner to create one. For more information, see [ListSharedEndpoints](#) in the *Amazon Simple Storage Service API Reference*.

Example : Create an endpoint

The following example creates an endpoint on a shared Outpost. Before using this command, replace the **user input placeholder** values for the Outpost ID, subnet ID, and security group ID with the appropriate values for your use case.

Note

The user can perform this operation only if the resource share includes the **Outposts** resource.

```
aws s3outposts create-endpoint --outposts-id op-01ac5d28a6a232904 --subnet-id XXXXXX --security-group-id XXXXXX
```

For more information about this command, see [create-endpoint](#) in the *AWS CLI Reference*.

Other AWS services that use S3 on Outposts

Other AWS services that run local to your AWS Outposts can also use your Amazon S3 on Outposts capacity. In Amazon CloudWatch the `S3Outposts` namespace shows detailed metrics for buckets within

S3 on Outposts, but these metrics don't include usage for other AWS services. To manage your S3 on Outposts capacity that is consumed by other AWS services, see the information in the following table.

AWS service	Description	Learn more
Amazon S3	All direct S3 on Outposts usage has a matching account and bucket CloudWatch metric.	See metrics
Amazon Elastic Block Store (Amazon EBS)	For Amazon EBS on Outposts, you can choose an AWS Outpost as your snapshot destination and store locally in your S3 on Outpost.	Learn more
Amazon Relational Database Service (Amazon RDS)	You can use Amazon RDS local backups to store your RDS backups locally on your Outpost.	Learn more

Developing with Amazon S3 on Outposts

With Amazon S3 on Outposts, you can create S3 buckets on your AWS Outposts and easily store and retrieve objects on premises for applications that require local data access, local data processing, and data residency. S3 on Outposts provides a new storage class, S3 Outposts (`OUTPOSTS`), which uses the Amazon S3 APIs, and is designed to store data durably and redundantly across multiple devices and servers on your AWS Outposts. You communicate with your Outpost bucket by using an access point and endpoint connection over a virtual private cloud (VPC). You can use the same APIs and features on Outpost buckets as you do on Amazon S3 buckets, including access policies, encryption, and tagging. You can use S3 on Outposts through the AWS Management Console, AWS Command Line Interface (AWS CLI), AWS SDKs, or REST API. For more information, see [What is Amazon S3 on Outposts? \(p. 1241\)](#)

The following topics provide information about developing with S3 on Outposts.

Topics

- [Amazon S3 on Outposts API operations \(p. 1323\)](#)
- [Configure the S3 control client for S3 on Outposts by using the SDK for Java \(p. 1325\)](#)

Amazon S3 on Outposts API operations

This topic lists the Amazon S3, Amazon S3 Control, and Amazon S3 on Outposts API operations that you can use with Amazon S3 on Outposts.

Topics

- [Amazon S3 API operations for managing objects \(p. 1323\)](#)
- [Amazon S3 Control API operations for managing buckets \(p. 1324\)](#)
- [S3 on Outposts API operations for managing endpoints \(p. 1325\)](#)

Amazon S3 API operations for managing objects

S3 on Outposts is designed to use the same object API operations as Amazon S3. Therefore, you can use most of your existing code and many of your existing policies by passing the S3 on Outposts Amazon Resource Name (ARN) as your identifier.

Amazon S3 on Outposts supports the following Amazon S3 API operations:

- [AbortMultipartUpload](#)
- [CompleteMultipartUpload](#)
- [CopyObject](#)
- [CreateMultipartUpload](#)
- [DeleteObject](#)
- [DeleteObjects](#)
- [DeleteObjectTagging](#)
- [GetObject](#)
- [GetObjectTagging](#)
- [HeadBucket](#)
- [HeadObject](#)
- [ListMultipartUploads](#)
- [ListObjects](#)
- [ListObjectsV2](#)
- [ListParts](#)
- [PutObject](#)
- [PutObjectTagging](#)
- [UploadPart](#)
- [UploadPartCopy](#)

Amazon S3 Control API operations for managing buckets

S3 on Outposts supports the following Amazon S3 Control API operations for working with buckets.

- [CreateAccessPoint](#)
- [CreateBucket](#)
- [DeleteAccessPoint](#)
- [DeleteAccessPointPolicy](#)
- [DeleteBucket](#)
- [DeleteBucketLifecycleConfiguration](#)
- [DeleteBucketPolicy](#)
- [DeleteBucketTagging](#)
- [GetAccessPoint](#)
- [GetAccessPointPolicy](#)
- [GetBucket](#)
- [GetBucketLifecycleConfiguration](#)
- [GetBucketPolicy](#)
- [GetBucketTagging](#)
- [ListAccessPoints](#)
- [ListRegionalBuckets](#)
- [PutAccessPointPolicy](#)
- [PutBucketLifecycleConfiguration](#)
- [PutBucketPolicy](#)
- [PutBucketTagging](#)

S3 on Outposts API operations for managing endpoints

S3 on Outposts supports the following Amazon S3 on Outposts API operations for managing endpoints.

- [CreateEndpoint](#)
- [DeleteEndpoint](#)
- [ListEndpoints](#)
- [ListSharedEndpoints](#)

Configure the S3 control client for S3 on Outposts by using the SDK for Java

The following example configures the Amazon S3 control client for Amazon S3 on Outposts by using the AWS SDK for Java. To use this example, replace each *user input placeholder* with your own information.

```
import com.amazonaws.auth.AWSStaticCredentialsProvider;
import com.amazonaws.auth.BasicAWSCredentials;
import com.amazonaws.services.s3control.AWSS3Control;
import com.amazonaws.services.s3control.AWSS3ControlClient;

public AWSS3Control createS3ControlClient() {

    String accessKey = AWSAccessKey;
    String secretKey = SecretAccessKey;
    BasicAWSCredentials awsCreds = new BasicAWSCredentials(accessKey, secretKey);

    return AWSS3ControlClient.builder().enableUseArnRegion()
        .withCredentials(new AWSStaticCredentialsProvider(awsCreds))
        .build();
}
```

Code examples for Amazon S3 using AWS SDKs

The following code examples show how to use Amazon S3 with an AWS software development kit (SDK).

The examples are divided into the following categories:

Actions

Code excerpts that show you how to call individual service functions.

Scenarios

Code examples that show you how to accomplish a specific task by calling multiple functions within the same service.

Cross-service examples

Sample applications that work across multiple AWS services.

For a complete list of AWS SDK developer guides and code examples, see [Using this service with an AWS SDK \(p. 1185\)](#). This topic also includes information about getting started and details about previous SDK versions.

Code examples

- [Actions for Amazon S3 using AWS SDKs \(p. 1327\)](#)

- Add CORS rules to an Amazon S3 bucket using an AWS SDK (p. 1328)
- Add a lifecycle configuration to an Amazon S3 bucket using an AWS SDK (p. 1332)
- Add a policy to an Amazon S3 bucket using an AWS SDK (p. 1336)
- Copy an object from one Amazon S3 bucket to another using an AWS SDK (p. 1341)
- Create an Amazon S3 bucket using an AWS SDK (p. 1349)
- Delete CORS rules from an Amazon S3 bucket using an AWS SDK (p. 1357)
- Delete a policy from an Amazon S3 bucket using an AWS SDK (p. 1358)
- Delete an empty Amazon S3 bucket using an AWS SDK (p. 1362)
- Delete an Amazon S3 object using an AWS SDK (p. 1367)
- Delete multiple objects from an Amazon S3 bucket using an AWS SDK (p. 1373)
- Delete the lifecycle configuration of an Amazon S3 bucket using an AWS SDK (p. 1381)
- Delete the website configuration from an Amazon S3 bucket using an AWS SDK (p. 1382)
- Determine the existence and content type of an object in an Amazon S3 bucket using an AWS SDK (p. 1384)
- Determine the existence of an Amazon S3 bucket using an AWS SDK (p. 1385)
- Get CORS rules for an Amazon S3 bucket using an AWS SDK (p. 1386)
- Get an object from an Amazon S3 bucket using an AWS SDK (p. 1388)
- Get the ACL of an Amazon S3 bucket using an AWS SDK (p. 1398)
- Get the ACL of an Amazon S3 object using an AWS SDK (p. 1401)
- Get the Region where the Amazon S3 bucket resides using an AWS SDK (p. 1405)
- Get the lifecycle configuration of an Amazon S3 bucket using an AWS SDK (p. 1406)

- Get the policy for an Amazon S3 bucket using an AWS SDK (p. 1407)
- Get the website configuration for an Amazon S3 bucket using an AWS SDK (p. 1411)
- List Amazon S3 buckets using an AWS SDK (p. 1413)
- List in-progress multipart uploads to an Amazon S3 bucket using an AWS SDK (p. 1417)
- List the version of objects in an Amazon S3 bucket using an AWS SDK (p. 1418)
- List objects in an Amazon S3 bucket using an AWS SDK (p. 1419)
- Restore an archived copy of an object back into an Amazon S3 bucket using an AWS SDK (p. 1427)
- Set a new ACL for an Amazon S3 bucket using an AWS SDK (p. 1428)
- Set the ACL of an Amazon S3 object using an AWS SDK (p. 1431)
- Set the website configuration for an Amazon S3 bucket using an AWS SDK (p. 1432)
- Upload an object to an Amazon S3 bucket using an AWS SDK (p. 1436)
- Scenarios for Amazon S3 using AWS SDKs (p. 1449)
 - Create a presigned URL for Amazon S3 using an AWS SDK (p. 1450)
 - Getting started with Amazon S3 buckets and objects using an AWS SDK (p. 1457)
 - Manage versioned Amazon S3 objects in batches with a Lambda function using an AWS SDK (p. 1492)
 - Use a transfer manager to upload and download files to and from Amazon S3 using an AWS SDK (p. 1492)
 - Work with Amazon S3 versioned objects using an AWS SDK (p. 1506)
- Cross-service examples for Amazon S3 using AWS SDKs (p. 1510)
 - Build an Amazon Transcribe app (p. 1510)
 - Convert text to speech and back to text using an AWS SDK (p. 1511)
 - Create a long-lived Amazon EMR cluster and run several steps using an AWS SDK (p. 1512)
 - Create a short-lived Amazon EMR cluster and run a step using an AWS SDK (p. 1512)
 - Create an Amazon Textract explorer application (p. 1513)
 - Detect PPE in images with Amazon Rekognition using an AWS SDK (p. 1514)
 - Detect entities in text extracted from an image using an AWS SDK (p. 1515)
 - Detect faces in an image using an AWS SDK (p. 1515)
 - Detect objects in images with Amazon Rekognition using an AWS SDK (p. 1516)
 - Detect people and objects in a video with Amazon Rekognition using an AWS SDK (p. 1518)
 - Save EXIF and other image information using an AWS SDK (p. 1519)

Actions for Amazon S3 using AWS SDKs

The following code examples demonstrate how to perform individual Amazon S3 actions with AWS SDKs. These excerpts call the Amazon S3 API and are not intended to be run in isolation. Each example includes a link to GitHub, where you can find instructions on how to set up and run the code in context.

The following examples include only the most commonly used actions. For a complete list, see the [Amazon Simple Storage Service API Reference](#).

Examples

- Add CORS rules to an Amazon S3 bucket using an AWS SDK (p. 1328)
- Add a lifecycle configuration to an Amazon S3 bucket using an AWS SDK (p. 1332)
- Add a policy to an Amazon S3 bucket using an AWS SDK (p. 1336)
- Copy an object from one Amazon S3 bucket to another using an AWS SDK (p. 1341)

- [Create an Amazon S3 bucket using an AWS SDK \(p. 1349\)](#)
- [Delete CORS rules from an Amazon S3 bucket using an AWS SDK \(p. 1357\)](#)
- [Delete a policy from an Amazon S3 bucket using an AWS SDK \(p. 1358\)](#)
- [Delete an empty Amazon S3 bucket using an AWS SDK \(p. 1362\)](#)
- [Delete an Amazon S3 object using an AWS SDK \(p. 1367\)](#)
- [Delete multiple objects from an Amazon S3 bucket using an AWS SDK \(p. 1373\)](#)
- [Delete the lifecycle configuration of an Amazon S3 bucket using an AWS SDK \(p. 1381\)](#)
- [Delete the website configuration from an Amazon S3 bucket using an AWS SDK \(p. 1382\)](#)
- [Determine the existence and content type of an object in an Amazon S3 bucket using an AWS SDK \(p. 1384\)](#)
- [Determine the existence of an Amazon S3 bucket using an AWS SDK \(p. 1385\)](#)
- [Get CORS rules for an Amazon S3 bucket using an AWS SDK \(p. 1386\)](#)
- [Get an object from an Amazon S3 bucket using an AWS SDK \(p. 1388\)](#)
- [Get the ACL of an Amazon S3 bucket using an AWS SDK \(p. 1398\)](#)
- [Get the ACL of an Amazon S3 object using an AWS SDK \(p. 1401\)](#)
- [Get the Region where the Amazon S3 bucket resides using an AWS SDK \(p. 1405\)](#)
- [Get the lifecycle configuration of an Amazon S3 bucket using an AWS SDK \(p. 1406\)](#)
- [Get the policy for an Amazon S3 bucket using an AWS SDK \(p. 1407\)](#)
- [Get the website configuration for an Amazon S3 bucket using an AWS SDK \(p. 1411\)](#)
- [List Amazon S3 buckets using an AWS SDK \(p. 1413\)](#)
- [List in-progress multipart uploads to an Amazon S3 bucket using an AWS SDK \(p. 1417\)](#)
- [List the version of objects in an Amazon S3 bucket using an AWS SDK \(p. 1418\)](#)
- [List objects in an Amazon S3 bucket using an AWS SDK \(p. 1419\)](#)
- [Restore an archived copy of an object back into an Amazon S3 bucket using an AWS SDK \(p. 1427\)](#)
- [Set a new ACL for an Amazon S3 bucket using an AWS SDK \(p. 1428\)](#)
- [Set the ACL of an Amazon S3 object using an AWS SDK \(p. 1431\)](#)
- [Set the website configuration for an Amazon S3 bucket using an AWS SDK \(p. 1432\)](#)
- [Upload an object to an Amazon S3 bucket using an AWS SDK \(p. 1436\)](#)

Add CORS rules to an Amazon S3 bucket using an AWS SDK

The following code examples show how to add cross-origin resource sharing (CORS) rules to an S3 bucket.

Java

SDK for Java 2.x

Tip

To learn how to set up and run this example, see [GitHub](#).

```
public static void deleteBucketCorsInformation(S3Client s3, String bucketName,
String accountId) {
    try {
        DeleteBucketCorsRequest bucketCorsRequest =
DeleteBucketCorsRequest.builder()
```

```
.bucket(bucketName)
.expectedBucketOwner(accountId)
.build();

s3.deleteBucketCors(bucketCorsRequest) ;

} catch (S3Exception e) {
    System.err.println(e.awsErrorDetails().errorMessage());
    System.exit(1);
}

public static void getBucketCorsInformation(S3Client s3, String bucketName,
String accountId) {

    try {
        GetBucketCorsRequest bucketCorsRequest = GetBucketCorsRequest.builder()
            .bucket(bucketName)
            .expectedBucketOwner(accountId)
            .build();

        GetBucketCorsResponse corsResponse =
s3.getBucketCors(bucketCorsRequest);
        List<CORSRule> corsRules = corsResponse.corsRules();
        for (CORSRule rule: corsRules) {
            System.out.println("allowOrigins: "+rule.allowedOrigins());
            System.out.println("AllowedMethod: "+rule.allowedMethods());
        }
    } catch (S3Exception e) {

        System.err.println(e.awsErrorDetails().errorMessage());
        System.exit(1);
    }
}

public static void setCorsInformation(S3Client s3, String bucketName, String
accountId) {

    List<String> allowMethods = new ArrayList<>();
    allowMethods.add("PUT");
    allowMethods.add("POST");
    allowMethods.add("DELETE");

    List<String> allowOrigins = new ArrayList<>();
    allowOrigins.add("http://example.com");
    try {
        // Define CORS rules.
        CORSRule corsRule = CORSRule.builder()
            .allowedMethods(allowMethods)
            .allowedOrigins(allowOrigins)
            .build();

        List<CORSRule> corsRules = new ArrayList<>();
        corsRules.add(corsRule);
        CORSConfiguration configuration = CORSConfiguration.builder()
            .corsRules(corsRules)
            .build();

        PutBucketCorsRequest putBucketCorsRequest =
PutBucketCorsRequest.builder()
            .bucket(bucketName)
            .corsConfiguration(configuration)
            .expectedBucketOwner(accountId)
            .build();
    }
}
```

```
        s3.putBucketCors(putBucketCorsRequest);

    } catch (S3Exception e) {
        System.err.println(e.awsErrorDetails().errorMessage());
        System.exit(1);
    }
}
```

- For API details, see [PutBucketCors](#) in *AWS SDK for Java 2.x API Reference*.

JavaScript

SDK for JavaScript V3

Tip

To learn how to set up and run this example, see [GitHub](#).

Create the client.

```
// Create service client module using ES6 syntax.
import { S3Client } from "@aws-sdk/client-s3";
// Set the AWS Region.
const REGION = "us-east-1";
// Create an Amazon S3 service client object.
const s3Client = new S3Client({ region: REGION });
export { s3Client };
```

Add a CORS rule.

```
// Import required AWS-SDK clients and commands for Node.js.
import { PutBucketCorsCommand } from "@aws-sdk/client-s3";
import { s3Client } from "./libs/s3Client.js"; // Helper function that creates an
Amazon S3 service client module.

// Set parameters.
// Create initial parameters JSON for putBucketCors.
const thisConfig = {
    AllowedHeaders: ["Authorization"],
    AllowedMethods: [],
    AllowedOrigins: ["*"],
    ExposeHeaders: [],
    MaxAgeSeconds: 3000,
};

// Assemble the list of allowed methods based on command line parameters
const allowedMethods = [];
process.argv.forEach(function (val, index, array) {
    if (val.toUpperCase() === "POST") {
        allowedMethods.push("POST");
    }
    if (val.toUpperCase() === "GET") {
        allowedMethods.push("GET");
    }
    if (val.toUpperCase() === "PUT") {
        allowedMethods.push("PUT");
    }
    if (val.toUpperCase() === "PATCH") {
        allowedMethods.push("PATCH");
    }
    if (val.toUpperCase() === "DELETE") {
```

```
        allowedMethods.push("DELETE");
    }
    if (val.toUpperCase() === "HEAD") {
        allowedMethods.push("HEAD");
    }
});

// Copy the array of allowed methods into the config object
thisConfig.AllowedImageMethods = allowedMethods;

// Create an array of configs then add the config object to it.
const corsRules = new Array(thisConfig);

// Create CORS parameters.
export const corsParams = {
    Bucket: "BUCKET_NAME",
    CORSConfiguration: { CORSRules: corsRules },
};

export async function run() {
    try {
        const data = await s3Client.send(new PutBucketCorsCommand(corsParams));
        console.log("Success", data);
        return data; // For unit tests.
    } catch (err) {
        console.log("Error", err);
    }
}
run();
```

- For more information, see [AWS SDK for JavaScript Developer Guide](#).
- For API details, see [PutBucketCors](#) in [AWS SDK for JavaScript API Reference](#).

Python

SDK for Python (Boto3)

Tip

To learn how to set up and run this example, see [GitHub](#).

```
class BucketWrapper:
    def __init__(self, bucket):
        self.bucket = bucket
        self.name = bucket.name

    def put_cors(self, cors_rules):
        """
        Apply CORS rules to the bucket. CORS rules specify the HTTP actions that
        are
        allowed from other domains.

        :param cors_rules: The CORS rules to apply.
        """
        try:
            self.bucket.Cors().put(CORSConfiguration={'CORSRules': cors_rules})
            logger.info(
                "Put CORS rules %s for bucket '%s'.", cors_rules, self.bucket.name)
        except ClientError:
            logger.exception("Couldn't put CORS rules for bucket %s.",
                            self.bucket.name)
            raise
```

- For API details, see [PutBucketCors](#) in *AWS SDK for Python (Boto3) API Reference*.

Ruby

SDK for Ruby

Tip

To learn how to set up and run this example, see [GitHub](#).

```
require "aws-sdk-s3"

# Wraps Amazon S3 bucket CORS configuration.
class BucketCorsWrapper
    attr_reader :bucket_cors

    # @param bucket_cors [Aws::S3::BucketCors] A bucket CORS object configured with
    # an existing bucket.
    def initialize(bucket_cors)
        @bucket_cors = bucket_cors
    end

    # Sets CORS rules on a bucket.
    #
    # @param allowed_methods [Array<String>] The types of HTTP requests to allow.
    # @param allowed_origins [Array<String>] The origins to allow.
    # @returns [Boolean] True if the CORS rules were set; otherwise, false.
    def set_cors(allowed_methods, allowed_origins)
        @bucket_cors.put(
            cors_configuration: {
                cors_rules: [
                    {
                        allowed_methods: allowed_methods,
                        allowed_origins: allowed_origins,
                        allowed_headers: "%w[*]",
                        max_age_seconds: 3600
                    }
                ]
            }
        )
        true
    rescue Aws::Errors::ServiceError => e
        puts "Couldn't set CORS rules for #{@bucket_cors.bucket.name}. Here's why:
#{e.message}"
        false
    end
end
```

- For API details, see [PutBucketCors](#) in *AWS SDK for Ruby API Reference*.

For a complete list of AWS SDK developer guides and code examples, see [Using this service with an AWS SDK \(p. 1185\)](#). This topic also includes information about getting started and details about previous SDK versions.

Add a lifecycle configuration to an Amazon S3 bucket using an AWS SDK

The following code examples show how to add a lifecycle configuration to an S3 bucket.

Java

SDK for Java 2.x

Tip

To learn how to set up and run this example, see [GitHub](#).

```
public static void setLifecycleConfig(S3Client s3, String bucketName, String
accountID) {

    try {
        // Create a rule to archive objects with the "glacierobjects/" prefix
        // to Amazon S3 Glacier.
        LifecycleRuleFilter ruleFilter = LifecycleRuleFilter.builder()
            .prefix("glacierobjects/")
            .build();

        Transition transition = Transition.builder()
            .storageClass(TransitionStorageClass.GLACIER)
            .days(0)
            .build();

        LifecycleRule rule1 = LifecycleRule.builder()
            .id("Archive immediately rule")
            .filter(ruleFilter)
            .transitions(transition)
            .status(ExpirationStatus.ENABLED)
            .build();

        // Create a second rule.
        Transition transition2 = Transition.builder()
            .storageClass(TransitionStorageClass.GLACIER)
            .days(0)
            .build();

        List<Transition> transitionList = new ArrayList<>();
        transitionList.add(transition2);

        LifecycleRuleFilter ruleFilter2 = LifecycleRuleFilter.builder()
            .prefix("glacierobjects/")
            .build();

        LifecycleRule rule2 = LifecycleRule.builder()
            .id("Archive and then delete rule")
            .filter(ruleFilter2)
            .transitions(transitionList)
            .status(ExpirationStatus.ENABLED)
            .build();

        // Add the LifecycleRule objects to an ArrayList.
        ArrayList<LifecycleRule> ruleList = new ArrayList<>();
        ruleList.add(rule1);
        ruleList.add(rule2);

        BucketLifecycleConfiguration lifecycleConfiguration =
        BucketLifecycleConfiguration.builder()
            .rules(ruleList)
            .build();

        PutBucketLifecycleConfigurationRequest
putBucketLifecycleConfigurationRequest =
PutBucketLifecycleConfigurationRequest.builder()
            .bucket(bucketName)
            .lifecycleConfiguration(lifecycleConfiguration)
```

```
        .expectedBucketOwner(accountId)
        .build();

s3.putBucketLifecycleConfiguration(putBucketLifecycleConfigurationRequest);

    } catch (S3Exception e) {
        System.err.println(e.awsErrorDetails().errorMessage());
        System.exit(1);
    }
}

// Retrieve the configuration and add a new rule.
public static void getLifecycleConfig(S3Client s3, String bucketName, String
accountID){

    try {
        GetBucketLifecycleConfigurationRequest
getBucketLifecycleConfigurationRequest =
GetBucketLifecycleConfigurationRequest.builder()
        .bucket(bucketName)
        .expectedBucketOwner(accountId)
        .build();

        GetBucketLifecycleConfigurationResponse response =
s3.getBucketLifecycleConfiguration(getBucketLifecycleConfigurationRequest);
        List<LifecycleRule> newList = new ArrayList<>();
        List<LifecycleRule> rules = response.rules();
        for (LifecycleRule rule: rules) {
            newList.add(rule);
        }

        // Add a new rule with both a prefix predicate and a tag predicate.
        LifecycleRuleFilter ruleFilter = LifecycleRuleFilter.builder()
            .prefix("YearlyDocuments/")
            .build();

        Transition transition = Transition.builder()
            .storageClass(TransitionStorageClass.GLACIER)
            .days(3650)
            .build();

        LifecycleRule rule1 = LifecycleRule.builder()
            .id("NewRule")
            .filter(ruleFilter)
            .transitions(transition)
            .status(ExpirationStatus.ENABLED)
            .build();

        // Add the new rule to the list.
        newList.add(rule1);
        BucketLifecycleConfiguration lifecycleConfiguration =
BucketLifecycleConfiguration.builder()
            .rules(newList)
            .build();

        PutBucketLifecycleConfigurationRequest
putBucketLifecycleConfigurationRequest =
PutBucketLifecycleConfigurationRequest.builder()
        .bucket(bucketName)
        .lifecycleConfiguration(lifecycleConfiguration)
        .expectedBucketOwner(accountId)
        .build();

s3.putBucketLifecycleConfiguration(putBucketLifecycleConfigurationRequest);
```

```
        } catch (S3Exception e) {
            System.err.println(e.awsErrorDetails().errorMessage());
            System.exit(1);
        }
    }

    // Delete the configuration from the Amazon S3 bucket.
    public static void deleteLifecycleConfig(S3Client s3, String bucketName, String
accountID) {

        try {
            DeleteBucketLifecycleRequest deleteBucketLifecycleRequest =
DeleteBucketLifecycleRequest.builder()
                .bucket(bucketName)
                .expectedBucketOwner(accountID)
                .build();

            s3.deleteBucketLifecycle(deleteBucketLifecycleRequest);

        } catch (S3Exception e) {
            System.err.println(e.awsErrorDetails().errorMessage());
            System.exit(1);
        }
    }
}
```

- For API details, see [PutBucketLifecycleConfiguration](#) in *AWS SDK for Java 2.x API Reference*.

Python

SDK for Python (Boto3)

Tip

To learn how to set up and run this example, see [GitHub](#).

```
class BucketWrapper:
    def __init__(self, bucket):
        self.bucket = bucket
        self.name = bucket.name

    def put_lifecycle_configuration(self, lifecycle_rules):
        """
        Apply a lifecycle configuration to the bucket. The lifecycle configuration
        can
            be used to archive or delete the objects in the bucket according to
        specified
            parameters, such as a number of days.

        :param lifecycle_rules: The lifecycle rules to apply.
        """
        try:
            self.bucket.LifecycleConfiguration().put(
                LifecycleConfiguration={'Rules': lifecycle_rules})
            logger.info(
                "Put lifecycle rules %s for bucket '%s'.", lifecycle_rules,
                self.bucket.name)
        except ClientError:
            logger.exception(
                "Couldn't put lifecycle rules for bucket '%s'.", self.bucket.name)
            raise
```

- For API details, see [PutBucketLifecycleConfiguration](#) in *AWS SDK for Python (Boto3) API Reference*.

For a complete list of AWS SDK developer guides and code examples, see [Using this service with an AWS SDK \(p. 1185\)](#). This topic also includes information about getting started and details about previous SDK versions.

Add a policy to an Amazon S3 bucket using an AWS SDK

The following code examples show how to add a policy to an S3 bucket.

C++

SDK for C++

Tip

To learn how to set up and run this example, see [GitHub](#).

```
bool AwsDoc::S3::PutBucketPolicy(const Aws::String& bucketName,
                                  const Aws::String& policyBody,
                                  const Aws::String& region)
{
    Aws::Client::ClientConfiguration config;

    if (!region.empty())
    {
        config.region = region;
    }

    Aws::S3::S3Client s3_client(config);

    std::shared_ptr<Aws::StringStream> request_body =
        Aws::MakeShared<Aws::StringStream>("");
    *request_body << policyBody;

    Aws::S3::Model::PutBucketPolicyRequest request;
    request.SetBucket(bucketName);
    request.SetBody(request_body);

    Aws::S3::Model::PutBucketPolicyOutcome outcome =
        s3_client.PutBucketPolicy(request);

    if (outcome.IsSuccess()) {
        std::cout << "Set the following policy body for the bucket '" <<
            bucketName << ":" << std::endl << std::endl;
        std::cout << policyBody << std::endl;

        return true;
    }
    else {
        std::cout << "Error: PutBucketPolicy: "
              << outcome.GetError().GetMessage() << std::endl;

        return false;
    }
}

int main()
{
```

```
Aws::SDKOptions options;
Aws::InitAPI(options);
{
    //TODO: Change bucket_name to the name of a bucket in your account.
    const Aws::String bucket_name = "DOC-EXAMPLE-BUCKET";
    //TODO: Set to the AWS Region in which the bucket was created.
    const Aws::String region = "us-east-1";

    // Get the caller's AWS account ID to be used in the bucket policy.
    Aws::STS::STSService sts_client;
    Aws::STS::Model::GetCallerIdentityRequest request;
    Aws::STS::Model::GetCallerIdentityOutcome outcome =
        sts_client.GetCallerIdentity(request);

    if (!outcome.IsSuccess())
    {
        std::cout << "Error: GetBucketPolicy setup: Get identity information:
"
        << outcome.GetError().GetMessage() << std::endl;

        return 1;
    }

    // Extract the caller's AWS account ID from the call to AWS STS.
    Aws::String account_id = outcome.GetResult().GetAccount();

    // Use the account ID and bucket name to form the bucket policy to be
    added.
    Aws::String policy_string = GetPolicyString(account_id, bucket_name);

    if (!AwsDoc::S3::PutBucketPolicy(bucket_name, policy_string, region))
    {
        return 1;
    }
}
Aws::ShutdownAPI(options);

return 0;
}
```

- For API details, see [PutBucketPolicy](#) in *AWS SDK for C++ API Reference*.

Java

SDK for Java 2.x

Tip

To learn how to set up and run this example, see [GitHub](#).

```
public static void setPolicy(S3Client s3, String bucketName, String policyText)
{
    System.out.println("Setting policy:");
    System.out.println("----");
    System.out.println(policyText);
    System.out.println("----");
    System.out.format("On Amazon S3 bucket: \"%s\"\n", bucketName);

    try {
        PutBucketPolicyRequest policyReq = PutBucketPolicyRequest.builder()
            .bucket(bucketName)
```

```
.policy(policyText)
.build();

s3.putBucketPolicy(policyReq);

} catch (S3Exception e) {
    System.err.println(e.awsErrorDetails().errorMessage());
    System.exit(1);
}

System.out.println("Done!");
}

// Loads a JSON-formatted policy from a file
public static String getBucketPolicyFromFile(String policyFile) {

    StringBuilder fileText = new StringBuilder();
    try {
        List<String> lines = Files.readAllLines(Paths.get(policyFile),
StandardCharsets.UTF_8);
        for (String line : lines) {
            fileText.append(line);
        }
    } catch (IOException e) {
        System.out.format("Problem reading file: \"%s\"", policyFile);
        System.out.println(e.getMessage());
    }

    try {
        final JsonParser parser = new
ObjectMapper().getFactory().createParser(fileText.toString());
        while (parser.nextToken() != null) {
        }

    } catch (IOException jpe) {
        jpe.printStackTrace();
    }
    return fileText.toString();
}
```

- For API details, see [PutBucketPolicy](#) in *AWS SDK for Java 2.x API Reference*.

JavaScript

SDK for JavaScript V3

Tip

To learn how to set up and run this example, see [GitHub](#).

Create the client.

```
// Create service client module using ES6 syntax.
import { S3Client } from "@aws-sdk/client-s3";
// Set the AWS Region.
const REGION = "us-east-1";
// Create an Amazon S3 service client object.
const s3Client = new S3Client({ region: REGION });
export { s3Client };
```

Add the policy.

```
// Import required AWS SDK clients and commands for Node.js.
import { CreateBucketCommand, PutBucketPolicyCommand } from "@aws-sdk/client-s3";
import { s3Client } from "./libs/s3Client.js"; // Helper function that creates an
// Amazon S3 service client module.

const BUCKET_NAME = "BUCKET_NAME";
export const bucketParams = {
    Bucket: BUCKET_NAME,
};

// Create the policy in JSON for the S3 bucket.
const readOnlyAnonUserPolicy = {
    Version: "2012-10-17",
    Statement: [
        {
            Sid: "AddPerm",
            Effect: "Allow",
            Principal: "*",
            Action: ["s3:GetObject"],
            Resource: ["*"],
        },
    ],
};

// Create selected bucket resource string for bucket policy.
const bucketResource = "arn:aws:s3:::" + BUCKET_NAME + "/*"; //BUCKET_NAME
readOnlyAnonUserPolicy.Statement[0].Resource[0] = bucketResource;

// Convert policy JSON into string and assign into parameters.
const bucketPolicyParams = {
    Bucket: BUCKET_NAME,
    Policy: JSON.stringify(readOnlyAnonUserPolicy),
};

export const run = async () => {
    try {
        const data = await s3Client.send(
            new CreateBucketCommand(bucketParams)
        );
        console.log('Success, bucket created.', data)
        try {
            const response = await s3Client.send(
                new PutBucketPolicyCommand(bucketPolicyParams)
            );
            console.log("Success, permissions added to bucket", response);
            return response;
        }
        catch (err) {
            console.log("Error adding policy to S3 bucket.", err);
        }
    } catch (err) {
        console.log("Error creating S3 bucket.", err);
    }
};

run();
```

- For more information, see [AWS SDK for JavaScript Developer Guide](#).
- For API details, see [PutBucketPolicy](#) in [AWS SDK for JavaScript API Reference](#).

Python

SDK for Python (Boto3)

Tip

To learn how to set up and run this example, see [GitHub](#).

```
class BucketWrapper:
    def __init__(self, bucket):
        self.bucket = bucket
        self.name = bucket.name

    def put_policy(self, policy):
        """
        Apply a security policy to the bucket. Policies control users' ability
        to perform specific actions, such as listing the objects in the bucket.

        :param policy: The policy to apply to the bucket.
        """
        try:
            self.bucket.Policy().put(Policy=json.dumps(policy))
            logger.info("Put policy %s for bucket '%s'.", policy, self.bucket.name)
        except ClientError:
            logger.exception("Couldn't apply policy to bucket '%s'.",
                             self.bucket.name)
            raise
```

- For API details, see [PutBucketPolicy](#) in *AWS SDK for Python (Boto3) API Reference*.

Ruby

SDK for Ruby

Tip

To learn how to set up and run this example, see [GitHub](#).

```
# Wraps an Amazon S3 bucket policy.
class BucketPolicyWrapper
  attr_reader :bucket_policy

  # @param bucket_policy [Aws::S3::BucketPolicy] A bucket policy object configured
  # with an existing bucket.
  def initialize(bucket_policy)
    @bucket_policy = bucket_policy
  end

  # Sets a policy on a bucket.
  #
  # @param policy [String]
  # @return [Boolean]
  def set_policy(policy)
    @bucket_policy.put(policy: policy)
    true
  rescue Aws::Errors::ServiceError => e
    puts "Couldn't set the policy for #{@bucket_policy.bucket.name}. Here's why:
#{e.message}"
    false
  end
end
```

- For API details, see [PutBucketPolicy](#) in *AWS SDK for Ruby API Reference*.

For a complete list of AWS SDK developer guides and code examples, see [Using this service with an AWS SDK \(p. 1185\)](#). This topic also includes information about getting started and details about previous SDK versions.

Copy an object from one Amazon S3 bucket to another using an AWS SDK

The following code examples show how to copy an S3 object from one bucket to another.

.NET

AWS SDK for .NET

Tip

To learn how to set up and run this example, see [GitHub](#).

```
/// <summary>
/// Copies an object in an Amazon S3 bucket to a folder within the
/// same bucket.
/// </summary>
/// <param name="client">An initialized Amazon S3 client object.</param>
/// <param name="bucketName">The name of the Amazon S3 bucket where the
/// object to copy is located.</param>
/// <param name="objectName">The object to be copied.</param>
/// <param name="folderName">The folder to which the object will
/// be copied.</param>
/// <returns>A boolean value that indicates the success or failure of
/// the copy operation.</returns>
public static async Task<bool> CopyObjectInBucketAsync(
    IAmazonS3 client,
    string bucketName,
    string objectName,
    string folderName)
{
    try
    {
        var request = new CopyObjectRequest
        {
            SourceBucket = bucketName,
            SourceKey = objectName,
            DestinationBucket = bucketName,
            DestinationKey = $"{folderName}\\{objectName}",
        };
        var response = await client.CopyObjectAsync(request);
        return response.HttpStatusCode == System.Net.HttpStatusCode.OK;
    }
    catch (AmazonS3Exception ex)
    {
        Console.WriteLine($"Error copying object: '{ex.Message}'");
        return false;
    }
}
```

- For API details, see [CopyObject](#) in *AWS SDK for .NET API Reference*.

C++

SDK for C++

Tip

To learn how to set up and run this example, see [GitHub](#).

```
bool AwsDoc::S3::CopyObject(const Aws::String &objectKey, const Aws::String
&fromBucket, const Aws::String &toBucket,
                             const Aws::String &region) {
    Aws::Client::ClientConfiguration clientConfig;
    if (!region.empty()) {
        clientConfig.region = region;
    }

    Aws::S3::S3Client client(clientConfig);
    Aws::S3::Model::CopyObjectRequest request;

    request.WithCopySource(fromBucket + "/" + objectKey)
        .WithKey(objectKey)
        .WithBucket(toBucket);

    Aws::S3::Model::CopyObjectOutcome outcome = client.CopyObject(request);

    if (!outcome.IsSuccess())
    {
        const auto err = outcome.GetError();
        std::cout << "Error: CopyObject: " <<
                    err.GetExceptionName() << ":" << err.GetMessage() << std::endl;
        return false;
    }

    return true;
}

int main() {
    Aws::SDKOptions options;
    Aws::InitAPI(options);

    //TODO: Name of object already in bucket.
    Aws::String objectKey = "<enter object key>";

    //TODO: Change from_bucket to the name of your bucket that already contains
    //my-file.txt".
    Aws::String fromBucket = "<Enter bucket name>";

    //TODO: Change to the name of another bucket in your account.
    Aws::String toBucket = "<Enter bucket name>";

    //TODO: Set to the AWS Region in which the bucket was created.
    Aws::String region = "us-east-1";

    AwsDoc::S3::CopyObject(objectKey, fromBucket, toBucket, region);

    ShutdownAPI(options);
    return 0;
}
```

- For API details, see [CopyObject in AWS SDK for C++ API Reference](#).

Go

SDK for Go V2

Tip

To learn how to set up and run this example, see [GitHub](#).

```
// Copy an object to another name.

// CopyObject is "Pull an object from the source bucket + path".
// The semantics of CopySource varies depending on whether you're using Amazon S3
// on Outposts,
// or through access points.
// See https://docs.aws.amazon.com/AmazonS3/latest/API/
API_CopyObject.html#API_CopyObject_RequestSyntax
fmt.Println("Copy an object from another bucket to our bucket.")
_, err = client.CopyObject(context.TODO(), &s3.CopyObjectInput{
    Bucket:     aws.String(name),
    CopySource: aws.String(name + "/path/myfile.jpg"),
    Key:        aws.String("other/file.jpg"),
})

if err != nil {
    panic("Couldn't copy the object to a new key")
}
```

- For API details, see [CopyObject in AWS SDK for Go API Reference](#).

Java

SDK for Java 2.x

Tip

To learn how to set up and run this example, see [GitHub](#).

```
public static String copyBucketObject (S3Client s3, String fromBucket, String
objectKey, String toBucket) {

    String encodedUrl = "";
    try {
        encodedUrl = URLEncoder.encode(fromBucket + "/" + objectKey,
StandardCharsets.UTF_8.toString());

    } catch (UnsupportedEncodingException e) {
        System.out.println("URL could not be encoded: " + e.getMessage());
    }

    CopyObjectRequest copyReq = CopyObjectRequest.builder()
        .copySource(encodedUrl)
        .destinationBucket(toBucket)
        .destinationKey(objectKey)
        .build();

    try {
        CopyObjectResponse copyRes = s3.copyObject(copyReq);
        return copyRes.copyObjectResult().toString();
    } catch (S3Exception e) {
        System.err.println(e.awsErrorDetails().errorMessage());
        System.exit(1);
    }
}
```

```
        }
    return "";
}
```

- For API details, see [CopyObject in AWS SDK for Java 2.x API Reference](#).

JavaScript

SDK for JavaScript V3

Tip

To learn how to set up and run this example, see [GitHub](#).

Create the client.

```
// Create service client module using ES6 syntax.
import { S3Client } from "@aws-sdk/client-s3";
// Set the AWS Region.
const REGION = "us-east-1";
// Create an Amazon S3 service client object.
const s3Client = new S3Client({ region: REGION });
export { s3Client };
```

Copy the object.

```
// Get service clients module and commands using ES6 syntax.
import { CopyObjectCommand } from "@aws-sdk/client-s3";
import { s3Client } from "./libs/s3Client.js";

// Set the bucket parameters.

export const params = {
    Bucket: "DESTINATION_BUCKET_NAME",
    CopySource: "/SOURCE_BUCKET_NAME/OBJECT_NAME",
    Key: "OBJECT_NAME"
};

// Create the Amazon S3 bucket.
export const run = async () => {
    try {
        const data = await s3Client.send(new CopyObjectCommand(params));
        console.log("Success", data);
        return data; // For unit tests.
    } catch (err) {
        console.log("Error", err);
    }
};
run();
```

- For API details, see [CopyObject in AWS SDK for JavaScript API Reference](#).

Kotlin

SDK for Kotlin

Note

This is prerelease documentation for a feature in preview release. It is subject to change.

Tip

To learn how to set up and run this example, see [GitHub](#).

```
suspend fun copyBucketObject(  
    fromBucket: String,  
    objectKey: String,  
    toBucket: String  
) {  
  
    var encodedUrl = ""  
    try {  
        encodedUrl = URLEncoder.encode("$fromBucket/$objectKey",  
StandardCharsets.UTF_8.toString())  
    } catch (e: UnsupportedEncodingException) {  
        println("URL could not be encoded: " + e.message)  
    }  
  
    val request = CopyObjectRequest {  
        copySource = encodedUrl  
        bucket = toBucket  
        key = objectKey  
    }  
    S3Client { region = "us-east-1" }.use { s3 ->  
        s3.copyObject(request)  
    }  
}
```

- For API details, see [CopyObject in AWS SDK for Kotlin API reference](#).

PHP

SDK for PHP

Tip

To learn how to set up and run this example, see [GitHub](#).

Simple copy of an object.

```
$s3client = new Aws\S3\S3Client(['region' => 'us-west-2', 'version' => 'latest']);  
  
try {  
    $folder = "copied-folder";  
    $s3client->copyObject([  
        'Bucket' => $bucket_name,  
        'CopySource' => "$bucket_name/$file_name",  
        'Key' => "$folder/$file_name-copy",  
    ]);  
    echo "Copied $file_name to $folder/$file_name-copy.\n";  
} catch (Exception $exception) {  
    echo "Failed to copy $file_name with error: " . $exception->getMessage();  
    exit("Please fix error with object copying before continuing.");  
}
```

- For API details, see [CopyObject in AWS SDK for PHP API Reference](#).

Python

SDK for Python (Boto3)

Tip

To learn how to set up and run this example, see [GitHub](#).

```
class ObjectWrapper:
    def __init__(self, s3_object):
        self.object = s3_object
        self.key = self.object.key

    def copy(self, dest_object):
        """
        Copies the object to another bucket.

        :param dest_object: The destination object initialized with a bucket and
        key.
        """
        try:
            dest_object.copy_from(CopySource={
                'Bucket': self.object.bucket_name,
                'Key': self.object.key
            })
            dest_object.wait_until_exists()
            logger.info(
                "Copied object from %s:%s to %s:%s.",
                self.object.bucket_name, self.object.key,
                dest_object.bucket_name, dest_object.key)
        except ClientError:
            logger.exception(
                "Couldn't copy object from %s/%s to %s/%s.",
                self.object.bucket_name, self.object.key,
                dest_object.bucket_name, dest_object.key)
        raise
```

- For API details, see [CopyObject in AWS SDK for Python \(Boto3\) API Reference](#).

Ruby

SDK for Ruby

Tip

To learn how to set up and run this example, see [GitHub](#).

Copy an object.

```
require "aws-sdk-s3"

# Wraps Amazon S3 object actions.
class ObjectCopyWrapper
  attr_reader :source_object

  # @param source_object [Aws::S3::Object] An existing Amazon S3 object. This is
  # used as the source object for
  #                               copy actions.
  def initialize(source_object)
    @source_object = source_object
  end

  # Copy the source object to the specified target bucket and rename it with the
  # target key.
```

```
#  
# @param target_bucket [Aws::S3::Bucket] An existing Amazon S3 bucket where the  
object is copied.  
# @param target_object_key [String] The key to give the copy of the object.  
# @return [Aws::S3::Object, nil] The copied object when successful; otherwise,  
nil.  
def copy_object(target_bucket, target_object_key)  
  @source_object.copy_to(bucket: target_bucket.name, key: target_object_key)  
  target_bucket.object(target_object_key)  
rescue Aws::Errors::ServiceError => e  
  puts "Couldn't copy #{@source_object.key} to #{target_object_key}. Here's why:  
#{e.message}"  
end  
end  
  
# Replace the source and target bucket names with existing buckets you own and  
replace the source object key  
# with an existing object in the source bucket.  
def run_demo  
  source_bucket_name = "doc-example-bucket1"  
  source_key = "my-source-file.txt"  
  target_bucket_name = "doc-example-bucket2"  
  target_key = "my-target-file.txt"  
  
  source_bucket = Aws::S3::Bucket.new(source_bucket_name)  
  wrapper = ObjectCopyWrapper.new(source_bucket.object(source_key))  
  target_bucket = Aws::S3::Bucket.new(target_bucket_name)  
  target_object = wrapper.copy_object(target_bucket, target_key)  
  return unless target_object  
  
  puts "Copied #{source_key} from #{source_bucket_name} to  
#{target_object.bucket_name}:#{target_object.key}."  
end  
  
run_demo if $PROGRAM_NAME == __FILE__
```

Copy an object and add server-side encryption to the destination object.

```
require "aws-sdk-s3"  
  
# Wraps Amazon S3 object actions.  
class ObjectCopyEncryptWrapper  
  attr_reader :source_object  
  
  # @param source_object [Aws::S3::Object] An existing Amazon S3 object. This is  
used as the source object for  
  #                                         copy actions.  
  def initialize(source_object)  
    @source_object = source_object  
  end  
  
  # Copy the source object to the specified target bucket, rename it with the  
target key, and encrypt it.  
  #  
  # @param target_bucket [Aws::S3::Bucket] An existing Amazon S3 bucket where the  
object is copied.  
  # @param target_object_key [String] The key to give the copy of the object.  
  # @return [Aws::S3::Object, nil] The copied object when successful; otherwise,  
nil.  
  def copy_object(target_bucket, target_object_key, encryption)  
    @source_object.copy_to(bucket: target_bucket.name, key: target_object_key,  
server_side_encryption: encryption)  
    target_bucket.object(target_object_key)  
rescue Aws::Errors::ServiceError => e
```

```
    puts "Couldn't copy #{@source_object.key} to #{@target_object.key}. Here's why:
#{e.message}"
  end
end

# Replace the source and target bucket names with existing buckets you own and
# replace the source object key
# with an existing object in the source bucket.
def run_demo
  source_bucket_name = "doc-example-bucket1"
  source_key = "my-source-file.txt"
  target_bucket_name = "doc-example-bucket2"
  target_key = "my-target-file.txt"
  target_encryption = "AES256"

  source_bucket = Aws::S3::Bucket.new(source_bucket_name)
  wrapper = ObjectCopyEncryptWrapper.new(source_bucket.object(source_key))
  target_bucket = Aws::S3::Bucket.new(target_bucket_name)
  target_object = wrapper.copy_object(target_bucket, target_key, target_encryption)
  return unless target_object

  puts "Copied #{source_key} from #{source_bucket_name} to
#{@target_object.bucket_name}:#{@target_object.key} and \"\
    encrypted the target with #{@target_object.server_side_encryption}
encryption."
end

run_demo if $PROGRAM_NAME == __FILE__
```

- For API details, see [CopyObject](#) in *AWS SDK for Ruby API Reference*.

Rust

SDK for Rust

Note

This documentation is for an SDK in preview release. The SDK is subject to change and should not be used in production.

Tip

To learn how to set up and run this example, see [GitHub](#).

```
pub async fn copy_object(
    client: &Client,
    bucket_name: &str,
    object_key: &str,
    target_key: &str,
) -> Result<(), Error> {
    let mut source_bucket_and_object: String = "".to_owned();
    source_bucket_and_object.push_str(bucket_name);
    source_bucket_and_object.push('/');
    source_bucket_and_object.push_str(object_key);

    client
        .copy_object()
        .copy_source(source_bucket_and_object)
        .bucket(bucket_name)
        .key(target_key)
        .send()
        .await?;

    Ok(())
}
```

```
}
```

- For API details, see [CopyObject in AWS SDK for Rust API reference](#).

Swift

SDK for Swift

Note

This is prerelease documentation for an SDK in preview release. It is subject to change.

Tip

To learn how to set up and run this example, see [GitHub](#).

```
public func copyFile(from sourceBucket: String, name: String, to destBucket: String) async throws {
    let srcUrl = ("\"(sourceBucket)/"
\name").addingPercentEncoding(withAllowedCharacters: .urlPathAllowed)

    let input = CopyObjectInput(
        bucket: destBucket,
        copySource: srcUrl,
        key: name
    )
    _ = try await client.copyObject(input: input)
}
```

- For API details, see [CopyObject in AWS SDK for Swift API reference](#).

For a complete list of AWS SDK developer guides and code examples, see [Using this service with an AWS SDK \(p. 1185\)](#). This topic also includes information about getting started and details about previous SDK versions.

Create an Amazon S3 bucket using an AWS SDK

The following code examples show how to create an S3 bucket.

.NET

AWS SDK for .NET

Tip

To learn how to set up and run this example, see [GitHub](#).

```
/// <summary>
/// Shows how to create a new Amazon S3 bucket.
/// </summary>
/// <param name="client">An initialized Amazon S3 client object.</param>
/// <param name="bucketName">The name of the bucket to create.</param>
/// <returns>A boolean value representing the success or failure of
/// the bucket creation process.</returns>
public static async Task<bool> CreateBucketAsync(IAmazonS3 client, string
bucketName)
{
    try
```

```
{  
    var request = new PutBucketRequest  
    {  
        BucketName = bucketName,  
        UseClientRegion = true,  
    };  
  
    var response = await client.PutBucketAsync(request);  
    return response.HttpStatusCode == System.Net.HttpStatusCode.OK;  
}  
catch (AmazonS3Exception ex)  
{  
    Console.WriteLine($"Error creating bucket: '{ex.Message}'");  
    return false;  
}  
}
```

- For API details, see [CreateBucket](#) in *AWS SDK for .NET API Reference*.

C++

SDK for C++

Tip

To learn how to set up and run this example, see [GitHub](#).

```
bool AwsDoc::S3::CreateBucket(const Aws::String &bucketName, const Aws::String  
&region) {  
    // Create the bucket.  
    Aws::Client::ClientConfiguration clientConfig;  
    if (!region.empty())  
        clientConfig.region = region;  
  
    Aws::S3::S3Client client(clientConfig);  
    Aws::S3::Model::CreateBucketRequest request;  
    request.SetBucket(bucketName);  
  
    Aws::S3::Model::CreateBucketOutcome outcome = client.CreateBucket(request);  
    if (!outcome.IsSuccess())  
    {  
        auto err = outcome.GetError();  
        std::cout << "Error: CreateBucket: " <<  
            err.GetExceptionName() << ":" << err.GetMessage() << std::endl;  
        return false;  
    }  
    else  
    {  
        std::cout << "Created bucket " << bucketName <<  
            " in the specified AWS Region." << std::endl;  
        return true;  
    }  
}  
int main()  
{  
    Aws::SDKOptions options;  
    InitAPI(options);  
    //TODO: Set to the AWS Region of your account. If you don't, you will get  
    //IllegalLocationConstraintException Message: "The unspecified location  
    //constraint is incompatible
```

```
//for the Region specific endpoint this request was sent to."  
Aws::String region = "us-east-1";  
// Create a unique bucket name to increase the chance of success  
// when trying to create the bucket.  
// Format: "doc-example-bucket-" + lowercase UUID.  
Aws::String uuid = Aws::Utils::UUID::RandomUUID();  
Aws::String bucketName = "doc-example-bucket-" +  
    Aws::Utils::StringUtils::ToLower(uuid.c_str());  
  
AwsDoc::S3::CreateBucket(bucketName, region);  
  
ShutdownAPI(options);  
}
```

- For API details, see [CreateBucket](#) in *AWS SDK for C++ API Reference*.

Go

SDK for Go V2

Tip

To learn how to set up and run this example, see [GitHub](#).

```
// Create a bucket: We're going to create a bucket to hold content.  
// Best practice is to use the preset private access control list (ACL).  
// If you are not creating a bucket from us-east-1, you must specify a bucket  
location constraint.  
// Bucket names must conform to several rules; read more at https://  
docs.aws.amazon.com/AmazonS3/latest/userguide/bucketnamingrules.html  
_, err := client.CreateBucket(context.TODO(), &s3.CreateBucketInput{  
    Bucket: aws.String(name),  
    ACL: types.BucketCannedACLPrivate,  
    CreateBucketConfiguration: &types.CreateBucketConfiguration{LocationConstraint:  
        types.BucketLocationConstraintUsWest2},  
})  
  
if err != nil {  
    panic("could not create bucket: " + err.Error())  
}
```

- For API details, see [CreateBucket](#) in *AWS SDK for Go API Reference*.

Java

SDK for Java 2.x

Tip

To learn how to set up and run this example, see [GitHub](#).

```
public static void createBucket( S3Client s3Client, String bucketName) {  
  
    try {  
        S3Waiter s3Waiter = s3Client.waiter();  
        CreateBucketRequest bucketRequest = CreateBucketRequest.builder()  
            .bucket(bucketName)  
            .build();
```

```
s3Client.createBucket(bucketRequest);
HeadBucketRequest bucketRequestWait = HeadBucketRequest.builder()
    .bucket(bucketName)
    .build();

        // Wait until the bucket is created and print out the response.
        WaiterResponse<HeadBucketResponse> waiterResponse =
s3Waiter.waitUntilBucketExists(bucketRequestWait);
        waiterResponse.matched().response().ifPresent(System.out::println);
        System.out.println(bucketName + " is ready");

    } catch (S3Exception e) {
        System.err.println(e.awsErrorDetails().errorMessage());
        System.exit(1);
    }
}
```

- For API details, see [CreateBucket](#) in *AWS SDK for Java 2.x API Reference*.

JavaScript

SDK for JavaScript V3

Tip

To learn how to set up and run this example, see [GitHub](#).

Create the client.

```
// Create service client module using ES6 syntax.
import { S3Client } from "@aws-sdk/client-s3";
// Set the AWS Region.
const REGION = "us-east-1";
// Create an Amazon S3 service client object.
const s3Client = new S3Client({ region: REGION });
export { s3Client };
```

Create the bucket.

```
// Get service clients module and commands using ES6 syntax.
import { CreateBucketCommand } from "@aws-sdk/client-s3";
import { s3Client } from "./libs/s3Client.js";

// Set the bucket parameters.

export const bucketParams = { Bucket: "BUCKET_NAME" };

// Create the Amazon S3 bucket.
export const run = async () => {
    try {
        const data = await s3Client.send(new CreateBucketCommand(bucketParams));
        console.log("Success", data);
        return data; // For unit tests.
    } catch (err) {
        console.log("Error", err);
    }
};
run();
```

- For more information, see [AWS SDK for JavaScript Developer Guide](#).

- For API details, see [CreateBucket](#) in *AWS SDK for JavaScript API Reference*.

Kotlin

SDK for Kotlin

Note

This is prerelease documentation for a feature in preview release. It is subject to change.

Tip

To learn how to set up and run this example, see [GitHub](#).

```
suspend fun createNewBucket(bucketName: String) {

    val request = CreateBucketRequest {
        bucket = bucketName
    }

    S3Client { region = "us-east-1" }.use { s3 ->
        s3.createBucket(request)
        println("$bucketName is ready")
    }
}
```

- For API details, see [CreateBucket](#) in *AWS SDK for Kotlin API reference*.

PHP

SDK for PHP

Tip

To learn how to set up and run this example, see [GitHub](#).

Create a bucket.

```
$s3client = new Aws\S3\S3Client(['region' => 'us-west-2', 'version' => 'latest']);

try {
    $s3client->createBucket([
        'Bucket' => $bucket_name,
        'CreateBucketConfiguration' => ['LocationConstraint' => $region],
    ]);
    echo "Created bucket named: $bucket_name \n";
} catch (Exception $exception) {
    echo "Failed to create bucket $bucket_name with error: " . $exception-
>getMessage();
    exit("Please fix error with bucket creation before continuing.");
}
```

- For API details, see [CreateBucket](#) in *AWS SDK for PHP API Reference*.

Python

SDK for Python (Boto3)

Tip

To learn how to set up and run this example, see [GitHub](#).

Create a bucket with default settings.

```
class BucketWrapper:
    def __init__(self, bucket):
        self.bucket = bucket
        self.name = bucket.name

    def create(self, region_override=None):
        """
        Create an Amazon S3 bucket in the default Region for the account or in the
        specified Region.

        :param region_override: The Region in which to create the bucket. If this
        is
                           not specified, the Region configured in your shared
                           credentials is used.
        """
        if region_override is not None:
            region = region_override
        else:
            region = self.bucket.meta.client.meta.region_name
        try:
            self.bucket.create(
                CreateBucketConfiguration={'LocationConstraint': region})

            self.bucket.wait_until_exists()
            logger.info(
                "Created bucket '%s' in region=%s", self.bucket.name, region)
        except ClientError as error:
            logger.exception(
                "Couldn't create bucket named '%s' in region=%s.",
                self.bucket.name, region)
            raise error
```

Create a versioned bucket with a lifecycle configuration.

```
def create_versioned_bucket(bucket_name, prefix):
    """
    Creates an Amazon S3 bucket, enables it for versioning, and configures a
    lifecycle
    that expires noncurrent object versions after 7 days.

    Adding a lifecycle configuration to a versioned bucket is a best practice.
    It helps prevent objects in the bucket from accumulating a large number of
    noncurrent versions, which can slow down request performance.

    Usage is shown in the usage_demo_single_object function at the end of this
    module.

    :param bucket_name: The name of the bucket to create.
    :param prefix: Identifies which objects are automatically expired under the
                   configured lifecycle rules.
    :return: The newly created bucket.
    """
    try:
        bucket = s3.create_bucket(
            Bucket=bucket_name,
            CreateBucketConfiguration={
                'LocationConstraint': s3.meta.client.meta.region_name
            }
        )
        logger.info("Created bucket %s.", bucket.name)
    except ClientError as error:
```

```
if error.response['Error']['Code'] == 'BucketAlreadyOwnedByYou':
    logger.warning("Bucket %s already exists! Using it.", bucket_name)
    bucket = s3.Bucket(bucket_name)
else:
    logger.exception("Couldn't create bucket %s.", bucket_name)
    raise

try:
    bucket.Versioning().enable()
    logger.info("Enabled versioning on bucket %s.", bucket.name)
except ClientError:
    logger.exception("Couldn't enable versioning on bucket %s.", bucket.name)
    raise

try:
    expiration = 7
    bucket.LifecycleConfiguration().put(
        LifecycleConfiguration={
            'Rules': [
                {
                    'Status': 'Enabled',
                    'Prefix': prefix,
                    'NoncurrentVersionExpiration': {'NoncurrentDays': expiration}
                }
            ]
        }
    )
    logger.info("Configured lifecycle to expire noncurrent versions after %s
days "
               "on bucket %s.", expiration, bucket.name)
except ClientError as error:
    logger.warning("Couldn't configure lifecycle on bucket %s because %s. "
                  "Continuing anyway.", bucket.name, error)

return bucket
```

- For API details, see [CreateBucket in AWS SDK for Python \(Boto3\) API Reference](#).

Ruby

SDK for Ruby

Tip

To learn how to set up and run this example, see [GitHub](#).

```
require "aws-sdk-s3"

# Wraps Amazon S3 bucket actions.
class BucketCreateWrapper
  attr_reader :bucket

  # @param bucket [Aws::S3::Bucket] An Amazon S3 bucket initialized with a name.
  # This is a client-side object until
  #                               create is called.
  def initialize(bucket)
    @bucket = bucket
  end

  # Creates an Amazon S3 bucket in the specified AWS Region.
  #
  # @param region [String] The Region where the bucket is created.
  # @return [Boolean] True when the bucket is created; otherwise, false.
  def create?(region)
    @bucket.create(create_bucket_configuration: { location_constraint: region })
```

```
    true
rescue Aws::Errors::ServiceError => e
  puts "Couldn't create bucket. Here's why: #{e.message}"
  false
end

# Gets the Region where the bucket is located.
#
# @return [String] The location of the bucket.
def location
  if @bucket.nil?
    "None. You must create a bucket before you can get its location!"
  else
    @bucket.client.get_bucket_location(bucket: @bucket.name).location_constraint
  end
rescue Aws::Errors::ServiceError => e
  "Couldn't get the location of #{@bucket.name}. Here's why: #{e.message}"
end

def run_demo
  region = "us-west-2"
  wrapper = BucketCreateWrapper.new(Aws::S3::Bucket.new("doc-example-bucket-#{Random.uuid}"))
  return unless wrapper.create?(region)

  puts "Created bucket #{wrapper.bucket.name}."
  puts "Your bucket's region is: #{wrapper.location}"
end

run_demo if $PROGRAM_NAME == __FILE__
```

- For API details, see [CreateBucket](#) in *AWS SDK for Ruby API Reference*.

Rust

SDK for Rust

Note

This documentation is for an SDK in preview release. The SDK is subject to change and should not be used in production.

Tip

To learn how to set up and run this example, see [GitHub](#).

```
pub async fn create_bucket(client: &Client, bucket_name: &str, region: &str) ->
Result<(), Error> {
  let constraint = BucketLocationConstraint::from(region);
  let cfg = CreateBucketConfiguration::builder()
    .location_constraint(constraint)
    .build();
  client
    .create_bucket()
    .create_bucket_configuration(cfg)
    .bucket(bucket_name)
    .send()
    .await?;
  println!("{}", bucket_name);
  Ok(())
}
```

- For API details, see [CreateBucket](#) in [AWS SDK for Rust API reference](#).

Swift

SDK for Swift

Note

This is prerelease documentation for an SDK in preview release. It is subject to change.

Tip

To learn how to set up and run this example, see [GitHub](#).

```
public func createBucket(name: String) async throws {
    let config = S3ClientTypes.CreateBucketConfiguration(
        locationConstraint: .usEast2
    )
    let input = CreateBucketInput(
        bucket: name,
        createBucketConfiguration: config
    )
    _ = try await client.createBucket(input: input)
}
```

- For API details, see [CreateBucket](#) in [AWS SDK for Swift API reference](#).

For a complete list of AWS SDK developer guides and code examples, see [Using this service with an AWS SDK \(p. 1185\)](#). This topic also includes information about getting started and details about previous SDK versions.

Delete CORS rules from an Amazon S3 bucket using an AWS SDK

The following code examples show how to delete CORS rules from an S3 bucket.

Python

SDK for Python (Boto3)

Tip

To learn how to set up and run this example, see [GitHub](#).

```
class BucketWrapper:
    def __init__(self, bucket):
        self.bucket = bucket
        self.name = bucket.name

    def delete_cors(self):
        """
        Delete the CORS rules from the bucket.

        :param bucket_name: The name of the bucket to update.
        """
        try:
            self.bucket.Cors().delete()
            logger.info("Deleted CORS from bucket '%s'.", self.bucket.name)
        except ClientError:
```

```
logger.exception("Couldn't delete CORS from bucket '%s' .",
self.bucket.name)
raise
```

- For API details, see [DeleteBucketCors](#) in *AWS SDK for Python (Boto3) API Reference*.

Ruby

SDK for Ruby

Tip

To learn how to set up and run this example, see [GitHub](#).

```
require "aws-sdk-s3"

# Wraps Amazon S3 bucket CORS configuration.
class BucketCorsWrapper
    attr_reader :bucket_cors

    # @param bucket_cors [Aws::S3::BucketCors] A bucket CORS object configured with
    # an existing bucket.
    def initialize(bucket_cors)
        @bucket_cors = bucket_cors
    end

    # Deletes the CORS configuration of a bucket.
    #
    # @return [Boolean] True if the CORS rules were deleted; otherwise, false.
    def delete_cors
        @bucket_cors.delete
        true
    rescue Aws::Errors::ServiceError => e
        puts "Couldn't delete CORS rules for #{@bucket_cors.bucket.name}. Here's why:
#{e.message}"
        false
    end
end
```

- For API details, see [DeleteBucketCors](#) in *AWS SDK for Ruby API Reference*.

For a complete list of AWS SDK developer guides and code examples, see [Using this service with an AWS SDK \(p. 1185\)](#). This topic also includes information about getting started and details about previous SDK versions.

Delete a policy from an Amazon S3 bucket using an AWS SDK

The following code examples show how to delete a policy from an S3 bucket.

C++

SDK for C++

Tip

To learn how to set up and run this example, see [GitHub](#).

```
bool AwsDoc::S3::DeleteBucketPolicy(const Aws::String &bucketName, const
Aws::String &region) {
    Aws::Client::ClientConfiguration clientConfig;
    if (!region.empty()) {
        clientConfig.region = region;
    }

    Aws::S3::S3Client client(clientConfig);

    Aws::S3::Model::DeleteBucketPolicyRequest request;
    request.SetBucket(bucketName);

    Aws::S3::Model::DeleteBucketPolicyOutcome outcome =
client.DeleteBucketPolicy(request);

    if (!outcome.IsSuccess())
    {
        auto err = outcome.GetError();
        std::cout << "Error: DeleteBucketPolicy: " <<
err.GetExceptionName() << ":" << err.GetMessage() << std::endl;

        return false;
    }
    else
    {
        std::cout << "Policy was deleted from the bucket." << std::endl;
        return true;
    }
}

int main()
{
    //TODO: Change bucket_name to the name of a bucket in your account.
    const Aws::String bucketName = "<Enter bucket name>";
    //TODO: Set to the AWS Region in which the bucket was created.
    const Aws::String region = "us-east-1";

    Aws::SDKOptions options;
    Aws::InitAPI(options);

    AwsDoc::S3::DeleteBucketPolicy(bucketName, region);

    ShutdownAPI(options);
}
```

- For API details, see [DeleteBucketPolicy](#) in *AWS SDK for C++ API Reference*.

Java

SDK for Java 2.x

Tip

To learn how to set up and run this example, see [GitHub](#).

```
// Delete the bucket policy.
public static void deleteS3BucketPolicy(S3Client s3, String bucketName) {

    DeleteBucketPolicyRequest delReq = DeleteBucketPolicyRequest.builder()
```

```
.bucket(bucketName)
.build();

try {
    s3.deleteBucketPolicy(delReq);
    System.out.println("Done!");
} catch (S3Exception e) {
    System.err.println(e.awsErrorDetails().errorMessage());
    System.exit(1);
}
```

- For API details, see [DeleteBucketPolicy](#) in *AWS SDK for Java 2.x API Reference*.

JavaScript

SDK for JavaScript V3

Tip

To learn how to set up and run this example, see [GitHub](#).

Create the client.

```
// Create service client module using ES6 syntax.
import { S3Client } from "@aws-sdk/client-s3";
// Set the AWS Region.
const REGION = "us-east-1";
// Create an Amazon S3 service client object.
const s3Client = new S3Client({ region: REGION });
export { s3Client };
```

Delete the bucket policy.

```
// Import required AWS SDK clients and commands for Node.js.
import { DeleteBucketPolicyCommand } from "@aws-sdk/client-s3";
import { s3Client } from "./libs/s3Client.js"; // Helper function that creates an
Amazon S3 service client module.

// Set the bucket parameters
export const bucketParams = { Bucket: "BUCKET_NAME" };

export const run = async () => {
    try {
        const data = await s3Client.send(new DeleteBucketPolicyCommand(bucketParams));
        console.log("Success", data + ", bucket policy deleted");
        return data; // For unit tests.
    } catch (err) {
        console.log("Error", err);
    }
};
// Invoke run() so these examples run out of the box.
run();
```

- For more information, see [AWS SDK for JavaScript Developer Guide](#).
- For API details, see [DeleteBucketPolicy](#) in *AWS SDK for JavaScript API Reference*.

Kotlin

SDK for Kotlin

Note

This is prerelease documentation for a feature in preview release. It is subject to change.

Tip

To learn how to set up and run this example, see [GitHub](#).

```
suspend fun deleteS3BucketPolicy(bucketName: String?) {  
  
    val request = DeleteBucketPolicyRequest {  
        bucket = bucketName  
    }  
  
    S3Client { region = "us-east-1" }.use { s3 ->  
        s3.deleteBucketPolicy(request)  
        println("Done!")  
    }  
}
```

- For API details, see [DeleteBucketPolicy](#) in *AWS SDK for Kotlin API reference*.

Python

SDK for Python (Boto3)

Tip

To learn how to set up and run this example, see [GitHub](#).

```
class BucketWrapper:  
    def __init__(self, bucket):  
        self.bucket = bucket  
        self.name = bucket.name  
  
    def delete_policy(self):  
        """  
        Delete the security policy from the bucket.  
        """  
        try:  
            self.bucket.Policy().delete()  
            logger.info("Deleted policy for bucket '%s'.", self.bucket.name)  
        except ClientError:  
            logger.exception("Couldn't delete policy for bucket '%s'.",  
                             self.bucket.name)  
            raise
```

- For API details, see [DeleteBucketPolicy](#) in *AWS SDK for Python (Boto3) API Reference*.

Ruby

SDK for Ruby

Tip

To learn how to set up and run this example, see [GitHub](#).

```
# Wraps an Amazon S3 bucket policy.
class BucketPolicyWrapper
  attr_reader :bucket_policy

  # @param bucket_policy [Aws::S3::BucketPolicy] A bucket policy object configured
  # with an existing bucket.
  def initialize(bucket_policy)
    @bucket_policy = bucket_policy
  end

  def delete_policy
    @bucket_policy.delete
    true
  rescue Aws::Errors::ServiceError => e
    puts "Couldn't delete the policy from #{@bucket_policy.bucket.name}. Here's
why: #{e.message}"
    false
  end

end
```

- For API details, see [DeleteBucketPolicy](#) in *AWS SDK for Ruby API Reference*.

For a complete list of AWS SDK developer guides and code examples, see [Using this service with an AWS SDK \(p. 1185\)](#). This topic also includes information about getting started and details about previous SDK versions.

Delete an empty Amazon S3 bucket using an AWS SDK

The following code examples show how to delete an empty S3 bucket.

.NET

AWS SDK for .NET

Tip

To learn how to set up and run this example, see [GitHub](#).

```
/// <summary>
/// Shows how to delete an Amazon S3 bucket.
/// </summary>
/// <param name="client">An initialized Amazon S3 client object.</param>
/// <param name="bucketName">The name of the Amazon S3 bucket to delete.</param>
/// <returns>A boolean value that represents the success or failure of
/// the delete operation.</returns>
public static async Task<bool> DeleteBucketAsync(IAmazonS3 client, string
bucketName)
{
    var request = new DeleteBucketRequest
    {
        BucketName = bucketName,
    };

    var response = await client.DeleteBucketAsync(request);
    return response.HttpStatusCode == System.Net.HttpStatusCode.OK;
```

```
}
```

- For API details, see [DeleteBucket](#) in [AWS SDK for .NET API Reference](#).

C++

SDK for C++

Tip

To learn how to set up and run this example, see [GitHub](#).

```
using namespace Aws;

bool AwsDoc::S3::DeleteBucket(const Aws::String &bucketName, const Aws::String
    &region)
{
    Aws::Client::ClientConfiguration clientConfig;
    if (!region.empty())
        clientConfig.region = region;
}

Aws::S3::S3Client client(clientConfig);

Aws::S3::Model::DeleteBucketRequest request;
request.SetBucket(bucketName);

Aws::S3::Model::DeleteBucketOutcome outcome =
    client.DeleteBucket(request);

if (!outcome.IsSuccess())
{
    auto err = outcome.GetError();
    std::cout << "Error: DeleteBucket: " <<
        err.ExceptionName() << ":" << err.Message() << std::endl;
    return false;
}
else
{
    std::cout << "The bucket was deleted" << std::endl;
    return true;
}
}

int main()
{
    //TODO: Change bucket_name to the name of a bucket in your account.
    //If the bucket is not in your account, you will get one of two errors:
    Aws::String bucketName = "<Bucket Name>";
    //TODO: Set to the AWS Region of the bucket bucket_name.
    Aws::String region = "us-east-1";

    Aws::SDKOptions options;
    Aws::InitAPI(options);

    AwsDoc::S3::DeleteBucket(bucketName, region);

    ShutdownAPI(options);
}
```

- For API details, see [DeleteBucket](#) in [AWS SDK for C++ API Reference](#).

Go

SDK for Go V2

Tip

To learn how to set up and run this example, see [GitHub](#).

```
fmt.Println("Delete a bucket")
// Delete the bucket.

_, err = client.DeleteBucket(context.TODO(), &s3.DeleteBucketInput{
    Bucket: aws.String(name),
})
if err != nil {
    panic("Couldn't delete bucket: " + err.Error())
}
```

- For API details, see [DeleteBucket](#) in [AWS SDK for Go API Reference](#).

Java

SDK for Java 2.x

Tip

To learn how to set up and run this example, see [GitHub](#).

```
DeleteBucketRequest deleteBucketRequest = DeleteBucketRequest.builder()
    .bucket(bucket)
    .build();

s3.deleteBucket(deleteBucketRequest);
s3.close();
```

- For API details, see [DeleteBucket](#) in [AWS SDK for Java 2.x API Reference](#).

JavaScript

SDK for JavaScript V3

Tip

To learn how to set up and run this example, see [GitHub](#).

Create the client.

```
// Create service client module using ES6 syntax.
import { S3Client } from "@aws-sdk/client-s3";
// Set the AWS Region.
const REGION = "us-east-1";
// Create an Amazon S3 service client object.
const s3Client = new S3Client({ region: REGION });
export { s3Client };
```

Delete the bucket.

```
// Import required AWS SDK clients and commands for Node.js.
import { DeleteBucketCommand } from "@aws-sdk/client-s3";
import { s3Client } from "./libs/s3Client.js"; // Helper function that creates an
// Amazon S3 service client module.

// Set the bucket parameters
export const bucketParams = { Bucket: "BUCKET_NAME" };

export const run = async () => {
  try {
    const data = await s3Client.send(new DeleteBucketCommand(bucketParams));
    return data; // For unit tests.
    console.log("Success - bucket deleted");
  } catch (err) {
    console.log("Error", err);
  }
};

// Invoke run() so these examples run out of the box.
run();
```

- For more information, see [AWS SDK for JavaScript Developer Guide](#).
- For API details, see [DeleteBucket](#) in [AWS SDK for JavaScript API Reference](#).

PHP

SDK for PHP

Tip

To learn how to set up and run this example, see [GitHub](#).

Delete an empty bucket.

```
$s3client = new Aws\S3\S3Client(['region' => 'us-west-2', 'version' => 'latest']);

try {
    $s3client->deleteBucket([
        'Bucket' => $bucket_name,
    ]);
    echo "Deleted bucket $bucket_name.\n";
} catch (Exception $exception) {
    echo "Failed to delete $bucket_name with error: " . $exception->getMessage();
    exit("Please fix error with bucket deletion before continuing.");
}
```

- For API details, see [DeleteBucket](#) in [AWS SDK for PHP API Reference](#).

Python

SDK for Python (Boto3)

Tip

To learn how to set up and run this example, see [GitHub](#).

```
class BucketWrapper:
    def __init__(self, bucket):
        self.bucket = bucket
        self.name = bucket.name
```

```
def delete(self):
    """
    Delete the bucket. The bucket must be empty or an error is raised.
    """
    try:
        self.bucket.delete()
        self.bucket.wait_until_not_exists()
        logger.info("Bucket %s successfully deleted.", self.bucket.name)
    except ClientError:
        logger.exception("Couldn't delete bucket %s.", self.bucket.name)
        raise
```

- For API details, see [DeleteBucket in AWS SDK for Python \(Boto3\) API Reference](#).

Ruby

SDK for Ruby

Tip

To learn how to set up and run this example, see [GitHub](#).

```
# Deletes the objects in an Amazon S3 bucket and deletes the bucket.
#
# @param bucket [Aws::S3::Bucket] The bucket to empty and delete.
def delete_bucket(bucket)
  puts("\nDo you want to delete all of the objects as well as the bucket (y/n)?")
  answer = gets.chomp.downcase
  if answer == "y"
    bucket.objects.batch_delete!
    bucket.delete
    puts("Emptied and deleted bucket #{bucket.name}.\n")
  end
rescue Aws::Errors::ServiceError => e
  puts("Couldn't empty and delete bucket #{bucket.name}.")
  puts("\t#{e.code}: #{e.message}")
  raise
end
```

- For API details, see [DeleteBucket in AWS SDK for Ruby API Reference](#).

Rust

SDK for Rust

Note

This documentation is for an SDK in preview release. The SDK is subject to change and should not be used in production.

Tip

To learn how to set up and run this example, see [GitHub](#).

```
pub async fn delete_bucket(client: &Client, bucket_name: &str) -> Result<(), Error>
{
    client.delete_bucket().bucket(bucket_name).send().await?;
    println!("bucket deleted");
    Ok(())
}
```

- For API details, see [DeleteBucket](#) in *AWS SDK for Rust API reference*.

Swift

SDK for Swift

Note

This is prerelease documentation for an SDK in preview release. It is subject to change.

Tip

To learn how to set up and run this example, see [GitHub](#).

```
public func deleteBucket(name: String) async throws {
    let input = DeleteBucketInput(
        bucket: name
    )
    _ = try await client.deleteBucket(input: input)
}
```

- For API details, see [DeleteBucket](#) in *AWS SDK for Swift API reference*.

For a complete list of AWS SDK developer guides and code examples, see [Using this service with an AWS SDK \(p. 1185\)](#). This topic also includes information about getting started and details about previous SDK versions.

Delete an Amazon S3 object using an AWS SDK

The following code examples show how to delete an S3 object.

C++

SDK for C++

Tip

To learn how to set up and run this example, see [GitHub](#).

```
bool AwsDoc::S3::DeleteObject(const Aws::String &objectKey, const Aws::String
&fromBucket, const Aws::String &region) {
    Aws::Client::ClientConfiguration clientConfig;
    if (!region.empty()) {
        clientConfig.region = region;
    }

    Aws::S3::S3Client client(clientConfig);
    Aws::S3::Model::DeleteObjectRequest request;

    request.WithKey(objectKey)
        .WithBucket(fromBucket);

    Aws::S3::Model::DeleteObjectOutcome outcome =
        client.DeleteObject(request);

    if (!outcome.IsSuccess())
    {
        auto err = outcome.GetError();
        std::cout << "Error: DeleteObject: " <<
```

```
        err.GetExceptionName() << ":" << err.GetMessage() << std::endl;
    return false;
}
else
{
    std::cout << "Successfully deleted the object." << std::endl;
    return true;
}

int main()
{
    //TODO: The object_key is the unique identifier for the object in the bucket.
    //In this example set,
    //it is the filename you added in put_object.cpp.
    Aws::String objectKey = "<Enter object key>";
    //TODO: Change from_bucket to the name of a bucket in your account.
    Aws::String fromBucket = "<Enter bucket name>";
    //TODO: Set to the AWS Region in which the bucket was created.
    Aws::String region = "us-east-1";

    Aws::SDKOptions options;
    Aws::InitAPI(options);

    AwsDoc::S3::DeleteObject(objectKey, fromBucket, region);

    ShutdownAPI(options);

    return 0;
}
```

- For API details, see [DeleteObject](#) in *AWS SDK for C++ API Reference*.

Go

SDK for Go V2

Tip

To learn how to set up and run this example, see [GitHub](#).

```
// Delete a single object.
fmt.Println("Delete an object from a bucket")
_, err := client.DeleteObject(context.TODO(), &s3.DeleteObjectInput{
    Bucket: aws.String(name),
    Key:    aws.String("other/file.jpg"),
})
if err != nil {
    panic("Couldn't delete object!")
}
```

- For API details, see [DeleteObject](#) in *AWS SDK for Go API Reference*.

JavaScript

SDK for JavaScript V3

Tip

To learn how to set up and run this example, see [GitHub](#).

Create the client.

```
// Create service client module using ES6 syntax.  
import { S3Client } from "@aws-sdk/client-s3";  
// Set the AWS Region.  
const REGION = "us-east-1";  
// Create an Amazon S3 service client object.  
const s3Client = new S3Client({ region: REGION });  
export { s3Client };
```

Delete an object.

```
import { DeleteObjectCommand } from "@aws-sdk/client-s3";  
import { s3Client } from "./libs/s3Client.js" // Helper function that creates an  
Amazon S3 service client module.  
  
export const bucketParams = { Bucket: "BUCKET_NAME", Key: "KEY" };  
  
export const run = async () => {  
    try {  
        const data = await s3Client.send(new DeleteObjectCommand(bucketParams));  
        console.log("Success. Object deleted.", data);  
        return data; // For unit tests.  
    } catch (err) {  
        console.log("Error", err);  
    }  
};  
run();
```

- For API details, see [DeleteObject](#) in *AWS SDK for JavaScript API Reference*.

Python

SDK for Python (Boto3)

Tip

To learn how to set up and run this example, see [GitHub](#).

Delete an object.

```
class ObjectWrapper:  
    def __init__(self, s3_object):  
        self.object = s3_object  
        self.key = self.object.key  
  
    def delete(self):  
        """  
        Deletes the object.  
        """  
        try:  
            self.object.delete()  
            self.object.wait_until_not_exists()  
            logger.info(  
                "Deleted object '%s' from bucket '%s'.",  
                self.object.key, self.object.bucket_name)  
        except ClientError:  
            logger.exception(  
                "Couldn't delete object '%s' from bucket '%s'.",  
                self.object.key, self.object.bucket_name)  
        raise
```

Roll an object back to a previous version by deleting later versions of the object.

```
def rollback_object(bucket, object_key, version_id):
    """
    Rolls back an object to an earlier version by deleting all versions that
    occurred after the specified rollback version.

    Usage is shown in the usage_demo_single_object function at the end of this
    module.

    :param bucket: The bucket that holds the object to roll back.
    :param object_key: The object to roll back.
    :param version_id: The version ID to roll back to.
    """
    # Versions must be sorted by last_modified date because delete markers are
    # at the end of the list even when they are interspersed in time.
    versions = sorted(bucket.object_versions.filter(Prefix=object_key),
                      key=attrgetter('last_modified'), reverse=True)

    logger.debug(
        "Got versions:\n%s",
        '\n'.join([f"\t{version.version_id}, last modified {version.last_modified}"
                  for version in versions]))

    if version_id in [ver.version_id for ver in versions]:
        print(f"Rolling back to version {version_id}")
        for version in versions:
            if version.version_id != version_id:
                version.delete()
                print(f"Deleted version {version.version_id}")
            else:
                break

        print(f"Active version is now {bucket.Object(object_key).version_id}")
    else:
        raise KeyError(f"{version_id} was not found in the list of versions for "
                      f"{object_key}.")
```

Revive a deleted object by removing the object's active delete marker.

```
def revive_object(bucket, object_key):
    """
    Revives a versioned object that was deleted by removing the object's active
    delete marker.

    A versioned object presents as deleted when its latest version is a delete
    marker.

    By removing the delete marker, we make the previous version the latest version
    and the object then presents as *not* deleted.

    Usage is shown in the usage_demo_single_object function at the end of this
    module.

    :param bucket: The bucket that contains the object.
    :param object_key: The object to revive.
    """
    # Get the latest version for the object.
    response = s3.meta.client.list_object_versions(
        Bucket=bucket.name, Prefix=object_key, MaxKeys=1)

    if 'DeleteMarkers' in response:
```

```
latest_version = response['DeleteMarkers'][0]
if latest_version['IsLatest']:
    logger.info("Object %s was indeed deleted on %s. Let's revive it.",
                object_key, latest_version['LastModified'])
    obj = bucket.Object(object_key)
    obj.Version(latest_version['VersionId']).delete()
    logger.info("Revived %s, active version is now %s with body '%s'",
                object_key, obj.version_id, obj.get()['Body'].read())
else:
    logger.warning("Delete marker is not the latest version for %s!",
                  object_key)
elif 'Versions' in response:
    logger.warning("Got an active version for %s, nothing to do.", object_key)
else:
    logger.error("Couldn't get any version info for %s.", object_key)
```

Create a Lambda handler that removes a delete marker from an S3 object. This handler can be used to efficiently clean up extraneous delete markers in a versioned bucket.

```
import logging
from urllib import parse
import boto3
from botocore.exceptions import ClientError

logger = logging.getLogger(__name__)
logger.setLevel('INFO')

s3 = boto3.client('s3')


def lambda_handler(event, context):
    """
    Removes a delete marker from the specified versioned object.

    :param event: The S3 batch event that contains the ID of the delete marker
                  to remove.
    :param context: Context about the event.
    :return: A result structure that Amazon S3 uses to interpret the result of the
            operation. When the result code is TemporaryFailure, S3 retries the
            operation.
    """
    # Parse job parameters from Amazon S3 batch operations
    invocation_id = event['invocationId']
    invocation_schema_version = event['invocationSchemaVersion']

    results = []
    result_code = None
    result_string = None

    task = event['tasks'][0]
    task_id = task['taskId']

    try:
        obj_key = parse.unquote(task['s3Key'], encoding='utf-8')
        obj_version_id = task['s3VersionId']
        bucket_name = task['s3BucketArn'].split(':')[ -1]

        logger.info("Got task: remove delete marker %s from object %s.",
                    obj_version_id, obj_key)

        try:
            # If this call does not raise an error, the object version is not a
            delete
```

```
# marker and should not be deleted.
response = s3.head_object(
    Bucket=bucket_name, Key=obj_key, VersionId=obj_version_id)
result_code = 'PermanentFailure'
result_string = f"Object {obj_key}, ID {obj_version_id} is not " \
                f'a delete marker.'

logger.debug(response)
logger.warning(result_string)
except ClientError as error:
    delete_marker = error.response['ResponseMetadata']['HTTPHeaders'] \
        .get('x-amz-delete-marker', 'false')
    if delete_marker == 'true':
        logger.info("Object %s, version %s is a delete marker.", obj_key, obj_version_id)
    try:
        s3.delete_object(
            Bucket=bucket_name, Key=obj_key, VersionId=obj_version_id)
        result_code = 'Succeeded'
        result_string = f"Successfully removed delete marker " \
                        f"{obj_version_id} from object {obj_key}."
        logger.info(result_string)
    except ClientError as error:
        # Mark request timeout as a temporary failure so it will be
        retried.
        if error.response['Error']['Code'] == 'RequestTimeout':
            result_code = 'TemporaryFailure'
            result_string = f"Attempt to remove delete marker from " \
                            f"object {obj_key} timed out."
            logger.info(result_string)
        else:
            raise
    else:
        raise ValueError(f"The x-amz-delete-marker header is either not "
                         f"present or is not 'true'.")
except Exception as error:
    # Mark all other exceptions as permanent failures.
    result_code = 'PermanentFailure'
    result_string = str(error)
    logger.exception(error)
finally:
    results.append({
        'taskId': task_id,
        'resultCode': result_code,
        'resultString': result_string
    })
return {
    'invocationSchemaVersion': invocation_schema_version,
    'treatMissingKeysAs': 'PermanentFailure',
    'invocationId': invocation_id,
    'results': results
}
```

- For API details, see [DeleteObject](#) in *AWS SDK for Python (Boto3) API Reference*.

Rust

SDK for Rust

Note

This documentation is for an SDK in preview release. The SDK is subject to change and should not be used in production.

Tip

To learn how to set up and run this example, see [GitHub](#).

```
async fn remove_object(client: &Client, bucket: &str, key: &str) -> Result<(),  
Error> {  
    client  
        .delete_object()  
        .bucket(bucket)  
        .key(key)  
        .send()  
        .await?;  
  
    println!("Object deleted.");  
  
    Ok(())
}
```

- For API details, see [DeleteObject](#) in *AWS SDK for Rust API reference*.

Swift

SDK for Swift

Note

This is prerelease documentation for an SDK in preview release. It is subject to change.

Tip

To learn how to set up and run this example, see [GitHub](#).

```
public func deleteFile(bucket: String, key: String) async throws {  
    let input = DeleteObjectInput(  
        bucket: bucket,  
        key: key  
    )  
  
    do {  
        _ = try await client.deleteObject(input: input)  
    } catch {  
        throw error  
    }  
}
```

- For API details, see [DeleteObject](#) in *AWS SDK for Swift API reference*.

For a complete list of AWS SDK developer guides and code examples, see [Using this service with an AWS SDK \(p. 1185\)](#). This topic also includes information about getting started and details about previous SDK versions.

Delete multiple objects from an Amazon S3 bucket using an AWS SDK

The following code examples show how to delete multiple objects from an S3 bucket.

.NET

AWS SDK for .NET

Tip

To learn how to set up and run this example, see [GitHub](#).

Delete all objects in an S3 bucket.

```
/// <summary>
/// Delete all of the objects stored in an existing Amazon S3 bucket.
/// </summary>
/// <param name="client">An initialized Amazon S3 client object.</param>
/// <param name="bucketName">The name of the bucket from which the
/// contents will be deleted.</param>
/// <returns>A boolean value that represents the success or failure of
/// deleting all of the objects in the bucket.</returns>
public static async Task<bool> DeleteBucketContentsAsync(IAmazonS3 client,
string bucketName)
{
    // Iterate over the contents of the bucket and delete all objects.
    var request = new ListObjectsV2Request
    {
        BucketName = bucketName,
    };

    try
    {
        var response = await client.ListObjectsV2Async(request);

        do
        {
            response.S3Objects
                .ForEach(async obj => await
client.DeleteObjectAsync(bucketName, obj.Key));

            // If the response is truncated, set the request
ContinuationToken
                // from the NextContinuationToken property of the response.
                request.ContinuationToken = response.NextContinuationToken;
        }
        while (response.IsTruncated);

        return true;
    }
    catch (AmazonS3Exception ex)
    {
        Console.WriteLine($"Error deleting objects: {ex.Message}");
        return false;
    }
}
```

- For API details, see [DeleteObjects](#) in *AWS SDK for .NET API Reference*.

Java

SDK for Java 2.x

Tip

To learn how to set up and run this example, see [GitHub](#).

```
public static void deleteBucketObjects(S3Client s3, String bucketName) {

    // Upload three sample objects to the specified Amazon S3 bucket.
    ArrayList<ObjectIdentifier> keys = new ArrayList<>();
    PutObjectRequest putOb;
    ObjectIdentifier objectId;

    for (int i = 0; i < 3; i++) {
        String keyName = "delete object example " + i;
        objectId = ObjectIdentifier.builder()
            .key(keyName)
            .build();

        putOb = PutObjectRequest.builder()
            .bucket(bucketName)
            .key(keyName)
            .build();

        s3.putObject(putOb, RequestBody.fromString(keyName));
        keys.add(objectId);
    }

    System.out.println(keys.size() + " objects successfully created.");

    // Delete multiple objects in one request.
    Delete del = Delete.builder()
        .objects(keys)
        .build();

    try {
        DeleteObjectsRequest multiObjectDeleteRequest =
DeleteObjectsRequest.builder()
        .bucket(bucketName)
        .delete(del)
        .build();

        s3.deleteObjects(multiObjectDeleteRequest);
        System.out.println("Multiple objects are deleted!");

    } catch (S3Exception e) {
        System.err.println(e.awsErrorDetails().errorMessage());
        System.exit(1);
    }
}
```

- For API details, see [DeleteObjects](#) in *AWS SDK for Java 2.x API Reference*.

JavaScript

SDK for JavaScript V3

Tip

To learn how to set up and run this example, see [GitHub](#).

Create the client.

```
// Create service client module using ES6 syntax.
import { S3Client } from "@aws-sdk/client-s3";
// Set the AWS Region.
const REGION = "us-east-1";
// Create an Amazon S3 service client object.
```

```
const s3Client = new S3Client({ region: REGION });
export { s3Client };
```

Delete multiple objects.

```
import { DeleteObjectsCommand } from "@aws-sdk/client-s3";
import { s3Client } from "./libs/s3Client.js" // Helper function that creates an
Amazon S3 service client module.

export const bucketParams = {
  Bucket: "BUCKET_NAME",
  Delete: {
    Objects: [
      {
        Key: "KEY_1",
      },
      {
        Key: "KEY_2",
      },
    ],
  },
};

export const run = async () => {
  try {
    const data = await s3Client.send(new DeleteObjectsCommand(bucketParams));
    return data; // For unit tests.
    console.log("Success. Object deleted.");
  } catch (err) {
    console.log("Error", err);
  }
};
run();
```

Delete all objects in a bucket.

```
import { ListObjectsCommand, DeleteObjectCommand } from "@aws-sdk/client-s3";
import { s3Client } from "./libs/s3Client.js"; // Helper function that creates an
Amazon S3 service client module.

export const bucketParams = { Bucket: "BUCKET_NAME" };

export const run = async () => {
  try {
    const data = await s3Client.send(new ListObjectsCommand(bucketParams));
    return data; // For unit tests.
    let i = 0;
    let noOfObjects = data.Contents;
    for (let i = 0; i < noOfObjects.length; i++) {
      const data = await s3Client.send(
        new DeleteObjectCommand({
          Bucket: bucketParams.Bucket,
          Key: noOfObjects[i].Key,
        })
      );
    }
    console.log("Success. Objects deleted.");
  } catch (err) {
    console.log("Error", err);
  }
};
```

```
run();
```

- For API details, see [DeleteObjects](#) in *AWS SDK for JavaScript API Reference*.

Kotlin

SDK for Kotlin

Note

This is prerelease documentation for a feature in preview release. It is subject to change.

Tip

To learn how to set up and run this example, see [GitHub](#).

```
suspend fun deleteBucketObjects(bucketName: String, objectName: String) {

    val objectId = ObjectIdentifier {
        key = objectName
    }

    val delOb = Delete {
        objects = listOf(objectId)
    }

    val request = DeleteObjectsRequest {
        bucket = bucketName
        delete = delOb
    }

    S3Client { region = "us-east-1" }.use { s3 ->
        s3.deleteObjects(request)
        println("$objectName was deleted from $bucketName")
    }
}
```

- For API details, see [DeleteObjects](#) in *AWS SDK for Kotlin API reference*.

PHP

SDK for PHP

Tip

To learn how to set up and run this example, see [GitHub](#).

Delete a set of objects from a list of keys.

```
$s3client = new Aws\S3\S3Client(['region' => 'us-west-2', 'version' => 'latest']);

try {
    $objects = [];
    foreach ($contents['Contents'] as $content) {
        $objects[] = [
            'Key' => $content['Key'],
        ];
    }
    $s3client->deleteObjects([
        'Bucket' => $bucket_name,
```

```
'Key' => $file_name,
'Delete' => [
    'Objects' => $objects,
],
]);
$check = $s3client->listObjects([
    'Bucket' => $bucket_name,
]);
if (count($check) <= 0) {
    throw new Exception("Bucket wasn't empty.");
}
echo "Deleted all objects and folders from $bucket_name.\n";
} catch (Exception $exception) {
    echo "Failed to delete $file_name from $bucket_name with error: " . $exception-
>getMessage();
    exit("Please fix error with object deletion before continuing.");
}
```

- For API details, see [DeleteObjects](#) in *AWS SDK for PHP API Reference*.

Python

SDK for Python (Boto3)

Tip

To learn how to set up and run this example, see [GitHub](#).

Delete a set of objects by using a list of object keys.

```
class ObjectWrapper:
    def __init__(self, s3_object):
        self.object = s3_object
        self.key = self.object.key

    @staticmethod
    def delete_objects(bucket, object_keys):
        """
        Removes a list of objects from a bucket.
        This operation is done as a batch in a single request.

        :param bucket: The bucket that contains the objects.
        :param object_keys: The list of keys that identify the objects to remove.
        :return: The response that contains data about which objects were deleted
                and any that could not be deleted.
        """
        try:
            response = bucket.delete_objects(Delete={
                'Objects': [
                    {
                        'Key': key
                    } for key in object_keys
                ]
            })
            if 'Deleted' in response:
                logger.info(
                    "Deleted objects '%s' from bucket '%s'.",
                    [del_obj['Key'] for del_obj in response['Deleted']],
                    bucket.name)
            if 'Errors' in response:
                logger.warning(
                    "Could not delete objects '%s' from bucket '%s'.",
                    [{"del_obj['Key']: del_obj['Code']} for del_obj in response['Errors']],
                    bucket.name)
        except ClientError:
```

```
        logger.exception("Couldn't delete any objects from bucket %s.",  
bucket.name)  
        raise  
    else:  
        return response
```

Delete all objects in a bucket.

```
class ObjectWrapper:  
    def __init__(self, s3_object):  
        self.object = s3_object  
        self.key = self.object.key  
  
    @staticmethod  
    def empty_bucket(bucket):  
        """  
        Remove all objects from a bucket.  
  
        :param bucket: The bucket to empty.  
        """  
        try:  
            bucket.objects.delete()  
            logger.info("Emptied bucket '%s'.", bucket.name)  
        except ClientError:  
            logger.exception("Couldn't empty bucket '%s'.", bucket.name)  
            raise
```

Permanently delete a versioned object by deleting all of its versions.

```
def permanently_delete_object(bucket, object_key):  
    """  
    Permanently deletes a versioned object by deleting all of its versions.  
  
    Usage is shown in the usage_demo_single_object function at the end of this  
    module.  
  
    :param bucket: The bucket that contains the object.  
    :param object_key: The object to delete.  
    """  
    try:  
        bucket.object_versions.filter(Prefix=object_key).delete()  
        logger.info("Permanently deleted all versions of object %s.", object_key)  
    except ClientError:  
        logger.exception("Couldn't delete all versions of %s.", object_key)  
        raise
```

- For API details, see [DeleteObjects](#) in *AWS SDK for Python (Boto3) API Reference*.

Ruby

SDK for Ruby

Tip

To learn how to set up and run this example, see [GitHub](#).

```
# Deletes the objects in an Amazon S3 bucket and deletes the bucket.
```

```
#  
# @param bucket [Aws::S3::Bucket] The bucket to empty and delete.  
def delete_bucket(bucket)  
    puts("\nDo you want to delete all of the objects as well as the bucket (y/n)?")  
    answer = gets.chomp.downcase  
    if answer == "y"  
        bucket.objects.batch_delete!  
        bucket.delete  
        puts("Emptied and deleted bucket #{bucket.name}.\n")  
    end  
rescue Aws::Errors::ServiceError => e  
    puts("Couldn't empty and delete bucket #{bucket.name}.\n")  
    puts("  t#{e.code}: #{e.message}")  
    raise  
end
```

- For API details, see [DeleteObjects](#) in *AWS SDK for Ruby API Reference*.

Rust

SDK for Rust

Note

This documentation is for an SDK in preview release. The SDK is subject to change and should not be used in production.

Tip

To learn how to set up and run this example, see [GitHub](#).

```
pub async fn delete_objects(client: &Client, bucket_name: &str) -> Result<(),  
    Error> {  
    let objects = client.list_objects_v2().bucket(bucket_name).send().await?;  
  
    let mut delete_objects: Vec<ObjectIdentifier> = vec![];  
    for obj in objects.contents().unwrap_or_default() {  
        let obj_id = ObjectIdentifier::builder()  
            .set_key(Some(obj.key().unwrap().to_string()))  
            .build();  
        delete_objects.push(obj_id);  
    }  
    client  
        .delete_objects()  
        .bucket(bucket_name)  
        .delete(Delete::builder().set_objects(Some(delete_objects)).build())  
        .send()  
        .await?;  
  
    let objects: ListObjectsV2Output =  
    client.list_objects_v2().bucket(bucket_name).send().await?;  
    match objects.key_count {  
        0 => Ok(()),  
        _ => Err(Error::Unhandled(Box::from(  
            "There were still objects left in the bucket.",  
        ))),  
    }  
}
```

- For API details, see [DeleteObjects](#) in *AWS SDK for Rust API reference*.

Swift

SDK for Swift

Note

This is prerelease documentation for an SDK in preview release. It is subject to change.

Tip

To learn how to set up and run this example, see [GitHub](#).

```
public func deleteObjects(bucket: String, keys: [String]) async throws {
    let input = DeleteObjectsInput(
        bucket: bucket,
        delete: S3ClientTypes.Delete(
            objects: keys.map({ S3ClientTypes.ObjectIdentifier(key: $0) }),
            quiet: true
        )
    )

    do {
        _ = try await client.deleteObjects(input: input)
    } catch {
        throw error
    }
}
```

- For API details, see [DeleteObjects](#) in *AWS SDK for Swift API reference*.

For a complete list of AWS SDK developer guides and code examples, see [Using this service with an AWS SDK \(p. 1185\)](#). This topic also includes information about getting started and details about previous SDK versions.

Delete the lifecycle configuration of an Amazon S3 bucket using an AWS SDK

The following code example shows how to delete the lifecycle configuration of an S3 bucket.

Python

SDK for Python (Boto3)

Tip

To learn how to set up and run this example, see [GitHub](#).

```
class BucketWrapper:
    def __init__(self, bucket):
        self.bucket = bucket
        self.name = bucket.name

    def delete_lifecycle_configuration(self):
        """
        Remove the lifecycle configuration from the specified bucket.
        """
        try:
            self.bucket.LifecycleConfiguration().delete()
            logger.info(
                "Deleted lifecycle configuration for bucket '%s'.",
                self.bucket.name
            )
        except Exception as e:
            logger.error("Failed to delete lifecycle configuration: %s", str(e))
```

```
        except ClientError:
            logger.exception(
                "Couldn't delete lifecycle configuration for bucket '%s'.",
                self.bucket.name)
            raise
```

- For API details, see [DeleteBucketLifecycle in AWS SDK for Python \(Boto3\) API Reference](#).

For a complete list of AWS SDK developer guides and code examples, see [Using this service with an AWS SDK \(p. 1185\)](#). This topic also includes information about getting started and details about previous SDK versions.

Delete the website configuration from an Amazon S3 bucket using an AWS SDK

The following code examples show how to delete the website configuration from an S3 bucket.

C++

SDK for C++

Tip

To learn how to set up and run this example, see [GitHub](#).

```
bool AwsDoc::S3::DeleteBucketWebsite(const Aws::String &bucketName, const
Aws::String &region) {
    // Create the bucket.
    Aws::Client::ClientConfiguration clientConfig;
    if (!region.empty()) {
        clientConfig.region = region;
    }

    Aws::S3::S3Client client(clientConfig);
    Aws::S3::Model::DeleteBucketWebsiteRequest request;
    request.SetBucket(bucketName);

    Aws::S3::Model::DeleteBucketWebsiteOutcome outcome =
        client.DeleteBucketWebsite(request);

    if (!outcome.IsSuccess())
    {
        auto err = outcome.GetError();
        std::cout << "Error: DeleteBucketWebsite: " <<
            err.GetExceptionName() << ":" << err.GetMessage() << std::endl;
        return false;
    }
    else
    {
        std::cout << "Website configuration was removed." << std::endl;
        return true;
    }
}

int main()
{
    //TODO: Change bucket_name to the name of a bucket in your account.
    const Aws::String bucketName = "<Enter bucket name>";
    //TODO: Set to the AWS Region in which the bucket was created.
    const Aws::String region = "us-east-1";
```

```
Aws::SDKOptions options;
Aws::InitAPI(options);

AwsDoc::S3::DeleteBucketWebsite(bucketName, region);

ShutdownAPI(options);
}
```

- For API details, see [DeleteBucketWebsite in AWS SDK for C++ API Reference](#).

Java

SDK for Java 2.x

Tip

To learn how to set up and run this example, see [GitHub](#).

```
public static void deleteBucketWebsiteConfig(S3Client s3, String bucketName) {

    DeleteBucketWebsiteRequest delReq = DeleteBucketWebsiteRequest.builder()
        .bucket(bucketName)
        .build();

    try {
        s3.deleteBucketWebsite(delReq);

    } catch (S3Exception e) {
        System.err.println(e.awsErrorDetails().errorMessage());
        System.out.println("Failed to delete website configuration!");
        System.exit(1);
    }
}
```

- For API details, see [DeleteBucketWebsite in AWS SDK for Java 2.x API Reference](#).

JavaScript

SDK for JavaScript V3

Tip

To learn how to set up and run this example, see [GitHub](#).

Create the client.

```
// Create service client module using ES6 syntax.
import { S3Client } from "@aws-sdk/client-s3";
// Set the AWS Region.
const REGION = "us-east-1";
// Create an Amazon S3 service client object.
const s3Client = new S3Client({ region: REGION });
export { s3Client };
```

Delete the website configuration from the bucket.

```
// Import required AWS SDK clients and commands for Node.js.

import { DeleteBucketWebsiteCommand } from "@aws-sdk/client-s3";
import { s3Client } from "./libs/s3Client.js"; // Helper function that creates an
Amazon S3 service client module.

// Create the parameters for calling
export const bucketParams = { Bucket: "BUCKET_NAME" };

export const run = async () => {
  try {
    const data = await s3Client.send(new DeleteBucketWebsiteCommand(bucketParams));
    return data; // For unit tests.
    console.log("Success", data);
  } catch (err) {
    console.log("Error", err);
  }
};
run();
```

- For more information, see [AWS SDK for JavaScript Developer Guide](#).
- For API details, see [DeleteBucketWebsite in AWS SDK for JavaScript API Reference](#).

For a complete list of AWS SDK developer guides and code examples, see [Using this service with an AWS SDK \(p. 1185\)](#). This topic also includes information about getting started and details about previous SDK versions.

Determine the existence and content type of an object in an Amazon S3 bucket using an AWS SDK

The following code examples show how to determine the existence and content type of an object in an S3 bucket.

Java

SDK for Java 2.x

Tip

To learn how to set up and run this example, see [GitHub](#).

```
public static void getContentType (S3Client s3, String bucketName, String
keyName) {

    try {
        HeadObjectRequest objectRequest = HeadObjectRequest.builder()
            .key(keyName)
            .bucket(bucketName)
            .build();

        HeadObjectResponse objectHead = s3.headObject(objectRequest);
        String type = objectHead.contentType();
        System.out.println("The object content type is "+type);

    } catch (S3Exception e) {
        System.err.println(e.awsErrorDetails().errorMessage());
        System.exit(1);
    }
}
```

- For API details, see [HeadObject in AWS SDK for Java 2.x API Reference](#).

Ruby

SDK for Ruby

Tip

To learn how to set up and run this example, see [GitHub](#).

```
require "aws-sdk-s3"

# Wraps Amazon S3 object actions.
class ObjectExistsWrapper
  attr_reader :object

  # @param object [Aws::S3::Object] An Amazon S3 object.
  def initialize(object)
    @object = object
  end

  # Checks whether the object exists.
  #
  # @return [Boolean] True if the object exists; otherwise false.
  def exists?
    @object.exists?
  rescue Aws::Errors::ServiceError => e
    puts "Couldn't check existence of object #{@object.bucket.name}:#{@object.key}. Here's why: #{e.message}"
    false
  end
end

# Replace bucket name and object key with an existing bucket and object that you own.
def run_demo
  bucket_name = "doc-example-bucket"
  object_key = "my-object.txt"

  wrapper = ObjectExistsWrapper.new(Aws::S3::Object.new(bucket_name, object_key))
  exists = wrapper.exists?

  puts "Object #{object_key} #{exists ? 'does' : 'does not'} exist."
end

run_demo if $PROGRAM_NAME == __FILE__
```

- For API details, see [HeadObject in AWS SDK for Ruby API Reference](#).

For a complete list of AWS SDK developer guides and code examples, see [Using this service with an AWS SDK \(p. 1185\)](#). This topic also includes information about getting started and details about previous SDK versions.

Determine the existence of an Amazon S3 bucket using an AWS SDK

The following code example shows how to determine the existence of an S3 bucket.

Python

SDK for Python (Boto3)

Tip

To learn how to set up and run this example, see [GitHub](#).

```
class BucketWrapper:
    def __init__(self, bucket):
        self.bucket = bucket
        self.name = bucket.name

    def exists(self):
        """
        Determine whether the bucket exists and you have access to it.

        :return: True when the bucket exists; otherwise, False.
        """
        try:
            self.bucket.meta.client.head_bucket(Bucket=self.bucket.name)
            logger.info("Bucket %s exists.", self.bucket.name)
            exists = True
        except ClientError:
            logger.warning(
                "Bucket %s doesn't exist or you don't have access to it.",
                self.bucket.name)
            exists = False
        return exists
```

- For API details, see [HeadBucket in AWS SDK for Python \(Boto3\) API Reference](#).

For a complete list of AWS SDK developer guides and code examples, see [Using this service with an AWS SDK \(p. 1185\)](#). This topic also includes information about getting started and details about previous SDK versions.

Get CORS rules for an Amazon S3 bucket using an AWS SDK

The following code examples show how to get cross-origin resource sharing (CORS) rules for an S3 bucket.

JavaScript

SDK for JavaScript V3

Tip

To learn how to set up and run this example, see [GitHub](#).

Create the client.

```
// Create service client module using ES6 syntax.
import { S3Client } from "@aws-sdk/client-s3";
// Set the AWS Region.
const REGION = "us-east-1";
// Create an Amazon S3 service client object.
const s3Client = new S3Client({ region: REGION });
export { s3Client };
```

Get the CORS policy for the bucket.

```
// Import required AWS SDK clients and commands for Node.js.
import { GetBucketCorsCommand } from "@aws-sdk/client-s3";
import { s3Client } from "./libs/s3Client.js"; // Helper function that creates an
// Amazon S3 service client module.

// Create the parameters for calling
export const bucketParams = { Bucket: "BUCKET_NAME" };

export const run = async () => {
    try {
        const data = await s3Client.send(new GetBucketCorsCommand(bucketParams));
        console.log("Success", JSON.stringify(data.CORSRules));
        return data; // For unit tests.
    } catch (err) {
        console.log("Error", err);
    }
};

run();
```

- For more information, see [AWS SDK for JavaScript Developer Guide](#).
- For API details, see [GetBucketCors](#) in [AWS SDK for JavaScript API Reference](#).

Python

SDK for Python (Boto3)

Tip

To learn how to set up and run this example, see [GitHub](#).

```
class BucketWrapper:
    def __init__(self, bucket):
        self.bucket = bucket
        self.name = bucket.name

    def get_cors(self):
        """
        Get the CORS rules for the bucket.

        :return The CORS rules for the specified bucket.
        """
        try:
            cors = self.bucket.Cors()
            logger.info(
                "Got CORS rules %s for bucket '%s'.", cors.cors_rules,
                self.bucket.name)
        except ClientError:
            logger.exception(("Couldn't get CORS for bucket %s.",
                self.bucket.name))
            raise
        else:
            return cors
```

- For API details, see [GetBucketCors](#) in [AWS SDK for Python \(Boto3\) API Reference](#).

Ruby

SDK for Ruby

Tip

To learn how to set up and run this example, see [GitHub](#).

```
require "aws-sdk-s3"

# Wraps Amazon S3 bucket CORS configuration.
class BucketCorsWrapper
    attr_reader :bucket_cors

    # @param bucket_cors [Aws::S3::BucketCors] A bucket CORS object configured with
    # an existing bucket.
    def initialize(bucket_cors)
        @bucket_cors = bucket_cors
    end

    # Gets the CORS configuration of a bucket.
    #
    # @return [Aws::S3::Type::GetBucketCorsOutput, nil] The current CORS
    # configuration for the bucket.
    def get_cors
        @bucket_cors.data
        rescue Aws::Errors::ServiceError => e
            puts "Couldn't get CORS configuration for #{@bucket_cors.bucket.name}. Here's
why: #{e.message}"
            nil
    end
end
```

- For API details, see [GetBucketCors](#) in *AWS SDK for Ruby API Reference*.

For a complete list of AWS SDK developer guides and code examples, see [Using this service with an AWS SDK \(p. 1185\)](#). This topic also includes information about getting started and details about previous SDK versions.

Get an object from an Amazon S3 bucket using an AWS SDK

The following code examples show how to read data from an object in an S3 bucket.

.NET

AWS SDK for .NET

Tip

To learn how to set up and run this example, see [GitHub](#).

```
/// <summary>
/// Shows how to download an object from an Amazon S3 bucket to the
/// local computer.
/// </summary>
/// <param name="client">An initialized Amazon S3 client object.</param>
```

```
/// <param name="bucketName">The name of the bucket where the object is  
/// currently stored.</param>  
/// <param name="objectName">The name of the object to download.</param>  
/// <param name="filePath">The path, including filename, where the  
/// downloaded object will be stored.</param>  
/// <returns>A boolean value indicating the success or failure of the  
/// download process.</returns>  
public static async Task<bool> DownloadObjectFromBucketAsync(  
    IAmazonS3 client,  
    string bucketName,  
    string objectName,  
    string filePath)  
{  
    // Create a GetObject request  
    var request = new GetObjectRequest  
    {  
        BucketName = bucketName,  
        Key = objectName,  
    };  
  
    // Issue request and remember to dispose of the response  
    // using GetObjectResponse response = await  
    client.GetObjectAsync(request);  
  
    try  
    {  
        // Save object to local file  
        await response.WriteResponseStreamToFileAsync($"{filePath}\\"  
        $"{objectName}", true, System.Threading.CancellationToken.None);  
        return response.HttpStatusCode == System.Net.HttpStatusCode.OK;  
    }  
    catch (AmazonS3Exception ex)  
    {  
        Console.WriteLine($"Error saving {objectName}: {ex.Message}");  
        return false;  
    }  
}
```

- For API details, see [GetObject](#) in *AWS SDK for .NET API Reference*.

C++

SDK for C++

Tip

To learn how to set up and run this example, see [GitHub](#).

```
bool AwsDoc::S3::GetObject(const Aws::String& objectKey,  
    const Aws::String& fromBucket, const Aws::String& region)  
{  
    Aws::Client::ClientConfiguration config;  
  
    if (!region.empty())  
    {  
        config.region = region;  
    }  
  
    Aws::S3::S3Client s3_client(config);  
  
    Aws::S3::Model::GetObjectRequest object_request;  
    object_request.SetBucket(fromBucket);
```

```
object_request.SetKey(objectKey);

Aws::S3::Model::GetObjectOutcome get_object_outcome =
    s3_client.GetObject(object_request);

if (get_object_outcome.IsSuccess())
{
    auto& retrieved_file = get_object_outcome.GetResultWithOwnership().GetBody();

    // Print a beginning portion of the text file.
    std::cout << "Beginning of file contents:\n";
    char file_data[255] = { 0 };
    retrieved_file.getline(file_data, 254);
    std::cout << file_data << std::endl;

    return true;
}
else
{
    auto err = get_object_outcome.GetError();
    std::cout << "Error: GetObject: " <<
        err.GetExceptionName() << ":" << err.GetMessage() << std::endl;

    return false;
}

int main()
{
    Aws::SDKOptions options;
    Aws::InitAPI(options);
    {
        //TODO: Change bucket_name to the name of a bucket in your account.
        const Aws::String bucket_name = "<Enter bucket name>";

        //TODO: The bucket "DOC-EXAMPLE-BUCKET" must have been created and
        //previously loaded with "my-file.txt".
        //See create_bucket.cpp and put_object.cpp to create a bucket and load an
        //object into that bucket.
        const Aws::String object_name = "<Enter object name>";

        //TODO: Set to the AWS Region in which the bucket was created.
        const Aws::String region = "us-east-1";

        if (!AwsDoc::S3::GetObject(object_name, bucket_name, region))
        {
            return 1;
        }
    }
    Aws::ShutdownAPI(options);

    return 0;
}
```

- For API details, see [GetObject](#) in *AWS SDK for C++ API Reference*.

Java

SDK for Java 2.x

Tip

To learn how to set up and run this example, see [GitHub](#).

Read data as a byte array.

```
public static void getObjectBytes (S3Client s3, String bucketName, String keyName, String path) {  
  
    try {  
        GetObjectRequest objectRequest = GetObjectRequest  
            .builder()  
            .key(keyName)  
            .bucket(bucketName)  
            .build();  
  
        ResponseBytes<GetObjectResponse> objectBytes =  
            s3.getObjectAsBytes(objectRequest);  
        byte[] data = objectBytes.asByteArray();  
  
        // Write the data to a local file.  
        File myFile = new File(path );  
        OutputStream os = new FileOutputStream(myFile);  
        os.write(data);  
        System.out.println("Successfully obtained bytes from an S3 object");  
        os.close();  
  
    } catch (IOException ex) {  
        ex.printStackTrace();  
    } catch (S3Exception e) {  
        System.err.println(e.awsErrorDetails().errorMessage());  
        System.exit(1);  
    }  
}
```

Read tags that belong to an object.

```
public static void listTags(S3Client s3, String bucketName, String keyName) {  
  
    try {  
        GetObjectTaggingRequest getTaggingRequest = GetObjectTaggingRequest  
            .builder()  
            .key(keyName)  
            .bucket(bucketName)  
            .build();  
  
        GetObjectTaggingResponse tags = s3.getObjectTagging(getTaggingRequest);  
        List<Tag> tagSet= tags.tagSet();  
        for (Tag tag : tagSet) {  
            System.out.println(tag.key());  
            System.out.println(tag.value());  
        }  
  
    } catch (S3Exception e) {  
        System.err.println(e.awsErrorDetails().errorMessage());  
        System.exit(1);  
    }  
}
```

Get a URL for an object.

```
public static void getURL(S3Client s3, String bucketName, String keyName ) {  
  
    try {  
        GetUrlRequest request = GetUrlRequest.builder()
```

```
        .bucket(bucketName)
        .key(keyName)
        .build();

    URL url = s3.utilities().getUrl(request);
    System.out.println("The URL for " + keyName + " is " + url);

} catch (S3Exception e) {
    System.err.println(e.awsErrorDetails().errorMessage());
    System.exit(1);
}
}
```

Get an object by using the S3Presigner client object.

```
public static void getPresignedUrl(S3Presigner presigner, String bucketName,
String keyName ) {

    try {
        GetObjectRequest getObjectRequest = GetObjectRequest.builder()
            .bucket(bucketName)
            .key(keyName)
            .build();

        GetObjectPresignRequest getObjectPresignRequest =
GetObjectPresignRequest.builder()
            .signatureDuration(Duration.ofMinutes(60))
            .getObjectRequest(getObjectRequest)
            .build();

        PresignedGetObjectRequest presignedGetObjectRequest =
presigner.presignGetObject(getObjectPresignRequest);
        String theUrl = presignedGetObjectRequest.url().toString();
        System.out.println("Presigned URL: " + theUrl);
        HttpURLConnection connection = (HttpURLConnection)
presignedGetObjectRequest.url().openConnection();
        presignedGetObjectRequest.httpRequest().headers().forEach((header,
values) -> {
            values.forEach(value -> {
                connection.addRequestProperty(header, value);
            });
        });

        // Send any request payload that the service needs (not needed when
isBrowserExecutable is true).
        if (presignedGetObjectRequest.signedPayload().isPresent()) {
            connection.setDoOutput(true);

            try (InputStream signedPayload =
presignedGetObjectRequest.signedPayload().get().asInputStream();
                OutputStream httpOutputStream = connection.getOutputStream()) {
                IoUtils.copy(signedPayload, httpOutputStream);
            }
        }

        // Download the result of executing the request.
        try (InputStream content = connection.getInputStream()) {
            System.out.println("Service returned response: ");
            IoUtils.copy(content, System.out);
        }

    } catch (S3Exception | IOException e) {
        e.printStackTrace();
    }
}
```

```
}
```

- For API details, see [GetObject](#) in *AWS SDK for Java 2.x API Reference*.

JavaScript

SDK for JavaScript V3

Tip

To learn how to set up and run this example, see [GitHub](#).

Create the client.

```
// Create service client module using ES6 syntax.  
import { S3Client } from "@aws-sdk/client-s3";  
// Set the AWS Region.  
const REGION = "us-east-1";  
// Create an Amazon S3 service client object.  
const s3Client = new S3Client({ region: REGION });  
export { s3Client };
```

Download the object.

```
// Import required AWS SDK clients and commands for Node.js.  
import { GetObjectCommand } from "@aws-sdk/client-s3";  
import { s3Client } from "./libs/s3Client.js"; // Helper function that creates an  
Amazon S3 service client module.  
  
export const bucketParams = {  
    Bucket: "BUCKET_NAME",  
    Key: "KEY",  
};  
  
export const run = async () => {  
    try {  
        // Create a helper function to convert a ReadableStream to a string.  
        const streamToString = (stream) =>  
            new Promise((resolve, reject) => {  
                const chunks = [];  
                stream.on("data", (chunk) => chunks.push(chunk));  
                stream.on("error", reject);  
                stream.on("end", () => resolve(Buffer.concat(chunks).toString("utf8")));  
            });  
  
        // Get the object} from the Amazon S3 bucket. It is returned as a  
ReadableStream.  
        const data = await s3Client.send(new GetObjectCommand(bucketParams));  
        return data; // For unit tests.  
        // Convert the ReadableStream to a string.  
        const bodyContents = await streamToString(data.Body);  
        console.log(bodyContents);  
        return bodyContents;  
    } catch (err) {  
        console.log("Error", err);  
    }  
};  
run();
```

- For more information, see [AWS SDK for JavaScript Developer Guide](#).

- For API details, see [GetObject](#) in *AWS SDK for JavaScript API Reference*.

Kotlin

SDK for Kotlin

Note

This is prerelease documentation for a feature in preview release. It is subject to change.

Tip

To learn how to set up and run this example, see [GitHub](#).

```
suspend fun getObjectBytes(bucketName: String, keyName: String, path: String) {  
  
    val request = GetObjectRequest {  
        key = keyName  
        bucket = bucketName  
    }  
  
    S3Client { region = "us-east-1" }.use { s3 ->  
        s3.getObject(request) { resp ->  
            val myFile = File(path)  
            resp.body?.writeTo(myFile)  
            println("Successfully read $keyName from $bucketName")  
        }  
    }  
}
```

- For API details, see [GetObject](#) in *AWS SDK for Kotlin API reference*.

PHP

SDK for PHP

Tip

To learn how to set up and run this example, see [GitHub](#).

Get an object.

```
$s3client = new Aws\S3\S3Client(['region' => 'us-west-2', 'version' => 'latest']);  
  
try {  
    $file = $s3client->getObject([  
        'Bucket' => $bucket_name,  
        'Key' => $file_name,  
    ]);  
    $body = $file->get('Body');  
    $body->rewind();  
    echo "Downloaded the file and it begins with: {$body->read(26)}.\n";  
} catch (Exception $exception) {  
    echo "Failed to download $file_name from $bucket_name with error: " .  
    $exception->getMessage();  
    exit("Please fix error with file downloading before continuing.");  
}
```

- For API details, see [GetObject](#) in *AWS SDK for PHP API Reference*.

Python

SDK for Python (Boto3)

Tip

To learn how to set up and run this example, see [GitHub](#).

```
class ObjectWrapper:
    def __init__(self, s3_object):
        self.object = s3_object
        self.key = self.object.key

    def get(self):
        """
        Gets the object.

        :return: The object data in bytes.
        """
        try:
            body = self.object.get()['Body'].read()
            logger.info(
                "Got object '%s' from bucket '%s'.",
                self.object.key, self.object.bucket_name)
        except ClientError:
            logger.exception(
                "Couldn't get object '%s' from bucket '%s'.",
                self.object.key, self.object.bucket_name)
            raise
        else:
            return body
```

- For API details, see [GetObject](#) in *AWS SDK for Python (Boto3) API Reference*.

Ruby

SDK for Ruby

Tip

To learn how to set up and run this example, see [GitHub](#).

Get an object.

```
require "aws-sdk-s3"

# Wraps Amazon S3 object actions.
class ObjectGetWrapper
  attr_reader :object

  # @param object [Aws::S3::Object] An existing Amazon S3 object.
  def initialize(object)
    @object = object
  end

  # Gets the object directly to a file.
  #
  # @param target_path [String] The path to the file where the object is
  # downloaded.
  # @return [Aws::S3::Types::GetObjectOutput, nil] The retrieved object data if
  # successful; otherwise nil.
  def get_object(target_path)
```

```
    @object.get(response_target: target_path)
rescue Aws::Errors::ServiceError => e
    puts "Couldn't get object #{@object.key}. Here's why: #{e.message}"
end
end

# Replace bucket name and object key with an existing bucket and object that you
# own.
def run_demo
    bucket_name = "doc-example-bucket"
    object_key = "my-object.txt"
    target_path = "my-object-as-file.txt"

    wrapper = ObjectGetWrapper.new(Aws::S3::Object.new(bucket_name, object_key))
    obj_data = wrapper.get_object(target_path)
    return unless obj_data

    puts "Object #{object_key} (#{obj_data.content_length} bytes) downloaded to
    #{target_path}."
end

run_demo if $PROGRAM_NAME == __FILE__
```

Get an object and report its server-side encryption state.

```
require "aws-sdk-s3"

# Wraps Amazon S3 object actions.
class ObjectGetEncryptionWrapper
    attr_reader :object

    # @param object [Aws::S3::Object] An existing Amazon S3 object.
    def initialize(object)
        @object = object
    end

    # Gets the object into memory.
    #
    # @return [Aws::S3::Types::GetObjectOutput, nil] The retrieved object data if
    # successful; otherwise nil.
    def get_object
        @object.get
        rescue Aws::Errors::ServiceError => e
            puts "Couldn't get object #{@object.key}. Here's why: #{e.message}"
        end
    end

    # Replace bucket name and object key with an existing bucket and object that you
    # own.
    def run_demo
        bucket_name = "doc-example-bucket"
        object_key = "my-object.txt"

        wrapper = ObjectGetEncryptionWrapper.new(Aws::S3::Object.new(bucket_name,
        object_key))
        obj_data = wrapper.get_object
        return unless obj_data

        encryption = obj_data.server_side_encryption.nil? ? "no" :
        obj_data.server_side_encryption
        puts "Object #{object_key} uses #{encryption} encryption."
    end

run_demo if $PROGRAM_NAME == __FILE__
```

- For API details, see [GetObject](#) in *AWS SDK for Ruby API Reference*.

Rust

SDK for Rust

Note

This documentation is for an SDK in preview release. The SDK is subject to change and should not be used in production.

Tip

To learn how to set up and run this example, see [GitHub](#).

```
pub async fn download_object(client: &Client, bucket_name: &str, key: &str) ->
Result<(), Error> {
    let resp = client
        .get_object()
        .bucket(bucket_name)
        .key(key)
        .send()
        .await?;
    let data = resp.body.collect().await;
    println!(
        "Data from downloaded object: {:+?}",
        data.unwrap().into_bytes().slice(0..20)
    );
    Ok(())
}
```

- For API details, see [GetObject](#) in *AWS SDK for Rust API reference*.

Swift

SDK for Swift

Note

This is prerelease documentation for an SDK in preview release. It is subject to change.

Tip

To learn how to set up and run this example, see [GitHub](#).

Download an object from a bucket to a local file.

```
public func downloadFile(bucket: String, key: String, to: String) async throws {
    let urlString = URL(fileURLWithPath: to).appendingPathComponent(key)

    let input = GetObjectInput(
        bucket: bucket,
        key: key
    )
    let output = try await client.getObject(input: input)

    // Get the data stream object. Return immediately if there isn't one.
    guard let body = output.body else {
        return
    }
    // Write the body to the file.
    try await body.write(to: to)
}
```

```
        }
        let data = body.toBytes().toData()
        try data.write(to: urlString)
    }
}
```

Read an object into a Swift Data object.

```
public func readFile(bucket: String, key: String) async throws -> Data {
    let input = GetObjectInput(
        bucket: bucket,
        key: key
    )
    let output = try await client.getObject(input: input)

    // Get the stream and return its contents in a `Data` object. If
    // there is no stream, return an empty `Data` object instead.
    guard let body = output.body else {
        return "".data(using: .utf8)!
    }
    let data = body.toBytes().toData()
    return data
}
```

- For API details, see [GetObject](#) in [AWS SDK for Swift API reference](#).

For a complete list of AWS SDK developer guides and code examples, see [Using this service with an AWS SDK \(p. 1185\)](#). This topic also includes information about getting started and details about previous SDK versions.

Get the ACL of an Amazon S3 bucket using an AWS SDK

The following code examples show how to get the access control list (ACL) of an S3 bucket.

Go

SDK for Go V2

Tip

To learn how to set up and run this example, see [GitHub](#).

```
package main

import (
    "context"
    "flag"
    "fmt"

    "github.com/aws/aws-sdk-go-v2/config"
    "github.com/aws/aws-sdk-go-v2/service/s3"
)

// S3GetBucketAclAPI defines the interface for the GetBucketAcl function.
// We use this interface to test the function using a mocked service.
type S3GetBucketAclAPI interface {
    GetBucketAcl(ctx context.Context,
        params *s3.GetBucketAclInput,
```

```
    optFns ...func(*s3.Options)) (*s3.GetBucketAclOutput, error)
}

// FindBucketAcl retrieves the access control list (ACL) for an Amazon Simple
// Storage Service (Amazon S3) bucket.
// Inputs:
//   c is the context of the method call, which includes the AWS Region
//   api is the interface that defines the method call
//   input defines the input arguments to the service call.
// Output:
//   If success, a GetBucketAclOutput object containing the result of the service
//   call and nil
//   Otherwise, nil and an error from the call to GetBucketAcl
func FindBucketAcl(c context.Context, api S3GetBucketAclAPI, input
*s3.GetBucketAclInput) (*s3.GetBucketAclOutput, error) {
    return api.GetBucketAcl(c, input)
}

func main() {
    bucket := flag.String("b", "", "The bucket for which the ACL is returned")
    flag.Parse()

    if *bucket == "" {
        fmt.Println("You must supply a bucket name (-b BUCKET)")
        return
    }

    cfg, err := config.LoadDefaultConfig(context.TODO())
    if err != nil {
        panic("configuration error, " + err.Error())
    }

    client := s3.NewFromConfig(cfg)

    input := &s3.GetBucketAclInput{
        Bucket: bucket,
    }

    result, err := FindBucketAcl(context.TODO(), client, input)
    if err != nil {
        fmt.Println("Got an error retrieving ACL for " + *bucket)
        return
    }

    fmt.Println("Owner:", *result.Owner.DisplayName)
    fmt.Println("")
    fmt.Println("Grants")

    for _, g := range result.Grants {
        // If we add a canned ACL, the name is nil
        if g.Grantee.DisplayName == nil {
            fmt.Println("  Grantee:  EVERYONE")
        } else {
            fmt.Println("  Grantee:  ", *g.Grantee.DisplayName)
        }

        fmt.Println("  Type:      ", string(g.Grantee.Type))
        fmt.Println("  Permission: ", string(g.Permission))
        fmt.Println("")
    }
}
```

- For API details, see [GetBucketAcl](#) in [AWS SDK for Go API Reference](#).

Java

SDK for Java 2.x

Tip

To learn how to set up and run this example, see [GitHub](#).

```
public static String getBucketACL(S3Client s3, String objectKey, String
bucketName) {

    try {
        GetObjectAclRequest aclReq = GetObjectAclRequest.builder()
            .bucket(bucketName)
            .key(objectKey)
            .build();

        GetObjectAclResponse aclRes = s3.getObjectAcl(aclReq);
        List<Grant> grants = aclRes.grants();
        String grantee = "";
        for (Grant grant : grants) {
            System.out.format(" %s: %s\n", grant.grantee().id(),
grant.permission());
            grantee = grant.grantee().id();
        }

        return grantee;
    } catch (S3Exception e) {
        System.err.println(e.awsErrorDetails().errorMessage());
        System.exit(1);
    }

    return "";
}
```

- For API details, see [GetBucketAcl](#) in *AWS SDK for Java 2.x API Reference*.

JavaScript

SDK for JavaScript V3

Tip

To learn how to set up and run this example, see [GitHub](#).

Create the client.

```
// Create service client module using ES6 syntax.
import { S3Client } from "@aws-sdk/client-s3";
// Set the AWS Region.
const REGION = "us-east-1";
// Create an Amazon S3 service client object.
const s3Client = new S3Client({ region: REGION });
export { s3Client };
```

Get the ACL permissions.

```
// Import required AWS SDK clients and commands for Node.js.
import { GetBucketAclCommand } from "@aws-sdk/client-s3";
import { s3Client } from "./libs/s3Client.js"; // Helper function that creates an
Amazon S3 service client module.
```

```
// Create the parameters.
export const bucketParams = { Bucket: "BUCKET_NAME" };

export const run = async () => {
  try {
    const data = await s3Client.send(new GetBucketAclCommand(bucketParams));
    console.log("Success", data.Grants);
    return data; // For unit tests.
  } catch (err) {
    console.log("Error", err);
  }
};
run();
```

- For more information, see [AWS SDK for JavaScript Developer Guide](#).
- For API details, see [GetBucketAcl](#) in [AWS SDK for JavaScript API Reference](#).

Python

SDK for Python (Boto3)

Tip

To learn how to set up and run this example, see [GitHub](#).

```
class BucketWrapper:
    def __init__(self, bucket):
        self.bucket = bucket
        self.name = bucket.name

    def get_acl(self):
        """
        Get the ACL of the bucket.

        :return: The ACL of the bucket.
        """
        try:
            acl = self.bucket.Acl()
            logger.info(
                "Got ACL for bucket %s. Owner is %s.", self.bucket.name, acl.owner)
        except ClientError:
            logger.exception("Couldn't get ACL for bucket %s.", self.bucket.name)
            raise
        else:
            return acl
```

- For API details, see [GetBucketAcl](#) in [AWS SDK for Python \(Boto3\) API Reference](#).

For a complete list of AWS SDK developer guides and code examples, see [Using this service with an AWS SDK \(p. 1185\)](#). This topic also includes information about getting started and details about previous SDK versions.

Get the ACL of an Amazon S3 object using an AWS SDK

The following code examples show how to get the access control list (ACL) of an S3 object.

C++

SDK for C++

Tip

To learn how to set up and run this example, see [GitHub](#).

```
bool AwsDoc::S3::GetBucketAcl(const Aws::String& bucketName, const Aws::String&
    region)
{
    Aws::Client::ClientConfiguration config;
    config.region = region;

    Aws::S3::S3Client s3_client(config);

    Aws::S3::Model::GetBucketAclRequest request;
    request.SetBucket(bucketName);

    Aws::S3::Model::GetBucketAclOutcome outcome =
        s3_client.GetBucketAcl(request);

    if (outcome.IsSuccess())
    {
        Aws::Vector<Aws::S3::Model::Grant> grants =
            outcome.GetResult().GetGrants();

        for (auto it = grants.begin(); it != grants.end(); it++)
        {
            Aws::S3::Model::Grant grant = *it;
            Aws::S3::Model::Grantee grantee = grant.GetGrantee();

            std::cout << "For bucket " << bucketName << ":" 
                << std::endl << std::endl;

            if (grantee.TypeHasBeenSet())
            {
                std::cout << "Type:          "
                    << GetGranteeTypeString(grantee.GetType()) << std::endl;
            }

            if (grantee.DisplayNameHasBeenSet())
            {
                std::cout << "Display name:   "
                    << grantee.GetDisplayName() << std::endl;
            }

            if (grantee.EmailAddressHasBeenSet())
            {
                std::cout << "Email address:  "
                    << grantee.GetEmailAddress() << std::endl;
            }

            if (grantee.IDHasBeenSet())
            {
                std::cout << "ID:           "
                    << grantee.GetID() << std::endl;
            }

            if (grantee.URIHasBeenSet())
            {
                std::cout << "URI:          "
                    << grantee.GetURI() << std::endl;
            }
        }
    }
}
```

```
        std::cout << "Permission:      " <<
            GetPermissionString("bucket", grant.GetPermission()) <<
            std::endl << std::endl;
    }
} else
{
    auto err = outcome.GetError();
    std::cout << "Error: GetBucketAcl: "
        << err.GetExceptionName() << ":" << err.GetMessage() << std::endl;

    return false;
}

return true;
}
```

- For API details, see [GetObjectAcl](#) in *AWS SDK for C++ API Reference*.

Go

SDK for Go V2

Tip

To learn how to set up and run this example, see [GitHub](#).

```
package main

import (
    "context"
    "flag"
    "fmt"

    "github.com/aws/aws-sdk-go-v2/config"
    "github.com/aws/aws-sdk-go-v2/service/s3"
)

// S3GetObjectAclAPI defines the interface for the GetObjectAcl function.
// We use this interface to test the function using a mocked service.
type S3GetObjectAclAPI interface {
    GetObjectAcl(ctx context.Context,
        params *s3.GetObjectAclInput,
        optFns ...func(*s3.Options)) (*s3.GetObjectAclOutput, error)
}

// FindObjectAcl gets the access control list (ACL) for an Amazon Simple Storage
// Service (Amazon S3) bucket object
// Inputs:
//     c is the context of the method call, which includes the AWS Region
//     api is the interface that defines the method call
//     input defines the input arguments to the service call.
// Output:
//     If success, a GetObjectAclOutput object containing the result of the service
//     call and nil
//     Otherwise, nil and an error from the call to GetObjectAcl
func FindObjectAcl(c context.Context, api S3GetObjectAclAPI, input
    *s3.GetObjectAclInput) (*s3.GetObjectAclOutput, error) {
    return api.GetObjectAcl(c, input)
}

func main() {
    bucket := flag.String("b", "", "The bucket containing the object")
```

```
objectName := flag.String("o", "", "The bucket object to get ACL from")
flag.Parse()

if *bucket == "" || *objectName == "" {
    fmt.Println("You must supply a bucket (-b BUCKET) and object (-o OBJECT)")
    return
}

cfg, err := config.LoadDefaultConfig(context.TODO())
if err != nil {
    panic("configuration error, " + err.Error())
}

client := s3.NewFromConfig(cfg)

input := &s3.GetObjectAclInput{
    Bucket: bucket,
    Key:    objectName,
}

result, err := FindObjectAcl(context.TODO(), client, input)
if err != nil {
    fmt.Println("Got an error getting ACL for " + *objectName)
    return
}

fmt.Println("Owner:", *result.Owner.DisplayName)
fmt.Println("")
fmt.Println("Grants")

for _, g := range result.Grants {
    fmt.Println("  Grantee: ", *g.Grantee.DisplayName)
    fmt.Println("  Type:     ", string(g.Grantee.Type))
    fmt.Println("  Permission: ", string(g.Permission))
    fmt.Println("")
}
}
```

- For API details, see [GetObjectAcl](#) in [AWS SDK for Go API Reference](#).

Kotlin

SDK for Kotlin

Note

This is prerelease documentation for a feature in preview release. It is subject to change.

Tip

To learn how to set up and run this example, see [GitHub](#).

```
suspend fun getBucketACL(objectKey: String, bucketName: String) {

    val request = GetObjectAclRequest {
        bucket = bucketName
        key = objectKey
    }

    S3Client { region = "us-east-1" }.use { s3 ->
        val response = s3.getObjectAcl(request)
        response.grants?.forEach { grant ->
```

```
        println("Grant permission is ${grant.permission}")
    }
}
```

- For API details, see [GetObjectAcl](#) in *AWS SDK for Kotlin API reference*.

Python

SDK for Python (Boto3)

Tip

To learn how to set up and run this example, see [GitHub](#).

```
class ObjectWrapper:
    def __init__(self, s3_object):
        self.object = s3_object
        self.key = self.object.key

    def get_acl(self):
        """
        Gets the ACL of the object.

        :return: The ACL of the object.
        """
        try:
            acl = self.object.Acl()
            logger.info(
                "Got ACL for object %s owned by %s.",
                self.object.key, acl.owner['DisplayName'])
        except ClientError:
            logger.exception("Couldn't get ACL for object %s.", self.object.key)
            raise
        else:
            return acl
```

- For API details, see [GetObjectAcl](#) in *AWS SDK for Python (Boto3) API Reference*.

For a complete list of AWS SDK developer guides and code examples, see [Using this service with an AWS SDK \(p. 1185\)](#). This topic also includes information about getting started and details about previous SDK versions.

Get the Region where the Amazon S3 bucket resides using an AWS SDK

The following code example shows how to get the Region location for an S3 bucket.

Rust

SDK for Rust

Note

This documentation is for an SDK in preview release. The SDK is subject to change and should not be used in production.

Tip

To learn how to set up and run this example, see [GitHub](#).

```
async fn show_buckets(strict: bool, client: &Client, region: &str) -> Result<(), Error> {
    let resp = client.list_buckets().send().await?;
    let buckets = resp.buckets().unwrap_or_default();
    let num_buckets = buckets.len();

    let mut in_region = 0;

    for bucket in buckets {
        if strict {
            let r = client
                .get_bucket_location()
                .bucket(bucket.name()).unwrap_or_default()
                .send()
                .await?;

            if r.location_constraint().unwrap().as_ref() == region {
                println!("{}", bucket.name().unwrap_or_default());
                in_region += 1;
            }
        } else {
            println!("{}", bucket.name().unwrap_or_default());
        }
    }

    println!();
    if strict {
        println!(
            "Found {} buckets in the {} region out of a total of {} buckets.",
            in_region, region, num_buckets
        );
    } else {
        println!("Found {} buckets in all regions.", num_buckets);
    }

    Ok(())
}
```

- For API details, see [GetBucketLocation](#) in *AWS SDK for Rust API reference*.

For a complete list of AWS SDK developer guides and code examples, see [Using this service with an AWS SDK \(p. 1185\)](#). This topic also includes information about getting started and details about previous SDK versions.

Get the lifecycle configuration of an Amazon S3 bucket using an AWS SDK

The following code example shows how to get the lifecycle configuration of an S3 bucket.

Python

SDK for Python (Boto3)

Tip

To learn how to set up and run this example, see [GitHub](#).

```
class BucketWrapper:
```

```
def __init__(self, bucket):
    self.bucket = bucket
    self.name = bucket.name

def get_lifecycle_configuration(self):
    """
    Get the lifecycle configuration of the bucket.

    :return: The lifecycle rules of the specified bucket.
    """
    try:
        config = self.bucket.LifecycleConfiguration()
        logger.info(
            "Got lifecycle rules %s for bucket '%s'.", config.rules,
            self.bucket.name)
    except:
        logger.exception(
            "Couldn't get lifecycle rules for bucket '%s'.", self.bucket.name)
        raise
    else:
        return config.rules
```

- For API details, see [GetBucketLifecycleConfiguration](#) in *AWS SDK for Python (Boto3) API Reference*.

For a complete list of AWS SDK developer guides and code examples, see [Using this service with an AWS SDK \(p. 1185\)](#). This topic also includes information about getting started and details about previous SDK versions.

Get the policy for an Amazon S3 bucket using an AWS SDK

The following code examples show how to get the policy for an S3 bucket.

C++

SDK for C++

Tip

To learn how to set up and run this example, see [GitHub](#).

```
bool AwsDoc::S3::GetBucketPolicy(const Aws::String& bucketName,
                                  const Aws::String& region)
{
    Aws::Client::ClientConfiguration config;
    config.region = region;

    Aws::S3::S3Client s3_client(config);

    Aws::S3::Model::GetBucketPolicyRequest request;
    request.SetBucket(bucketName);

    Aws::S3::Model::GetBucketPolicyOutcome outcome =
        s3_client.GetBucketPolicy(request);

    if (outcome.IsSuccess())
    {
        Aws::StringStream policy_stream;
        Aws::String line;
```

```
outcome.GetResult().GetPolicy() >> line;
policy_stream << line;

std::cout << "Policy:" << std::endl << std::endl <<
    policy_stream.str() << std::endl;

    return true;
}
else
{
    auto err = outcome.GetError();
    std::cout << "Error: GetBucketPolicy: "
        << err.GetExceptionName() << ":" << err.GetMessage() << std::endl;

    return false;
}

int main()
{
    //TODO: Change bucket_name to the name of a bucket in your account.
    const Aws::String bucket_name = "<Enter bucket name>";
    //TODO: Set to the AWS Region in which the bucket was created.
    const Aws::String region = "us-east-1";

    Aws::SDKOptions options;
    Aws::InitAPI(options);
    {
        if (!AwsDoc::S3::GetBucketPolicy(bucket_name, region))
        {
            return 1;
        }
    }
    Aws::ShutdownAPI(options);

    return 0;
}
```

- For API details, see [GetBucketPolicy](#) in *AWS SDK for C++ API Reference*.

Java

SDK for Java 2.x

Tip

To learn how to set up and run this example, see [GitHub](#).

```
public static String getPolicy(S3Client s3, String bucketName) {

    String policyText;
    System.out.format("Getting policy for bucket: \"%s\"\n\n", bucketName);
    GetBucketPolicyRequest policyReq = GetBucketPolicyRequest.builder()
        .bucket(bucketName)
        .build();

    try {
        GetBucketPolicyResponse policyRes = s3.getBucketPolicy(policyReq);
        policyText = policyRes.policy();
        return policyText;
    } catch (S3Exception e) {
```

```
        System.err.println(e.awsErrorDetails().errorMessage());
        System.exit(1);
    }

    return "";
}
```

- For API details, see [GetBucketPolicy](#) in *AWS SDK for Java 2.x API Reference*.

JavaScript

SDK for JavaScript V3

Tip

To learn how to set up and run this example, see [GitHub](#).

Create the client.

```
// Create service client module using ES6 syntax.
import { S3Client } from "@aws-sdk/client-s3";
// Set the AWS Region.
const REGION = "us-east-1";
// Create an Amazon S3 service client object.
const s3Client = new S3Client({ region: REGION });
export { s3Client };
```

Get the bucket policy.

```
// Import required AWS SDK clients and commands for Node.js.
import { GetBucketPolicyCommand } from "@aws-sdk/client-s3";
import { s3Client } from "./libs/s3Client.js"; // Helper function that creates an
Amazon S3 service client module.

// Create the parameters for calling
export const bucketParams = { Bucket: "BUCKET_NAME" };

export const run = async () => {
  try {
    const data = await s3Client.send(new GetBucketPolicyCommand(bucketParams));
    console.log("Success", data);
    return data; // For unit tests.
  } catch (err) {
    console.log("Error", err);
  }
};
run();
```

- For more information, see [AWS SDK for JavaScript Developer Guide](#).
- For API details, see [GetBucketPolicy](#) in *AWS SDK for JavaScript API Reference*.

Kotlin

SDK for Kotlin

Note

This is prerelease documentation for a feature in preview release. It is subject to change.

Tip

To learn how to set up and run this example, see [GitHub](#).

```
suspend fun getPolicy(bucketName: String): String? {  
  
    println("Getting policy for bucket $bucketName")  
  
    val request = GetBucketPolicyRequest {  
        bucket = bucketName  
    }  
  
    S3Client { region = "us-east-1" }.use { s3 ->  
        val policyRes = s3.getBucketPolicy(request)  
        return policyRes.policy  
    }  
}
```

- For API details, see [GetBucketPolicy](#) in *AWS SDK for Kotlin API reference*.

Python

SDK for Python (Boto3)

Tip

To learn how to set up and run this example, see [GitHub](#).

```
class BucketWrapper:  
    def __init__(self, bucket):  
        self.bucket = bucket  
        self.name = bucket.name  
  
    def get_policy(self):  
        """  
        Get the security policy of the bucket.  
  
        :return: The security policy of the specified bucket, in JSON format.  
        """  
        try:  
            policy = self.bucket.Policy()  
            logger.info("Got policy %s for bucket '%s'.", policy.policy,  
self.bucket.name)  
        except ClientError:  
            logger.exception("Couldn't get policy for bucket '%s'.",  
self.bucket.name)  
            raise  
        else:  
            return json.loads(policy.policy)
```

- For API details, see [GetBucketPolicy](#) in *AWS SDK for Python (Boto3) API Reference*.

Ruby

SDK for Ruby

Tip

To learn how to set up and run this example, see [GitHub](#).

```
# Wraps an Amazon S3 bucket policy.
class BucketPolicyWrapper
  attr_reader :bucket_policy

  # @param bucket_policy [Aws::S3::BucketPolicy] A bucket policy object configured
  # with an existing bucket.
  def initialize(bucket_policy)
    @bucket_policy = bucket_policy
  end

  # Gets the policy of a bucket.
  #
  # @return [Aws::S3::GetBucketPolicyOutput, nil] The current bucket policy.
  def get_policy
    policy = @bucket_policy.data.policy
    policy.respond_to?(:read) ? policy.read : policy
    rescue Aws::Errors::ServiceError => e
      puts "Couldn't get the policy for #{@bucket_policy.bucket.name}. Here's why:
#{e.message}"
      nil
    end
  end
end
```

- For API details, see [GetBucketPolicy](#) in *AWS SDK for Ruby API Reference*.

For a complete list of AWS SDK developer guides and code examples, see [Using this service with an AWS SDK \(p. 1185\)](#). This topic also includes information about getting started and details about previous SDK versions.

Get the website configuration for an Amazon S3 bucket using an AWS SDK

The following code examples show how to get the website configuration for an S3 bucket.

C++

SDK for C++

Tip

To learn how to set up and run this example, see [GitHub](#).

```
bool AwsDoc::S3::GetWebsiteConfig(const Aws::String& bucketName,
                                    const Aws::String& region)
{
    Aws::Client::ClientConfiguration config;

    if (!region.empty())
    {
        config.region = region;
    }

    Aws::S3::S3Client s3_client(config);

    Aws::S3::Model::GetBucketWebsiteRequest request;
    request.SetBucket(bucketName);

    Aws::S3::Model::GetBucketWebsiteOutcome outcome =
```

```
s3_client.GetBucketWebsite(request);

if (outcome.IsSuccess())
{
    Aws::S3::Model::GetBucketWebsiteResult result = outcome.GetResult();

    std::cout << "Success: GetBucketWebsite: "
        << std::endl << std::endl
        << "For bucket '" << bucketName << "'";
    << std::endl
    << "Index page : "
    << result.GetIndexDocument().GetSuffix()
    << std::endl
    << "Error page: "
    << result.GetErrorDocument().GetKey()
    << std::endl;

    return true;
}
else
{
    auto err = outcome.GetError();

    std::cout << "Error: GetBucketWebsite: "
        << err.GetMessage() << std::endl;

    return false;
}

int main()
{
    Aws::SDKOptions options;
    Aws::InitAPI(options);
    {
        //TODO: Change bucket_name to the name of a bucket in your account.
        const Aws::String bucket_name = "DOC-EXAMPLE-BUCKET";
        //TODO: Set to the AWS Region in which the bucket was created.
        const Aws::String region = "us-east-1";

        if (!AwsDoc::S3::GetWebsiteConfig(bucket_name, region))
        {
            return 1;
        }
    }
    Aws::ShutdownAPI(options);

    return 0;
}
```

- For API details, see [GetBucketWebsite](#) in *AWS SDK for C++ API Reference*.

JavaScript

SDK for JavaScript V3

Tip

To learn how to set up and run this example, see [GitHub](#).

Create the client.

```
// Create service client module using ES6 syntax.
import { S3Client } from "@aws-sdk/client-s3";
```

```
// Set the AWS Region.  
const REGION = "us-east-1";  
// Create an Amazon S3 service client object.  
const s3Client = new S3Client({ region: REGION });  
export { s3Client };
```

Get the website configuration.

```
// Import required AWS SDK clients and commands for Node.js.  
import { GetBucketWebsiteCommand } from "@aws-sdk/client-s3";  
import { s3Client } from "./libs/s3Client.js"; // Helper function that creates an  
Amazon S3 service client module.  
  
// Create the parameters for calling  
export const bucketParams = { Bucket: "BUCKET_NAME" };  
  
export const run = async () => {  
    try {  
        const data = await s3Client.send(new GetBucketWebsiteCommand(bucketParams));  
        console.log("Success", data);  
        return data; // For unit tests.  
    } catch (err) {  
        console.log("Error", err);  
    }  
};  
run();
```

- For API details, see [GetBucketWebsite](#) in *AWS SDK for JavaScript API Reference*.

For a complete list of AWS SDK developer guides and code examples, see [Using this service with an AWS SDK \(p. 1185\)](#). This topic also includes information about getting started and details about previous SDK versions.

List Amazon S3 buckets using an AWS SDK

The following code examples show how to list S3 buckets.

C++

SDK for C++

Tip

To learn how to set up and run this example, see [GitHub](#).

```
bool AwsDoc::S3::ListBuckets() {  
    Aws::S3::S3Client client;  
  
    auto outcome = client.ListBuckets();  
    if (outcome.IsSuccess()) {  
        std::cout << "Found " << outcome.GetResult().GetBuckets().size() << "  
buckets\n";  
        for (auto&& b : outcome.GetResult().GetBuckets()) {  
            std::cout << b.GetName() << std::endl;  
        }  
    }  
    return true;
```

```
        }
    else {
        std::cout << "Failed with error: " << outcome.GetError() << std::endl;
        return false;
    }
}

int main()
{
    //The Aws::SDKOptions struct contains SDK configuration options.
    //An instance of Aws::SDKOptions is passed to the Aws::InitAPI and
    //Aws::ShutdownAPI methods. The same instance should be sent to both methods.
    Aws::SDKOptions options;
    options.loggingOptions.logLevel = Aws::Utils::Logging::LogLevel::Debug;

    //The AWS SDK for C++ must be initialized by calling Aws::InitAPI.
    InitAPI(options);

    AwsDoc::S3::ListBuckets();

    //Before the application terminates, the SDK must be shut down.
    ShutdownAPI(options);
    return 0;
}
```

- For API details, see [ListBuckets in AWS SDK for C++ API Reference](#).

Go

SDK for Go V2

Tip

To learn how to set up and run this example, see [GitHub](#).

```
listBucketsResult, err := client.ListBuckets(context.TODO(),
&s3.ListBucketsInput{})

if err != nil {
    panic("Couldn't list buckets")
}

for _, bucket := range listBucketsResult.Buckets {
    fmt.Printf("Bucket name: %s\t\tcreated at: %v\n", *bucket.Name,
bucket.CreationDate)
}
```

- For API details, see [ListBuckets in AWS SDK for Go API Reference](#).

JavaScript

SDK for JavaScript V3

Tip

To learn how to set up and run this example, see [GitHub](#).

Create the client.

```
// Create service client module using ES6 syntax.
```

```
import { S3Client } from "@aws-sdk/client-s3";
// Set the AWS Region.
const REGION = "us-east-1";
// Create an Amazon S3 service client object.
const s3Client = new S3Client({ region: REGION });
export { s3Client };
```

List the buckets.

```
// Import required AWS SDK clients and commands for Node.js.
import { ListBucketsCommand } from "@aws-sdk/client-s3";
import { s3Client } from "./libs/s3Client.js"; // Helper function that creates an
Amazon S3 service client module.

export const run = async () => {
  try {
    const data = await s3Client.send(new ListBucketsCommand({}));
    console.log("Success", data.Buckets);
    return data; // For unit tests.
  } catch (err) {
    console.log("Error", err);
  }
}
run();
```

- For more information, see [AWS SDK for JavaScript Developer Guide](#).
- For API details, see [ListBuckets](#) in [AWS SDK for JavaScript API Reference](#).

Python

SDK for Python (Boto3)

Tip

To learn how to set up and run this example, see [GitHub](#).

```
class BucketWrapper:
    def __init__(self, bucket):
        self.bucket = bucket
        self.name = bucket.name

    @staticmethod
    def list(s3_resource):
        """
        Get the buckets in all Regions for the current account.

        :return: The list of buckets.
        """
        try:
            buckets = list(s3_resource.buckets.all())
            logger.info("Got buckets: %s.", buckets)
        except ClientError:
            logger.exception("Couldn't get buckets.")
            raise
        else:
            return buckets
```

- For API details, see [ListBuckets](#) in [AWS SDK for Python \(Boto3\) API Reference](#).

Ruby

SDK for Ruby

Tip

To learn how to set up and run this example, see [GitHub](#).

```
require "aws-sdk-s3"

# Wraps Amazon S3 resource actions.
class BucketListWrapper
  attr_reader :s3_resource

  # @param s3_resource [Aws::S3::Resource] An Amazon S3 resource.
  def initialize(s3_resource)
    @s3_resource = s3_resource
  end

  # Lists buckets for the current account.
  #
  # @param count [Integer] The maximum number of buckets to list.
  def list_buckets(count)
    puts "Found these buckets:"
    @s3_resource.buckets.each do |bucket|
      puts "\t#{bucket.name}"
      count -= 1
      break if count.zero?
    end
    true
  rescue Aws::Errors::ServiceError => e
    puts "Couldn't list buckets. Here's why: #{e.message}"
    false
  end
end

def run_demo
  wrapper = BucketListWrapper.new(Aws::S3::Resource.new)
  wrapper.list_buckets(25)
end

run_demo if $PROGRAM_NAME == __FILE__
```

- For API details, see [ListBuckets in AWS SDK for Ruby API Reference](#).

Rust

SDK for Rust

Note

This documentation is for an SDK in preview release. The SDK is subject to change and should not be used in production.

Tip

To learn how to set up and run this example, see [GitHub](#).

```
async fn show_buckets(strict: bool, client: &Client, region: &str) -> Result<(), Error> {
    let resp = client.list_buckets().send().await?;
    let buckets = resp.buckets().unwrap_or_default();
```

```
let num_buckets = buckets.len();

let mut in_region = 0;

for bucket in buckets {
    if strict {
        let r = client
            .get_bucket_location()
            .bucket(bucket.name()).unwrap_or_default()
            .send()
            .await?;

        if r.location_constraint().unwrap().as_ref() == region {
            println!("{}", bucket.name().unwrap_or_default());
            in_region += 1;
        }
    } else {
        println!("{}", bucket.name().unwrap_or_default());
    }
}

println!();
if strict {
    println!(
        "Found {} buckets in the {} region out of a total of {} buckets.",
        in_region, region, num_buckets
    );
} else {
    println!("Found {} buckets in all regions.", num_buckets);
}

Ok(())
}
```

- For API details, see [ListBuckets in AWS SDK for Rust API reference](#).

For a complete list of AWS SDK developer guides and code examples, see [Using this service with an AWS SDK \(p. 1185\)](#). This topic also includes information about getting started and details about previous SDK versions.

List in-progress multipart uploads to an Amazon S3 bucket using an AWS SDK

The following code example shows how to list in-progress multipart uploads to an S3 bucket.

Java

SDK for Java 2.x

Tip

To learn how to set up and run this example, see [GitHub](#).

```
public static void listUploads( S3Client s3, String bucketName) {

    try {
        ListMultipartUploadsRequest listMultipartUploadsRequest =
ListMultipartUploadsRequest.builder()
            .bucket(bucketName)
```

```
.build();

List<MultipartUpload> uploads = response/uploads();
for (MultipartUpload upload: uploads) {
    System.out.println("Upload in progress: Key = " + upload.key() +
"\n", id = " + upload.uploadId());
}

} catch (S3Exception e) {
    System.err.println(e.getMessage());
    System.exit(1);
}
}
```

- For API details, see [ListMultipartUploads](#) in *AWS SDK for Java 2.x API Reference*.

For a complete list of AWS SDK developer guides and code examples, see [Using this service with an AWS SDK \(p. 1185\)](#). This topic also includes information about getting started and details about previous SDK versions.

List the version of objects in an Amazon S3 bucket using an AWS SDK

The following code example shows how to list object versions in an S3 bucket.

Rust

SDK for Rust

Note

This documentation is for an SDK in preview release. The SDK is subject to change and should not be used in production.

Tip

To learn how to set up and run this example, see [GitHub](#).

```
async fn show_versions(client: &Client, bucket: &str) -> Result<(), Error> {
    let resp = client.list_object_versions().bucket(bucket).send().await?;

    for version in resp.versions().unwrap_or_default() {
        println!("{}", version.key().unwrap_or_default());
        println!(" version ID: {}", version.version_id().unwrap_or_default());
        println!();
    }
    Ok(())
}
```

- For API details, see [ListObjectVersions](#) in *AWS SDK for Rust API reference*.

For a complete list of AWS SDK developer guides and code examples, see [Using this service with an AWS SDK \(p. 1185\)](#). This topic also includes information about getting started and details about previous SDK versions.

List objects in an Amazon S3 bucket using an AWS SDK

The following code examples show how to list objects in an S3 bucket.

.NET

AWS SDK for .NET

Tip

To learn how to set up and run this example, see [GitHub](#).

```
/// <summary>
/// Shows how to list the objects in an Amazon S3 bucket.
/// </summary>
/// <param name="client">An initialized Amazon S3 client object.</param>
/// <param name="bucketName">The name of the bucket for which to list
/// the contents.</param>
/// <returns>A boolean value indicating the success or failure of the
/// copy operation.</returns>
public static async Task<bool> ListBucketContentsAsync(IAmazonS3 client,
string bucketName)
{
    try
    {
        var request = new ListObjectsV2Request
        {
            BucketName = bucketName,
            MaxKeys = 5,
        };

        Console.WriteLine("-----");
        Console.WriteLine($"Listing the contents of {bucketName}:");
        Console.WriteLine("-----");

        var response = new ListObjectsV2Response();

        do
        {
            response = await client.ListObjectsV2Async(request);

            response.S3Objects
                .ForEach(obj => Console.WriteLine($"{obj.Key,-35}
{obj.LastModified.ToShortDateString(),10}{obj.Size,10}"));

            // If the response is truncated, set the request
            ContinuationToken
                // from the NextContinuationToken property of the response.
                request.ContinuationToken = response.NextContinuationToken;
        }
        while (response.IsTruncated);

        return true;
    }
    catch (AmazonS3Exception ex)
    {
        Console.WriteLine($"Error encountered on server.
Message:{ex.Message}' getting list of objects.");
        return false;
    }
}
```

- For API details, see [ListObjects in AWS SDK for .NET API Reference](#).

C++

SDK for C++

Tip

To learn how to set up and run this example, see [GitHub](#).

```
bool AwsDoc::S3::ListObjects(const Aws::String& bucketName,
    const Aws::String& region)
{
    Aws::Client::ClientConfiguration config;

    if (!region.empty())
    {
        config.region = region;
    }

    Aws::S3::S3Client s3_client(config);

    Aws::S3::Model::ListObjectsRequest request;
    request.WithBucket(bucketName);

    auto outcome = s3_client.ListObjects(request);

    if (outcome.IsSuccess())
    {
        std::cout << "Objects in bucket '" << bucketName << "'":
        << std::endl << std::endl;

        Aws::Vector<Aws::S3::Model::Object> objects =
            outcome.GetResult().GetContents();

        for (Aws::S3::Model::Object& object : objects)
        {
            std::cout << object.GetKey() << std::endl;
        }

        return true;
    }
    else
    {
        std::cout << "Error: ListObjects: " <<
            outcome.GetError().GetMessage() << std::endl;

        return false;
    }
}

int main()
{
    Aws::SDKOptions options;
    Aws::InitAPI(options);
    {
        //TODO: Name of a bucket in your account.
        //The bucket must have at least one object in it. One way to achieve
        //this is to configure and run put_object.cpp's executable first.
        const Aws::String bucket_name = "DOC-EXAMPLE-BUCKET";
```

```
//TODO: Set to the AWS Region in which the bucket was created.  
Aws::String region = "us-east-1";  
  
if (!AwsDoc::S3::ListObjects(bucket_name, region))  
{  
    return 1;  
}  
  
}  
Aws::ShutdownAPI(options);  
  
return 0;  
}
```

- For API details, see [ListObjects in AWS SDK for C++ API Reference](#).

Go

SDK for Go V2

Tip

To learn how to set up and run this example, see [GitHub](#).

```
// List objects in the bucket.  
// n.b. object keys in Amazon S3 do not begin with '/'. You do not need to lead  
your  
// prefix with it.  
fmt.Println("Listing the objects in the bucket:")  
listObjsResponse, err := client.ListObjectsV2(context.TODO(),  
&s3.ListObjectsV2Input{  
    Bucket: aws.String(name),  
    Prefix: aws.String(""),  
})  
  
if err != nil {  
    panic("Couldn't list bucket contents")  
}  
  
for _, object := range listObjsResponse.Contents {  
    fmt.Printf("%s (%d bytes, class %v) \n", *object.Key, object.Size,  
    object.StorageClass)  
}
```

- For API details, see [ListObjects in AWS SDK for Go API Reference](#).

Java

SDK for Java 2.x

Tip

To learn how to set up and run this example, see [GitHub](#).

```
public static void listBucketObjects(S3Client s3, String bucketName ) {  
  
    try {  
        ListObjectsRequest listObjects = ListObjectsRequest  
            .builder()  
            .bucket(bucketName)
```

```
.build();

ListObjectsResponse res = s3.listObjects(listObjects);
List<S3Object> objects = res.contents();
for (S3Object myValue : objects) {
    System.out.print("\n The name of the key is " + myValue.key());
    System.out.print("\n The object is " + calKb(myValue.size()) + " KBs");
    System.out.print("\n The owner is " + myValue.owner());
}

} catch (S3Exception e) {
    System.err.println(e.awsErrorDetails().errorMessage());
    System.exit(1);
}
}

//convert bytes to kbs.
private static long calKb(Long val) {
    return val/1024;
}
```

- For API details, see [ListObjects in AWS SDK for Java 2.x API Reference](#).

JavaScript

SDK for JavaScript V3

Tip

To learn how to set up and run this example, see [GitHub](#).

Create the client.

```
// Create service client module using ES6 syntax.
import { S3Client } from "@aws-sdk/client-s3";
// Set the AWS Region.
const REGION = "us-east-1";
// Create an Amazon S3 service client object.
const s3Client = new S3Client({ region: REGION });
export { s3Client };
```

List the objects.

```
// Import required AWS SDK clients and commands for Node.js.
import { ListObjectsCommand } from "@aws-sdk/client-s3";
import { s3Client } from "./libs/s3Client.js"; // Helper function that creates an
Amazon S3 service client module.

// Create the parameters for the bucket
export const bucketParams = { Bucket: "BUCKET_NAME" };

export const run = async () => {
    try {
        const data = await s3Client.send(new ListObjectsCommand(bucketParams));
        console.log("Success", data);
        return data; // For unit tests.
    } catch (err) {
        console.log("Error", err);
    }
};
run();
```

List 1000 or more objects.

```
// Import required AWS SDK clients and commands for Node.js.
import { ListObjectsCommand } from "@aws-sdk/client-s3";
import { s3Client } from "./libs/s3Client.js"; // Helper function that creates an
// Amazon S3 service client module.

// Create the parameters for the bucket
export const bucketParams = { Bucket: "BUCKET_NAME" };

export async function run() {
    // Declare truncated as a flag that the while loop is based on.
    let truncated = true;
    // Declare a variable to which the key of the last element is assigned to in the
    response.
    let pageMarker;
    // while loop that runs until 'response.truncated' is false.
    while (truncated) {
        try {
            const response = await s3Client.send(new ListObjectsCommand(bucketParams));
            // return response; //For unit tests
            response.Contents.forEach((item) => {
                console.log(item.Key);
            });
            // Log the key of every item in the response to standard output.
            truncated = response.IsTruncated;
            // If truncated is true, assign the key of the last element in the response
            to the pageMarker variable.
            if (truncated) {
                pageMarker = response.Contents.slice(-1)[0].Key;
                // Assign the pageMarker value to bucketParams so that the next iteration
                starts from the new pageMarker.
                bucketParams.Marker = pageMarker;
            }
            // At end of the list, response.truncated is false, and the function exits
            the while loop.
        } catch (err) {
            console.log("Error", err);
            truncated = false;
        }
    }
}
run();
```

- For API details, see [ListObjects in AWS SDK for JavaScript API Reference](#).

Kotlin

SDK for Kotlin

Note

This is prerelease documentation for a feature in preview release. It is subject to change.

Tip

To learn how to set up and run this example, see [GitHub](#).

```
suspend fun listBucketObjects(bucketName: String) {
```

```
val request = ListObjectsRequest {
    bucket = bucketName
}

S3Client { region = "us-east-1" }.use { s3 ->

    val response = s3.listObjects(request)
    response.contents?.forEach { myObject ->
        println("The name of the key is ${myObject.key}")
        println("The object is ${calKb(myObject.size)} KBs")
        println("The owner is ${myObject.owner}")
    }
}

private fun calKb(intValue: Long): Long {
    return intValue / 1024
}
```

- For API details, see [ListObjects in AWS SDK for Kotlin API reference](#).

PHP

SDK for PHP

Tip

To learn how to set up and run this example, see [GitHub](#).

List objects in a bucket.

```
$s3client = new Aws\S3\S3Client(['region' => 'us-west-2', 'version' => 'latest']);

try {
    $contents = $s3client->listObjects([
        'Bucket' => $bucket_name,
    ]);
    echo "The contents of your bucket are: \n";
    foreach ($contents['Contents'] as $content) {
        echo $content['Key'] . "\n";
    }
} catch (Exception $exception) {
    echo "Failed to list objects in $bucket_name with error: " . $exception-
>getMessage();
    exit("Please fix error with listing objects before continuing.");
}
```

- For API details, see [ListObjects in AWS SDK for PHP API Reference](#).

Python

SDK for Python (Boto3)

Tip

To learn how to set up and run this example, see [GitHub](#).

```
class ObjectWrapper:
    def __init__(self, s3_object):
        self.object = s3_object
        self.key = self.object.key
```

```
@staticmethod
def list(bucket, prefix=None):
    """
    Lists the objects in a bucket, optionally filtered by a prefix.

    :param bucket: The bucket to query.
    :param prefix: When specified, only objects that start with this prefix are
    listed.
    :return: The list of objects.
    """
    try:
        if not prefix:
            objects = list(bucket.objects.all())
        else:
            objects = list(bucket.objects.filter(Prefix=prefix))
        logger.info("Got objects %s from bucket '%s'",
                    [o.key for o in objects], bucket.name)
    except ClientError:
        logger.exception("Couldn't get objects for bucket '%s'.", bucket.name)
        raise
    else:
        return objects
```

- For API details, see [ListObjects in AWS SDK for Python \(Boto3\) API Reference](#).

Ruby

SDK for Ruby

Tip

To learn how to set up and run this example, see [GitHub](#).

```
require "aws-sdk-s3"

# Wraps Amazon S3 bucket actions.
class BucketListObjectsWrapper
  attr_reader :bucket

  # @param bucket [Aws::S3::Bucket] An existing Amazon S3 bucket.
  def initialize(bucket)
    @bucket = bucket
  end

  # Lists object in a bucket.
  #
  # @param max_objects [Integer] The maximum number of objects to list.
  # @return [Integer] The number of objects listed.
  def list_objects(max_objects)
    count = 0
    puts "The objects in #{@bucket.name} are:"
    @bucket.objects.each do |obj|
      puts "\t#{obj.key}"
      count += 1
      break if count == max_objects
    end
    count
  rescue Aws::Errors::ServiceError => e
    puts "Couldn't list objects in bucket #{@bucket.name}. Here's why: #{e.message}"
    0
  end
end
```

```
def run_demo
  bucket_name = "doc-example-bucket"

  wrapper = BucketListObjectsWrapper.new(Aws::S3::Bucket.new(bucket_name))
  count = wrapper.list_objects(25)
  puts "Listed #{count} objects."
end

run_demo if $PROGRAM_NAME == __FILE__
```

- For API details, see [ListObjects in AWS SDK for Ruby API Reference](#).

Rust

SDK for Rust

Note

This documentation is for an SDK in preview release. The SDK is subject to change and should not be used in production.

Tip

To learn how to set up and run this example, see [GitHub](#).

```
pub async fn list_objects(client: &Client, bucket_name: &str) -> Result<(), Error>
{
    let objects = client.list_objects_v2().bucket(bucket_name).send().await?;
    println!("Objects in bucket:");
    for obj in objects.contents().unwrap_or_default() {
        println!("{}:{}", obj.key().unwrap());
    }
    Ok(())
}
```

- For API details, see [ListObjects in AWS SDK for Rust API reference](#).

Swift

SDK for Swift

Note

This is prerelease documentation for an SDK in preview release. It is subject to change.

Tip

To learn how to set up and run this example, see [GitHub](#).

```
public func listBucketFiles(bucket: String) async throws -> [String] {
    let input = ListObjectsV2Input(
        bucket: bucket
    )
    let output = try await client.listObjectsV2(input: input)
    var names: [String] = []

    guard let objList = output.contents else {
        return []
    }

    for obj in objList {
```

```
        if let objName = obj.key {
            names.append(objName)
        }
    }

    return names
}
```

- For API details, see [ListObjects in AWS SDK for Swift API reference](#).

For a complete list of AWS SDK developer guides and code examples, see [Using this service with an AWS SDK \(p. 1185\)](#). This topic also includes information about getting started and details about previous SDK versions.

Restore an archived copy of an object back into an Amazon S3 bucket using an AWS SDK

The following code example shows how to restore an archived copy of an object back into an S3 bucket.

Java

SDK for Java 2.x

Tip

To learn how to set up and run this example, see [GitHub](#).

```
public static void restoreS3Object(S3Client s3, String bucketName, String
keyName, String expectedBucketOwner) {

    try {
        RestoreRequest restoreRequest = RestoreRequest.builder()
            .days(10)

        .glacierJobParameters(GlacierJobParameters.builder().tier(Tier.STANDARD).build())
            .build();

        RestoreObjectRequest objectRequest = RestoreObjectRequest.builder()
            .expectedBucketOwner(expectedBucketOwner)
            .bucket(bucketName)
            .key(keyName)
            .restoreRequest(restoreRequest)
            .build();

        s3.restoreObject(objectRequest);

    } catch (S3Exception e) {
        System.err.println(e.awsErrorDetails().errorMessage());
        System.exit(1);
    }
}
```

- For API details, see [RestoreObject in AWS SDK for Java 2.x API Reference](#).

For a complete list of AWS SDK developer guides and code examples, see [Using this service with an AWS SDK \(p. 1185\)](#). This topic also includes information about getting started and details about previous SDK versions.

Set a new ACL for an Amazon S3 bucket using an AWS SDK

The following code examples show how to set a new access control list (ACL) for an S3 bucket.

Java

SDK for Java 2.x

Tip

To learn how to set up and run this example, see [GitHub](#).

```
public static void setBucketAcl(S3Client s3, String bucketName, String id) {

    try {
        Grant ownerGrant = Grant.builder()
            .grantee(builder -> builder.id(id))
            .type(Type.CANONICAL_USER)
            .permission(Permission.FULL_CONTROL)
            .build();

        List<Grant> grantList2 = new ArrayList<>();
        grantList2.add(ownerGrant);

        AccessControlPolicy acl = AccessControlPolicy.builder()
            .owner(builder -> builder.id(id))
            .grants(grantList2)
            .build();

        PutBucketAclRequest putAclReq = PutBucketAclRequest.builder()
            .bucket(bucketName)
            .accessControlPolicy(acl)
            .build();

        s3.putBucketAcl(putAclReq);

    } catch (S3Exception e) {
        e.printStackTrace();
        System.exit(1);
    }
}
```

- For API details, see [PutBucketAcl](#) in *AWS SDK for Java 2.x API Reference*.

JavaScript

SDK for JavaScript V3

Tip

To learn how to set up and run this example, see [GitHub](#).

Create the client.

```
// Create service client module using ES6 syntax.
import { S3Client } from "@aws-sdk/client-s3";
// Set the AWS Region.
const REGION = "us-east-1";
// Create an Amazon S3 service client object.
const s3Client = new S3Client({ region: REGION });
```

```
export { s3Client };
```

Put the bucket ACL.

```
// Import required AWS SDK clients and commands for Node.js.
import { PutBucketAclCommand } from "@aws-sdk/client-s3";
import { s3Client } from "./libs/s3Client.js"; // Helper function that creates an
Amazon S3 service client module.

// Set the parameters. For more information,
// see https://docs.aws.amazon.com/AWSJavaScriptSDK/latest/AWS/
S3.html#putBucketAcl-property.
export const bucketParams = {
  Bucket: "BUCKET_NAME",
  // 'GrantFullControl' allows grantee the read, write, read ACP, and write ACL
  permissions on the bucket.
  // Use a canonical user ID for an AWS account, formatted as follows:
  // id=002160194XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXa7a49125274
  GrantFullControl: "GRANTEE_1",
  // 'GrantWrite' allows grantee to create, overwrite, and delete any object in the
  bucket.
  // For example, 'uri=http://acs.amazonaws.com/groups/s3/LogDelivery'
  GrantWrite: "GRANTEE_2",
};

export const run = async () => {
  try {
    const data = await s3Client.send(new PutBucketAclCommand(bucketParams));
    console.log("Success, permissions added to bucket", data);
    return data; // For unit tests.
  } catch (err) {
    console.log("Error", err);
  }
};
run();
```

- For more information, see [AWS SDK for JavaScript Developer Guide](#).
- For API details, see [PutBucketAcl](#) in [AWS SDK for JavaScript API Reference](#).

Kotlin

SDK for Kotlin

Note

This is prerelease documentation for a feature in preview release. It is subject to change.

Tip

To learn how to set up and run this example, see [GitHub](#).

```
suspend fun setBucketAcl(bucketName: String, idVal: String) {

    val myGrant = Grantee {
        id = idVal
        type = Type.CanonicalUser
    }

    val ownerGrant = Grant {
        grantee = myGrant
        permission = Permission.FullControl
    }
}
```

```
}

val grantList = mutableListOf<Grant>()
grantList.add(ownerGrant)

val ownerOb = Owner {
    id = idVal
}

val acl = AccessControlPolicy {
    owner = ownerOb
    grants = grantList
}

val request = PutBucketAclRequest {
    bucket = bucketName
    accessControlPolicy = acl
}

S3Client { region = "us-east-1" }.use { s3 ->
    s3.putBucketAcl(request)
    println("An ACL was successfully set on $bucketName")
}
}
```

- For API details, see [PutBucketAcl](#) in *AWS SDK for Kotlin API reference*.

Python

SDK for Python (Boto3)

Tip

To learn how to set up and run this example, see [GitHub](#).

```
class BucketWrapper:
    def __init__(self, bucket):
        self.bucket = bucket
        self.name = bucket.name

    def grant_log_delivery_access(self):
        """
        Grant the AWS Log Delivery group write access to the bucket so that
        Amazon S3 can deliver access logs to the bucket. This is the only
        recommended
        use of an S3 bucket ACL.
        """
        try:
            acl = self.bucket.Acl()
            # Putting an ACL overwrites the existing ACL. If you want to preserve
            # existing grants, append new grants to the list of existing grants.
            grants = acl.grants if acl.grants else []
            grants.append({
                'Grantee': {
                    'Type': 'Group',
                    'URI': 'http://acs.amazonaws.com/groups/s3/LogDelivery'
                },
                'Permission': 'WRITE'
            })
            acl.put(
                AccessControlPolicy={
                    'Grants': grants,
                    'Owner': acl.owner
                }
            )
        except Exception as e:
            print(f"Error granting log delivery access: {e}")

```

```
        }
    )
    logger.info("Granted log delivery access to bucket '%s',
self.bucket.name)
except ClientError:
    logger.exception("Couldn't add ACL to bucket '%s'.", self.bucket.name)
raise
```

- For API details, see [PutBucketAcl](#) in *AWS SDK for Python (Boto3) API Reference*.

For a complete list of AWS SDK developer guides and code examples, see [Using this service with an AWS SDK \(p. 1185\)](#). This topic also includes information about getting started and details about previous SDK versions.

Set the ACL of an Amazon S3 object using an AWS SDK

The following code example shows how to set the access control list (ACL) of an S3 object.

Python

SDK for Python (Boto3)

Tip

To learn how to set up and run this example, see [GitHub](#).

```
class ObjectWrapper:
    def __init__(self, s3_object):
        self.object = s3_object
        self.key = self.object.key

    def put_acl(self, email):
        """
        Applies an ACL to the object that grants read access to an AWS user
        identified
        by email address.

        :param email: The email address of the user to grant access.
        """
        try:
            acl = self.object.Acl()
            # Putting an ACL overwrites the existing ACL, so append new grants
            # if you want to preserve existing grants.
            grants = acl.grants if acl.grants else []
            grants.append({
                'Grantee': {
                    'Type': 'AmazonCustomerByEmail',
                    'EmailAddress': email
                },
                'Permission': 'READ'
            })
            acl.put(
                AccessControlPolicy={
                    'Grants': grants,
                    'Owner': acl.owner
                }
            )
            logger.info("Granted read access to %s.", email)
        except ClientError:
```

```
logger.exception("Couldn't add ACL to object '%s'.", self.object.key)
raise
```

- For API details, see [PutObjectAcl](#) in *AWS SDK for Python (Boto3) API Reference*.

For a complete list of AWS SDK developer guides and code examples, see [Using this service with an AWS SDK \(p. 1185\)](#). This topic also includes information about getting started and details about previous SDK versions.

Set the website configuration for an Amazon S3 bucket using an AWS SDK

The following code examples show how to set the website configuration for an S3 bucket.

C++

SDK for C++

Tip

To learn how to set up and run this example, see [GitHub](#).

```
bool AwsDoc::S3::PutWebsiteConfig(const Aws::String& bucketName,
                                    const Aws::String& indexPage, const Aws::String& errorPage,
                                    const Aws::String& region)
{
    Aws::Client::ClientConfiguration config;

    if (!region.empty())
    {
        config.region = region;
    }

    Aws::S3::S3Client s3_client(config);

    Aws::S3::Model::IndexDocument index_doc;
    index_doc.SetSuffix(indexPage);

    Aws::S3::Model::ErrorDocument error_doc;
    error_doc.SetKey(errorPage);

    Aws::S3::Model::WebsiteConfiguration website_config;
    website_config.SetIndexDocument(index_doc);
    website_config.SetErrorDocument(error_doc);

    Aws::S3::Model::PutBucketWebsiteRequest request;
    request.SetBucket(bucketName);
    request.SetWebsiteConfiguration(website_config);

    Aws::S3::Model::PutBucketWebsiteOutcome outcome =
        s3_client.PutBucketWebsite(request);

    if (outcome.IsSuccess())
    {
        std::cout << "Success: Set website configuration for bucket '" 
        << bucketName << "." << std::endl;

        return true;
    }
}
```

```
        else
    {
        std::cout << "Error: PutBucketWebsite: "
        << outcome.GetError().GetMessage() << std::endl;

        return false;
    }

    return 1;
}

int main()
{
    Aws::SDKOptions options;
    Aws::InitAPI(options);
    {
        //TODO: Change bucket_name to the name of a bucket in your account.
        const Aws::String bucket_name = "DOC-EXAMPLE-BUCKET";
        //TODO: Set to the AWS Region in which the bucket was created.
        const Aws::String region = "us-east-1";
        //TODO: Create these two files to serve as your website
        const Aws::String index_page = "index.html";
        const Aws::String error_page = "404.html";

        if (!AwsDoc::S3::PutWebsiteConfig(bucket_name, index_page, error_page,
region))
        {
            return 1;
        }

    }
    Aws::ShutdownAPI(options);

    return 0;
}
```

- For API details, see [PutBucketWebsite](#) in *AWS SDK for C++ API Reference*.

Java

SDK for Java 2.x

Tip

To learn how to set up and run this example, see [GitHub](#).

```
public static void setWebsiteConfig( S3Client s3, String bucketName, String
indexDoc) {

    try {
        WebsiteConfiguration websiteConfig = WebsiteConfiguration.builder()
            .indexDocument(IndexDocument.builder().suffix(indexDoc).build())
            .build();

        PutBucketWebsiteRequest pubWebsiteReq =
PutBucketWebsiteRequest.builder()
            .bucket(bucketName)
            .websiteConfiguration(websiteConfig)
            .build();

        s3.putBucketWebsite(pubWebsiteReq);
        System.out.println("The call was successful");
    }
}
```

```
        } catch (S3Exception e) {
            System.err.println(e.awsErrorDetails().errorMessage());
            System.exit(1);
        }
    }
```

- For API details, see [PutBucketWebsite](#) in *AWS SDK for Java 2.x API Reference*.

JavaScript

SDK for JavaScript V3

Tip

To learn how to set up and run this example, see [GitHub](#).

Create the client.

```
// Create service client module using ES6 syntax.
import { S3Client } from "@aws-sdk/client-s3";
// Set the AWS Region.
const REGION = "us-east-1";
// Create an Amazon S3 service client object.
const s3Client = new S3Client({ region: REGION });
export { s3Client };
```

Set the website configuration.

```
// Import required AWS SDK clients and commands for Node.js.
import { PutBucketWebsiteCommand } from "@aws-sdk/client-s3";
import { s3Client } from "./libs/s3Client.js"; // Helper function that creates an
// Amazon S3 service client module.

// Create the parameters for the bucket
export const bucketParams = { Bucket: "BUCKET_NAME" };
export const staticHostParams = {
    Bucket: bucketParams,
    WebsiteConfiguration: {
        ErrorDocument: {
            Key: "",
        },
        IndexDocument: {
            Suffix: "",
        },
    },
};

export const run = async () => {
    // Insert specified bucket name and index and error documents into parameters
    JSON
    // from command line arguments
    staticHostParams.Bucket = bucketParams;
    staticHostParams.WebsiteConfiguration.IndexDocument.Suffix = "INDEX_PAGE"; // The
    index document inserted into parameters JSON.
    staticHostParams.WebsiteConfiguration.ErrorDocument.Key = "ERROR_PAGE"; // The
    error document inserted into parameters JSON.
    // Set the new website configuration on the selected bucket.
    try {
        const data = await s3Client.send(new
PutBucketWebsiteCommand(staticHostParams));
        console.log("Success", data);
    }
}
```

```
        return data;
    } catch (err) {
        console.log("Error", err);
    }
};

run();
```

- For more information, see [AWS SDK for JavaScript Developer Guide](#).
- For API details, see [PutBucketWebsite](#) in [AWS SDK for JavaScript API Reference](#).

Ruby

SDK for Ruby

Tip

To learn how to set up and run this example, see [GitHub](#).

```
require "aws-sdk-s3"

# Wraps Amazon S3 bucket website actions.
class BucketWebsiteWrapper
    attr_reader :bucket_website

    # @param bucket_website [Aws::S3::BucketWebsite] A bucket website object
    # configured with an existing bucket.
    def initialize(bucket_website)
        @bucket_website = bucket_website
    end

    # Sets a bucket as a static website.
    #
    # @param index_document [String] The name of the index document for the website.
    # @param error_document [String] The name of the error document to show for 4XX
    # errors.
    # @return [Boolean] True when the bucket is configured as a website; otherwise,
    # false.
    def set_website(index_document, error_document)
        @bucket_website.put(
            website_configuration: {
                index_document: { suffix: index_document },
                error_document: { key: error_document }
            }
        )
        true
    rescue Aws::Errors::ServiceError => e
        puts "Couldn't configure #{@bucket_website.bucket.name} as a website. Here's
why: #{e.message}"
        false
    end
end

def run_demo
    bucket_name = "doc-example-bucket"
    index_document = "index.html"
    error_document = "404.html"

    wrapper = BucketWebsiteWrapper.new(Aws::S3::BucketWebsite.new(bucket_name))
    return unless wrapper.set_website(index_document, error_document)

    puts "Successfully configured bucket #{bucket_name} as a static website."
end
```

```
run_demo if $PROGRAM_NAME == __FILE__
```

- For API details, see [PutBucketWebsite](#) in *AWS SDK for Ruby API Reference*.

For a complete list of AWS SDK developer guides and code examples, see [Using this service with an AWS SDK \(p. 1185\)](#). This topic also includes information about getting started and details about previous SDK versions.

Upload an object to an Amazon S3 bucket using an AWS SDK

The following code examples show how to upload an object to an S3 bucket.

.NET

AWS SDK for .NET

Tip

To learn how to set up and run this example, see [GitHub](#).

```
/// <summary>
/// Shows how to upload a file from the local computer to an Amazon S3
/// bucket.
/// </summary>
/// <param name="client">An initialized Amazon S3 client object.</param>
/// <param name="bucketName">The Amazon S3 bucket to which the object
/// will be uploaded.</param>
/// <param name="objectName">The object to upload.</param>
/// <param name="filePath">The path, including file name, of the object
/// on the local computer to upload.</param>
/// <returns>A boolean value indicating the success or failure of the
/// upload procedure.</returns>
public static async Task<bool> UploadFileAsync(
    IAmazonS3 client,
    string bucketName,
    string objectName,
    string filePath)
{
    var request = new PutObjectRequest
    {
        BucketName = bucketName,
        Key = objectName,
        FilePath = filePath,
    };

    var response = await client.PutObjectAsync(request);
    if (response.HttpStatusCode == System.Net.HttpStatusCode.OK)
    {
        Console.WriteLine($"Successfully uploaded {objectName} to
{bucketName}.");
        return true;
    }
    else
    {
        Console.WriteLine($"Could not upload {objectName} to
{bucketName}.");
        return false;
    }
}
```

```
}
```

- For API details, see [PutObject](#) in *AWS SDK for .NET API Reference*.

C++

SDK for C++

Tip

To learn how to set up and run this example, see [GitHub](#).

```
bool AwsDoc::S3::PutObject(const Aws::String& bucketName,
    const Aws::String& objectName,
    const Aws::String& region)
{
    // Verify that the file exists.
    struct stat buffer;

    if (stat(objectName.c_str(), &buffer) == -1)
    {
        std::cout << "Error: PutObject: File '" <<
            objectName << "' does not exist." << std::endl;

        return false;
    }

    Aws::Client::ClientConfiguration config;

    if (!region.empty())
    {
        config.region = region;
    }

    Aws::S3::S3Client s3_client(config);

    Aws::S3::Model::PutObjectRequest request;
    request.SetBucket(bucketName);
    //We are using the name of the file as the key for the object in the bucket.
    //However, this is just a string and can set according to your retrieval needs.
    request.SetKey(objectName);

    std::shared_ptr<Aws::IOStream> input_data =
        Aws::MakeShared<Aws::FStream>("SampleAllocationTag",
            objectName.c_str(),
            std::ios_base::in | std::ios_base::binary);

    request.SetBody(input_data);

    Aws::S3::Model::PutObjectOutcome outcome =
        s3_client.PutObject(request);

    if (outcome.IsSuccess()) {

        std::cout << "Added object '" << objectName << "' to bucket '" 
            << bucketName << "'.";
        return true;
    }
    else
    {
        std::cout << "Error: PutObject: " <<
            outcome.GetError().GetMessage() << std::endl;
    }
}
```

```
        return false;
    }

}

int main()
{
    Aws::SDKOptions options;
    Aws::InitAPI(options);
    {
        //TODO: Change bucket_name to the name of a bucket in your account.
        const Aws::String bucket_name = "<Enter bucket name>";
        //TODO: Create a file called "my-file.txt" in the local folder where your
        executables are built to.
        const Aws::String object_name = "<Enter file>";
        //TODO: Set to the AWS Region in which the bucket was created.
        const Aws::String region = "us-east-1";

        if (!AwsDoc::S3::PutObject(bucket_name, object_name, region)) {

            return 1;
        }
    }
    Aws::ShutdownAPI(options);

    return 0;
}
```

- For API details, see [PutObject](#) in *AWS SDK for C++ API Reference*.

Go

SDK for Go V2

Tip

To learn how to set up and run this example, see [GitHub](#).

```
// Place an object in a bucket.
fmt.Println("Upload an object to the bucket")
// Get the object body to upload.
// Image credit: https://unsplash.com/photos/iz58d89q3ss
stat, err := os.Stat("image.jpg")
if err != nil {
    panic("Couldn't stat image: " + err.Error())
}
file, err := os.Open("image.jpg")

if err != nil {
    panic("Couldn't open local file")
}

_, err = client.PutObject(context.TODO(), &s3.PutObjectInput{
    Bucket:      aws.String(name),
    Key:         aws.String("path/myfile.jpg"),
    Body:         file,
    ContentLength: stat.Size(),
})
file.Close()

if err != nil {
    panic("Couldn't upload file: " + err.Error())
```

```
}
```

- For API details, see [PutObject](#) in [AWS SDK for Go API Reference](#).

Java

SDK for Java 2.x

Tip

To learn how to set up and run this example, see [GitHub](#).

Upload an object to a bucket.

```
public static String putS3Object(S3Client s3, String bucketName, String
objectKey, String objectPath) {

    try {
        Map<String, String> metadata = new HashMap<>();
        metadata.put("x-amz-meta-myVal", "test");
        PutObjectRequest putOb = PutObjectRequest.builder()
            .bucket(bucketName)
            .key(objectKey)
            .metadata(metadata)
            .build();

        PutObjectResponse response = s3.putObject(putOb,
RequestBody.fromBytes(getObjectFile(objectPath)));
        return response.eTag();

    } catch (S3Exception e) {
        System.err.println(e.getMessage());
        System.exit(1);
    }

    return "";
}

// Return a byte array.
private static byte[] getObjectFile(String filePath) {

    FileInputStream fileInputStream = null;
    byte[] bytesArray = null;

    try {
        File file = new File(filePath);
        bytesArray = new byte[(int) file.length()];
        fileInputStream = new FileInputStream(file);
        fileInputStream.read(bytesArray);

    } catch (IOException e) {
        e.printStackTrace();
    } finally {
        if (fileInputStream != null) {
            try {
                fileInputStream.close();
            } catch (IOException e) {
                e.printStackTrace();
            }
        }
    }
}

return bytesArray;
```

```
}
```

Upload an object to a bucket and set tags.

```
public static void putS3ObjectTags(S3Client s3, String bucketName, String
objectKey, String objectPath) {

    try {

        Tag tag1 = Tag.builder()
            .key("Tag 1")
            .value("This is tag 1")
            .build();

        Tag tag2 = Tag.builder()
            .key("Tag 2")
            .value("This is tag 2")
            .build();

        List<Tag> tags = new ArrayList<>();
        tags.add(tag1);
        tags.add(tag2);

        Tagging allTags = Tagging.builder()
            .tagSet(tags)
            .build();

        PutObjectRequest putOb = PutObjectRequest.builder()
            .bucket(bucketName)
            .key(objectKey)
            .tagging(allTags)
            .build();

        s3.putObject(putOb, RequestBody.fromBytes(getObjectFile(objectPath)));

    } catch (S3Exception e) {
        System.err.println(e.getMessage());
        System.exit(1);
    }
}

public static void updateObjectTags(S3Client s3, String bucketName, String
objectKey) {

    try {
        GetObjectTaggingRequest taggingRequest =
GetObjectTaggingRequest.builder()
            .bucket(bucketName)
            .key(objectKey)
            .build();

        GetObjectTaggingResponse getTaggingRes =
s3.getObjectTagging(taggingRequest);
        List<Tag> obTags = getTaggingRes.tagSet();
        for (Tag sinTag: obTags) {
            System.out.println("The tag key is: "+sinTag.key());
            System.out.println("The tag value is: "+sinTag.value());
        }

        // Replace the object's tags with two new tags.
        Tag tag3 = Tag.builder()
            .key("Tag 3")
            .value("This is tag 3")
            .build();
    }
}
```

```
Tag tag4 = Tag.builder()
    .key("Tag 4")
    .value("This is tag 4")
    .build();

List<Tag> tags = new ArrayList<>();
tags.add(tag3);
tags.add(tag4);

Tagging updatedTags = Tagging.builder()
    .tagSet(tags)
    .build();

PutObjectTaggingRequest taggingRequest1 =
PutObjectTaggingRequest.builder()
    .bucket(bucketName)
    .key(objectKey)
    .tagging(updatedTags)
    .build();

s3.putObjectTagging(taggingRequest1);
GetObjectTaggingResponse getTaggingRes2 =
s3.getObjectTagging(taggingRequest);
List<Tag> modTags = getTaggingRes2.tagSet();
for (Tag sinTag: modTags) {
    System.out.println("The tag key is: "+sinTag.key());
    System.out.println("The tag value is: "+sinTag.value());
}

} catch (S3Exception e) {
    System.err.println(e.getMessage());
    System.exit(1);
}
}
```

Upload an object to a bucket and set metadata.

```
public static String putS3Object(S3Client s3, String bucketName, String
objectKey, String objectPath) {

    try {
        Map<String, String> metadata = new HashMap<>();
        metadata.put("author", "Mary Doe");
        metadata.put("version", "1.0.0.0");

        PutObjectRequest putOb = PutObjectRequest.builder()
            .bucket(bucketName)
            .key(objectKey)
            .metadata(metadata)
            .build();

        PutObjectResponse response = s3.putObject(putOb,
RequestBody.fromBytes(getObjectFile(objectPath)));
        return response.eTag();

    } catch (S3Exception e) {
        System.err.println(e.getMessage());
        System.exit(1);
    }

    return "";
}
```

```
// Return a byte array.
private static byte[] getObjectFile(String filePath) {

    FileInputStream fileInputStream = null;
    byte[] byteArray = null;

    try {
        File file = new File(filePath);
        byteArray = new byte[(int) file.length()];
        fileInputStream = new FileInputStream(file);
        fileInputStream.read(byteArray);

    } catch (IOException e) {
        e.printStackTrace();
    } finally {
        if (fileInputStream != null) {
            try {
                fileInputStream.close();
            } catch (IOException e) {
                e.printStackTrace();
            }
        }
    }
    return byteArray;
}
```

Upload an object to a bucket and set an object retention value.

```
public static void setRetentionPeriod(S3Client s3, String key, String bucket) {

    try{
        LocalDate localDate = LocalDate.parse("2020-07-17");
        LocalDateTime localDateTime = localDate.atStartOfDay();
        Instant instant = localDateTime.toInstant(ZoneOffset.UTC);

        ObjectLockRetention lockRetention = ObjectLockRetention.builder()
            .mode("COMPLIANCE")
            .retainUntilDate(instant)
            .build();

        PutObjectRetentionRequest retentionRequest =
        PutObjectRetentionRequest.builder()
            .bucket(bucket)
            .key(key)
            .bypassGovernanceRetention(true)
            .retention(lockRetention)
            .build();

        // To set Retention on an object, the Amazon S3 bucket must support
        // object locking, otherwise an exception is thrown.
        s3.putObjectRetention(retentionRequest);
        System.out.print("An object retention configuration was successfully
placed on the object");

    } catch (S3Exception e) {
        System.err.println(e.awsErrorDetails().errorMessage());
        System.exit(1);
    }
}
```

- For API details, see [PutObject](#) in *AWS SDK for Java 2.x API Reference*.

JavaScript

SDK for JavaScript V3

Tip

To learn how to set up and run this example, see [GitHub](#).

Create the client.

```
// Create service client module using ES6 syntax.  
import { S3Client } from "@aws-sdk/client-s3";  
// Set the AWS Region.  
const REGION = "us-east-1";  
// Create an Amazon S3 service client object.  
const s3Client = new S3Client({ region: REGION });  
export { s3Client };
```

Create and upload the object.

```
// Import required AWS SDK clients and commands for Node.js.  
import { PutObjectCommand } from "@aws-sdk/client-s3";  
import { s3Client } from "./libs/s3Client.js"; // Helper function that creates an  
Amazon S3 service client module.  
  
// Set the parameters.  
export const bucketParams = {  
    Bucket: "BUCKET_NAME",  
    // Specify the name of the new object. For example, 'index.html'.  
    // To create a directory for the object, use '/'. For example, 'myApp/  
    package.json'.  
    Key: "OBJECT_NAME",  
    // Content of the new object.  
    Body: "BODY",  
};  
  
// Create and upload the object to the S3 bucket.  
export const run = async () => {  
    try {  
        const data = await s3Client.send(new PutObjectCommand(bucketParams));  
        return data; // For unit tests.  
        console.log(  
            "Successfully uploaded object: " +  
            bucketParams.Bucket +  
            "/" +  
            bucketParams.Key  
        );  
    } catch (err) {  
        console.log("Error", err);  
    }  
};  
run();
```

Upload the object.

```
// Import required AWS SDK clients and commands for Node.js.  
import { PutObjectCommand } from "@aws-sdk/client-s3";  
import { s3Client } from "./libs/s3Client.js"; // Helper function that creates an  
Amazon S3 service client module.  
import {path} from "path";
```

```
import {fs} from "fs";

const file = "OBJECT_PATH_AND_NAME"; // Path to and name of object. For example
'./myFiles/index.js'.
const fileStream = fs.createReadStream(file);

// Set the parameters
export const uploadParams = {
  Bucket: "BUCKET_NAME",
  // Add the required 'Key' parameter using the 'path' module.
  Key: path.basename(file),
  // Add the required 'Body' parameter
  Body: fileStream,
};

// Upload file to specified bucket.
export const run = async () => {
  try {
    const data = await s3Client.send(new PutObjectCommand(uploadParams));
    console.log("Success", data);
    return data; // For unit tests.
  } catch (err) {
    console.log("Error", err);
  }
};
run();
```

- For more information, see [AWS SDK for JavaScript Developer Guide](#).
- For API details, see [PutObject](#) in [AWS SDK for JavaScript API Reference](#).

Kotlin

SDK for Kotlin

Note

This is prerelease documentation for a feature in preview release. It is subject to change.

Tip

To learn how to set up and run this example, see [GitHub](#).

```
suspend fun putS3Object(bucketName: String, objectKey: String, objectPath: String)
{
    val metadataVal = mutableMapOf<String, String>()
    metadataVal["myVal"] = "test"

    val request = PutObjectRequest {
        bucket = bucketName
        key = objectKey
        metadata = metadataVal
        body = File(objectPath).asByteStream()
    }

    S3Client { region = "us-east-1" }.use { s3 ->
        val response = s3.putObject(request)
        println("Tag information is ${response.eTag}")
    }
}
```

- For API details, see [PutObject](#) in *AWS SDK for Kotlin API reference*.

PHP

SDK for PHP

Tip

To learn how to set up and run this example, see [GitHub](#).

Upload an object to a bucket.

```
$s3client = new Aws\S3\S3Client(['region' => 'us-west-2', 'version' => 'latest']);

$file_name = "local-file-" . uniqid();
try {
    $s3client->putObject([
        'Bucket' => $bucket_name,
        'Key' => $file_name,
        'SourceFile' => 'testfile.txt'
    ]);
    echo "Uploaded $file_name to $bucket_name.\n";
} catch (Exception $exception) {
    echo "Failed to upload $file_name with error: " . $exception->getMessage();
    exit("Please fix error with file upload before continuing.");
}
```

- For API details, see [PutObject](#) in *AWS SDK for PHP API Reference*.

Python

SDK for Python (Boto3)

Tip

To learn how to set up and run this example, see [GitHub](#).

```
class ObjectWrapper:
    def __init__(self, s3_object):
        self.object = s3_object
        self.key = self.object.key

    def put(self, data):
        """
        Upload data to the object.

        :param data: The data to upload. This can either be bytes or a string. When
        this
                    argument is a string, it is interpreted as a file name, which
        is
                    opened in read bytes mode.
        """
        put_data = data
        if isinstance(data, str):
            try:
                put_data = open(data, 'rb')
            except IOError:
                logger.exception("Expected file name or binary data, got '%s'.",
data)
                raise
```

```
try:
    self.object.put(Body=put_data)
    self.object.wait_until_exists()
    logger.info(
        "Put object '%s' to bucket '%s'.", self.object.key,
        self.object.bucket_name)
except ClientError:
    logger.exception(
        "Couldn't put object '%s' to bucket '%s'.", self.object.key,
        self.object.bucket_name)
    raise
finally:
    if getattr(put_data, 'close', None):
        put_data.close()
```

- For API details, see [PutObject](#) in *AWS SDK for Python (Boto3) API Reference*.

Ruby

SDK for Ruby

Tip

To learn how to set up and run this example, see [GitHub](#).

Upload a file using a managed uploader (`Object.upload_file`).

```
require "aws-sdk-s3"

# Wraps Amazon S3 object actions.
class ObjectUploadFileWrapper
  attr_reader :object

  # @param object [Aws::S3::Object] An existing Amazon S3 object.
  def initialize(object)
    @object = object
  end

  # Uploads a file to an Amazon S3 object by using a managed uploader.
  #
  # @param file_path [String] The path to the file to upload.
  # @return [Boolean] True when the file is uploaded; otherwise false.
  def upload_file(file_path)
    @object.upload_file(file_path)
    true
  rescue Aws::Errors::ServiceError => e
    puts "Couldn't upload file #{file_path} to #{@object.key}. Here's why:
#{e.message}"
    false
  end
end

def run_demo
  bucket_name = "doc-example-bucket"
  object_key = "my-uploaded-file"
  file_path = "object_upload_file.rb"

  wrapper = ObjectUploadFileWrapper.new(Aws::S3::Object.new(bucket_name,
object_key))
  return unless wrapper.upload_file(file_path)

  puts "File #{file_path} successfully uploaded to #{bucket_name}:#{object_key}."
end
```

```
run_demo if $PROGRAM_NAME == __FILE__
```

Upload a file using Object.put.

```
require "aws-sdk-s3"

# Wraps Amazon S3 object actions.
class ObjectPutWrapper
  attr_reader :object

  # @param object [Aws::S3::Object] An existing Amazon S3 object.
  def initialize(object)
    @object = object
  end

  def put_object(source_file_path)
    File.open(source_file_path, "rb") do |file|
      @object.put(body: file)
    end
    true
  rescue Aws::Errors::ServiceError => e
    puts "Couldn't put #{source_file_path} to #{object.key}. Here's why: #{e.message}"
    false
  end
end

def run_demo
  bucket_name = "doc-example-bucket"
  object_key = "my-object-key"
  file_path = "my-local-file.txt"

  wrapper = ObjectPutWrapper.new(Aws::S3::Object.new(bucket_name, object_key))
  success = wrapper.put_object(file_path)
  return unless success

  puts "Put file #{file_path} into #{object_key} in #{bucket_name}."
end

run_demo if $PROGRAM_NAME == __FILE__
```

Upload a file using Object.put and add server-side encryption.

```
require "aws-sdk-s3"

# Wraps Amazon S3 object actions.
class ObjectPutSseWrapper
  attr_reader :object

  # @param object [Aws::S3::Object] An existing Amazon S3 object.
  def initialize(object)
    @object = object
  end

  def put_object_encrypted(object_content, encryption)
    @object.put(body: object_content, server_side_encryption: encryption)
    true
  rescue Aws::Errors::ServiceError => e
    puts "Couldn't put your content to #{object.key}. Here's why: #{e.message}"
    false
  end
end
```

```
    end
end

def run_demo
  bucket_name = "doc-example-bucket"
  object_key = "my-encrypted-content"
  object_content = "This is my super-secret content."
  encryption = "AES256"

  wrapper = ObjectPutSseWrapper.new(Aws::S3::Object.new(bucket_name,
object_content))
  return unless wrapper.put_object_encrypted(object_content, encryption)

  puts "Put your content into #{bucket_name}:#{object_key} and encrypted it with
#{encryption}."
end

run_demo if $PROGRAM_NAME == __FILE__
```

- For API details, see [PutObject](#) in *AWS SDK for Ruby API Reference*.

Rust

SDK for Rust

Note

This documentation is for an SDK in preview release. The SDK is subject to change and should not be used in production.

Tip

To learn how to set up and run this example, see [GitHub](#).

```
pub async fn upload_object(
    client: &Client,
    bucket_name: &str,
    file_name: &str,
    key: &str,
) -> Result<(), Error> {
    let body = ByteStream::from_path(Path::new(file_name)).await;
    client
        .put_object()
        .bucket(bucket_name)
        .key(key)
        .body(body.unwrap())
        .send()
        .await?;

    println!("Uploaded file: {}", file_name);
    Ok(())
}
```

- For API details, see [PutObject](#) in *AWS SDK for Rust API reference*.

Swift

SDK for Swift

Note

This is prerelease documentation for an SDK in preview release. It is subject to change.

Tip

To learn how to set up and run this example, see [GitHub](#).

Upload a file from local storage to a bucket.

```
public func uploadFile(bucket: String, key: String, file: String) async throws {
    let fileUrl = URL(fileURLWithPath: file)
    let fileData = try Data(contentsOf: fileUrl)
    let dataStream = ByteStream.from(data: fileData)

    let input = PutObjectInput(
        body: dataStream,
        bucket: bucket,
        key: key
    )
    _ = try await client.putObject(input: input)
}
```

Upload the contents of a Swift Data object to a bucket.

```
public func createFile(bucket: String, key: String, withData data: Data) async throws {
    let dataStream = ByteStream.from(data: data)

    let input = PutObjectInput(
        body: dataStream,
        bucket: bucket,
        key: key
    )
    _ = try await client.putObject(input: input)
}
```

- For API details, see [PutObject](#) in [AWS SDK for Swift API reference](#).

For a complete list of AWS SDK developer guides and code examples, see [Using this service with an AWS SDK \(p. 1185\)](#). This topic also includes information about getting started and details about previous SDK versions.

Scenarios for Amazon S3 using AWS SDKs

The following code examples show you how to implement common scenarios in Amazon S3 with AWS SDKs. These scenarios show you how to accomplish specific tasks by calling multiple functions within Amazon S3. Each scenario includes a link to GitHub, where you can find instructions on how to set up and run the code.

Examples

- [Create a presigned URL for Amazon S3 using an AWS SDK \(p. 1450\)](#)
- [Getting started with Amazon S3 buckets and objects using an AWS SDK \(p. 1457\)](#)
- [Manage versioned Amazon S3 objects in batches with a Lambda function using an AWS SDK \(p. 1492\)](#)
- [Use a transfer manager to upload and download files to and from Amazon S3 using an AWS SDK \(p. 1492\)](#)
- [Work with Amazon S3 versioned objects using an AWS SDK \(p. 1506\)](#)

Create a presigned URL for Amazon S3 using an AWS SDK

The following code examples show how to create a presigned URL for S3 and upload an object.

Go

SDK for Go V2

Tip

To learn how to set up and run this example, see [GitHub](#).

```
// Get a presigned URL for the object.  
// In order to get a presigned URL for an object, you must  
// create a Presignclient  
fmt.Println("Create Presign client")  
presignClient := s3.NewPresignClient(&client)  
  
presignResult, err := presignClient.PresignGetObject(context.TODO(),  
&s3.GetObjectInput{  
    Bucket: aws.String(name),  
    Key:    aws.String("path/myfile.jpg"),  
})  
  
if err != nil {  
    panic("Couldn't get presigned URL for GetObject")  
}  
  
fmt.Printf("Presigned URL For object: %s\n", presignResult.URL)
```

Java

SDK for Java 2.x

Tip

To learn how to set up and run this example, see [GitHub](#).

```
public static void signBucket(S3Presigner presigner, String bucketName, String  
keyName) {  
  
    try {  
        PutObjectRequest objectRequest = PutObjectRequest.builder()  
            .bucket(bucketName)  
            .key(keyName)  
            .contentType("text/plain")  
            .build();  
  
        PutObjectPresignRequest presignRequest =  
PutObjectPresignRequest.builder()  
            .signatureDuration(Duration.ofMinutes(10))  
            .putObjectRequest(objectRequest)  
            .build();  
  
        PresignedPutObjectRequest presignedRequest =  
presigner.presignPutObject(presignRequest);  
        String myURL = presignedRequest.url().toString();  
        System.out.println("Presigned URL to upload a file to: " +myURL);  
    }  
}
```

```
System.out.println("Which HTTP method needs to be used when uploading a
file: " + presignedRequest.httpRequest().method());

// Upload content to the Amazon S3 bucket by using this URL.
URL url = presignedRequest.url();

// Create the connection and use it to upload the new object by using
the presigned URL.
HttpURLConnection connection = (HttpURLConnection)
url.openConnection();
connection.setDoOutput(true);
connection.setRequestProperty("Content-Type", "text/plain");
connection.setRequestMethod("PUT");
OutputStreamWriter out = new
OutputStreamWriter(connection.getOutputStream());
out.write("This text was uploaded as an object by using a presigned
URL.");
out.close();

connection.getResponseCode();
System.out.println("HTTP response code is " +
connection.getResponseCode());

} catch (S3Exception | IOException e) {
e.printStackTrace();
}
}
```

JavaScript

SDK for JavaScript V3

Tip

To learn how to set up and run this example, see [GitHub](#).

Create the client.

```
// Create service client module using ES6 syntax.
import { S3Client } from "@aws-sdk/client-s3";
// Set the AWS Region.
const REGION = "us-east-1";
// Create an Amazon S3 service client object.
const s3Client = new S3Client({ region: REGION });
export { s3Client };
```

Create a presigned URL to upload an object to a bucket.

```
// Import the required AWS SDK clients and commands for Node.js
import {
  CreateBucketCommand,
  DeleteObjectCommand,
  PutObjectCommand,
  DeleteBucketCommand }
from "@aws-sdk/client-s3";
import { s3Client } from "./libs/s3Client.js"; // Helper function that creates an
  Amazon S3 service client module.
import { getSignedUrl } from "@aws-sdk/s3-request-presigner";
import fetch from "node-fetch";

// Set parameters
```

```
// Create a random name for the Amazon Simple Storage Service (Amazon S3) bucket
// and key
export const bucketParams = {
  Bucket: `test-bucket-${Math.ceil(Math.random() * 10 ** 10)}`,
  Key: `test-object-${Math.ceil(Math.random() * 10 ** 10)}`,
  Body: "BODY"
};
export const run = async () => {
  try {
    // Create an S3 bucket.
    console.log(`Creating bucket ${bucketParams.Bucket}`);
    await s3Client.send(new CreateBucketCommand({ Bucket: bucketParams.Bucket }));
    console.log(`Waiting for "${bucketParams.Bucket}" bucket creation...`);
  } catch (err) {
    console.log("Error creating bucket", err);
  }
  try {
    // Create a command to put the object in the S3 bucket.
    const command = new PutObjectCommand(bucketParams);
    // Create the presigned URL.
    const signedUrl = await getSignedUrl(s3Client, command, {
      expiresIn: 3600,
    });
    console.log(
      `\nPutting "${bucketParams.Key}" using signedUrl with body
"${bucketParams.Body}" in v3`
    );
    console.log(signedUrl);
    const response = await fetch(signedUrl, {method: 'PUT', body:
      bucketParams.Body});
    console.log(
      `\nResponse returned by signed URL: ${await response.text()}\n`
    );
  } catch (err) {
    console.log("Error creating presigned URL", err);
  }
  try {
    // Delete the object.
    console.log(`\nDeleting object "${bucketParams.Key}" from bucket`);
    await s3Client.send(
      new DeleteObjectCommand({ Bucket: bucketParams.Bucket, Key:
        bucketParams.Key })
    );
  } catch (err) {
    console.log("Error deleting object", err);
  }
  try {
    // Delete the S3 bucket.
    console.log(`\nDeleting bucket ${bucketParams.Bucket}`);
    await s3Client.send(
      new DeleteBucketCommand({ Bucket: bucketParams.Bucket })
    );
  } catch (err) {
    console.log("Error deleting bucket", err);
  }
};
run();
```

Create a presigned URL to download an object from a bucket.

```
// Import the required AWS SDK clients and commands for Node.js
import {
  CreateBucketCommand,
  PutObjectCommand,
```

```
GetObjectCommand,
DeleteObjectCommand,
DeleteBucketCommand }
from "@aws-sdk/client-s3";
import { s3Client } from "./libs/s3Client.js"; // Helper function that creates an
Amazon S3 service client module.
import { getSignedUrl } from "@aws-sdk/s3-request-presigner";
const fetch = require("node-fetch");

// Set parameters
// Create a random names for the S3 bucket and key.
export const bucketParams = {
  Bucket: `test-bucket-${Math.ceil(Math.random() * 10 ** 10)}`,
  Key: `test-object-${Math.ceil(Math.random() * 10 ** 10)}`,
  Body: "BODY"
};

export const run = async () => {
  // Create an S3 bucket.
  try {
    console.log(`Creating bucket ${bucketParams.Bucket}`);
    const data = await s3Client.send(
      new CreateBucketCommand({ Bucket: bucketParams.Bucket })
    );
    return data; // For unit tests.
    console.log(`Waiting for "${bucketParams.Bucket}" bucket creation...\n`);
  } catch (err) {
    console.log("Error creating bucket", err);
  }
  // Put the object in the S3 bucket.
  try {
    console.log(`Putting object "${bucketParams.Key}" in bucket`);
    const data = await s3Client.send(
      new PutObjectCommand({
        Bucket: bucketParams.Bucket,
        Key: bucketParams.Key,
        Body: bucketParams.Body,
      })
    );
    return data; // For unit tests.
  } catch (err) {
    console.log("Error putting object", err);
  }
  // Create a presigned URL.
  try {
    // Create the command.
    const command = new GetObjectCommand(bucketParams);

    // Create the presigned URL.
    const signedUrl = await getSignedUrl(s3Client, command, {
      expiresIn: 3600,
    });
    console.log(
      `\nGetting "${bucketParams.Key}" using signedUrl with body
"${bucketParams.Body}" in v3`
    );
    console.log(signedUrl);
    const response = await fetch(signedUrl);
    console.log(
      `\nResponse returned by signed URL: ${await response.text()}\n`
    );
  } catch (err) {
    console.log("Error creating presigned URL", err);
  }
  // Delete the object.
  try {
```

```
    console.log(`\nDeleting object "${bucketParams.Key}" from bucket`);
    const data = await s3Client.send(
      new DeleteObjectCommand({ Bucket: bucketParams.Bucket, Key:
        bucketParams.Key })
    );
    return data; // For unit tests.
  } catch (err) {
    console.log("Error deleting object", err);
  }
  // Delete the S3 bucket.
  try {
    console.log(`\nDeleting bucket ${bucketParams.Bucket}`);
    const data = await s3Client.send(
      new DeleteBucketCommand({ Bucket: bucketParams.Bucket, Key:
        bucketParams.Key })
    );
    return data; // For unit tests.
  } catch (err) {
    console.log("Error deleting object", err);
  }
}
run();
```

- For more information, see [AWS SDK for JavaScript Developer Guide](#).

Python

SDK for Python (Boto3)

Tip

To learn how to set up and run this example, see [GitHub](#).

Generate a presigned URL that can perform an S3 action for a limited time. Use the Requests package to make a request with the URL.

```
import argparse
import logging
import boto3
from botocore.exceptions import ClientError
import requests

logger = logging.getLogger(__name__)

def generate_presigned_url(s3_client, client_method, method_parameters,
                           expires_in):
    """
    Generate a presigned Amazon S3 URL that can be used to perform an action.

    :param s3_client: A Boto3 Amazon S3 client.
    :param client_method: The name of the client method that the URL performs.
    :param method_parameters: The parameters of the specified client method.
    :param expires_in: The number of seconds the presigned URL is valid for.
    :return: The presigned URL.
    """
    try:
        url = s3_client.generate_presigned_url(
            ClientMethod=client_method,
            Params=method_parameters,
            ExpiresIn=expires_in
        )
        logger.info("Got presigned URL: %s", url)
    except ClientError:
```

```

        logger.exception(
            "Couldn't get a presigned URL for client method '%s'.", client_method)
        raise
    return url

def usage_demo():
    logging.basicConfig(level=logging.INFO, format='%(levelname)s: %(message)s')

    print('*'*88)
    print("Welcome to the Amazon S3 presigned URL demo.")
    print('*'*88)

    parser = argparse.ArgumentParser()
    parser.add_argument('bucket', help="The name of the bucket.")
    parser.add_argument(
        'key', help="For a GET operation, the key of the object in Amazon S3. For a "
        "PUT operation, the name of a file to upload.")
    parser.add_argument(
        'action', choices=('get', 'put'), help="The action to perform.")
    args = parser.parse_args()

    s3_client = boto3.client('s3')
    client_action = 'get_object' if args.action == 'get' else 'put_object'
    url = generate_presigned_url(
        s3_client, client_action, {'Bucket': args.bucket, 'Key': args.key}, 1000)

    print("Using the Requests package to send a request to the URL.")
    response = None
    if args.action == 'get':
        response = requests.get(url)
    elif args.action == 'put':
        print("Putting data to the URL.")
        try:
            with open(args.key, 'r') as object_file:
                object_text = object_file.read()
            response = requests.put(url, data=object_text)
        except FileNotFoundError:
            print(f"Couldn't find {args.key}. For a PUT operation, the key must be "
            "the "
            f"name of a file that exists on your computer.")

    if response is not None:
        print("Got response:")
        print(f"Status: {response.status_code}")
        print(response.text)

    print('*'*88)

if __name__ == '__main__':
    usage_demo()

```

Generate a presigned POST request to upload a file.

```

class BucketWrapper:
    def __init__(self, bucket):
        self.bucket = bucket
        self.name = bucket.name

    def generate_presigned_post(self, object_key, expires_in):
        """
        Generate a presigned Amazon S3 POST request to upload a file.

```

AWS A presigned POST can be used for a limited time to let someone without an account upload a file to a bucket.

```
:param object_key: The object key to identify the uploaded object.
:param expires_in: The number of seconds the presigned POST is valid.
:return: A dictionary that contains the URL and form fields that contain required access data.

"""
try:
    response = self.bucket.meta.client.generate_presigned_post(
        Bucket=self.bucket.name, Key=object_key, ExpiresIn=expires_in)
    logger.info("Got presigned POST URL: %s", response['url'])
except ClientError:
    logger.exception(
        "Couldn't get a presigned POST URL for bucket '%s' and object '%s'",
        self.bucket.name, object_key)
    raise
return response
```

Ruby

SDK for Ruby

Tip

To learn how to set up and run this example, see [GitHub](#).

```
require "aws-sdk-s3"
require "net/http"

# Creates a presigned URL that can be used to upload content to an object.
#
# @param bucket [Aws::S3::Bucket] An existing Amazon S3 bucket.
# @param object_key [String] The key to give the uploaded object.
# @return [URI, nil] The parsed URI if successful; otherwise nil.
def get_presigned_url(bucket, object_key)
    url = bucket.object(object_key).presigned_url(:put)
    puts "Created presigned URL: #{url}."
    URI(url)
rescue Aws::Errors::ServiceError => e
    puts "Couldn't create presigned URL for #{bucket.name}:#{object_key}. Here's why: #{e.message}"
end

def run_demo
    bucket_name = "doc-example-bucket"
    object_key = "my-file.txt"
    object_content = "This is the content of my-file.txt."

    bucket = Aws::S3::Bucket.new(bucket_name)
    presigned_url = get_presigned_url(bucket, object_key)
    return unless presigned_url

    response = Net::HTTP.start(presigned_url.host) do |http|
        http.send_request("PUT", presigned_url.request_uri, object_content,
"content_type" => "")
    end

    case response
    when Net::HTTPSuccess
        puts "Content uploaded!"
```

```
    else
        puts response.value
    end
end

run_demo if $PROGRAM_NAME == __FILE__
```

For a complete list of AWS SDK developer guides and code examples, see [Using this service with an AWS SDK \(p. 1185\)](#). This topic also includes information about getting started and details about previous SDK versions.

Getting started with Amazon S3 buckets and objects using an AWS SDK

The following code examples show how to:

- Create a bucket.
- Upload a file to the bucket.
- Download an object from a bucket.
- Copy an object to a subfolder in a bucket.
- List the objects in a bucket.
- Delete the objects in a bucket.
- Delete a bucket.

.NET

AWS SDK for .NET

Tip

To learn how to set up and run this example, see [GitHub](#).

```
using System;
using System.IO;
using System.Threading.Tasks;
using Amazon.S3;

public class S3_Basics
{
    public static async Task Main()
    {
        // Create an Amazon S3 client object. The constructor uses the
        // default user installed on the system. To work with Amazon S3
        // features in a different AWS Region, pass the AWS Region as a
        // parameter to the client constructor.
        IAmazonS3 client = new AmazonS3Client();
        string bucketName = string.Empty;
        string filePath = string.Empty;
        string keyName = string.Empty;

        Console.WriteLine("Amazon Simple Storage Service (Amazon S3) basic");
        Console.WriteLine("procedures. This application will:");
        Console.WriteLine("\n\t1. Create a bucket");
        Console.WriteLine("\n\t2. Upload an object to the new bucket");
        Console.WriteLine("\n\t3. Copy the uploaded object to a folder in the
bucket");
        Console.WriteLine("\n\t4. List the items in the new bucket");
```

```
Console.WriteLine("\n\t5. Delete all the items in the bucket");
Console.WriteLine("\n\t6. Delete the bucket");

Console.WriteLine("-----");

// Create a bucket.
Console.WriteLine("\nCreate a new Amazon S3 bucket.\n");

Console.Write("Please enter a name for the new bucket: ");
bucketName = Console.ReadLine();

var success = await S3Bucket.CreateBucketAsync(client, bucketName);
if (success)
{
    Console.WriteLine($"Successfully created bucket: {bucketName}.\n");
}
else
{
    Console.WriteLine($"Could not create bucket: {bucketName}.\n");
}

Console.WriteLine("Upload a file to the new bucket.");

// Get the local path and filename for the file to upload.
while (string.IsNullOrEmpty(filePath))
{
    Console.Write("Please enter the path and filename of the file to
upload: ");
    filePath = Console.ReadLine();

    // Confirm that the file exists on the local computer.
    if (!File.Exists(filePath))
    {
        Console.WriteLine($"Couldn't find {filePath}. Try again.\n");
        filePath = string.Empty;
    }
}

// Get the file name from the full path.
keyName = Path.GetFileName(filePath);

success = await S3Bucket.UploadFileAsync(client, bucketName, keyName,
filePath);

if (success)
{
    Console.WriteLine($"Successfully uploaded {keyName} from {filePath}
to {bucketName}.\n");
}
else
{
    Console.WriteLine($"Could not upload {keyName}.\n");
}

// Set the file path to an empty string to avoid overwriting the
// file we just uploaded to the bucket.
filePath = string.Empty;

// Now get a new location where we can save the file.
while (string.IsNullOrEmpty(filePath))
{
    // First get the path to which the file will be downloaded.
    Console.Write("Please enter the path where the file will be
downloaded: ");
    filePath = Console.ReadLine();
```

```

        // Confirm that the file exists on the local computer.
        if (File.Exists($"{filePath}\\{keyName}"))
        {
            Console.WriteLine($"Sorry, the file already exists in that
location.\n");
            filePath = string.Empty;
        }
    }

    // Download an object from a bucket.
    success = await S3Bucket.DownloadObjectFromBucketAsync(client,
bucketName, keyName, filePath);

    if (success)
    {
        Console.WriteLine($"Successfully downloaded {keyName}.\n");
    }
    else
    {
        Console.WriteLine($"Sorry, could not download {keyName}.\n");
    }

    // Copy the object to a different folder in the bucket.
    string folderName = string.Empty;

    while (string.IsNullOrEmpty(folderName))
    {
        Console.Write("Please enter the name of the folder to copy your
object to: ");
        folderName = Console.ReadLine();
    }

    while (string.IsNullOrEmpty(keyName))
    {
        // Get the name to give to the object once uploaded.
        Console.Write("Enter the name of the object to copy: ");
        keyName = Console.ReadLine();
    }

    await S3Bucket.CopyObjectInBucketAsync(client, bucketName, keyName,
folderName);

    // List the objects in the bucket.
    await S3Bucket.ListBucketContentsAsync(client, bucketName);

    // Delete the contents of the bucket.
    await S3Bucket.DeleteBucketContentsAsync(client, bucketName);

    // Deleting the bucket too quickly after deleting its contents will
    // cause an error that the bucket isn't empty. So...
    Console.WriteLine("Press <Enter> when you are ready to delete the
bucket.");
    _ = Console.ReadLine();

    // Delete the bucket.
    await S3Bucket.DeleteBucketAsync(client, bucketName);
}
}

```

- For API details, see the following topics in *AWS SDK for .NET API Reference*.

- [CopyObject](#)
- [CreateBucket](#)

- [DeleteBucket](#)
- [DeleteObjects](#)
- [GetObject](#)
- [ListObjects](#)
- [PutObject](#)

C++

SDK for C++

Tip

To learn how to set up and run this example, see [GitHub](#).

```
#include <iostream>
#include <aws/core/Aws.h>
#include <aws/s3/S3Client.h>
#include <aws/s3/model/CopyObjectRequest.h>
#include <aws/s3/model/CreateBucketRequest.h>
#include <aws/s3/model/DeleteBucketRequest.h>
#include <aws/s3/model/DeleteObjectRequest.h>
#include <aws/s3/model/GetObjectRequest.h>
#include <aws/s3/model/ListObjectsRequest.h>
#include <aws/s3/model/PutObjectRequest.h>
#include <aws/s3/model/BucketLocationConstraint.h>
#include <aws/s3/model/CreateBucketConfiguration.h>
#include <aws/core/utils/UUID.h>
#include <aws/core/utils/StringUtils.h>
#include <aws/core/utils/memory/stl/AWSAllocator.h>
#include <aws/core/utils/memory/stl/AWSStreamFwd.h>
#include <fstream>
#include "awsdoc/s3/s3_examples.h"

namespace AwsDoc {
    namespace S3 {

        //! Delete an S3 bucket.
        /**
         * \sa DeleteBucket()
         * \param bucketName the S3 bucket's name.
         * \param client an S3 client.
         * \param logProgress enables verbose logging.
        */
        static bool DeleteBucket(const Aws::String &bucketName, Aws::S3::S3Client &client, bool logProgress);

        //! Delete an object in an S3 bucket.
        /**
         * \sa DeleteObjectFromBucket()
         * \param bucketName the S3 bucket's name.
         * \param key the key for the object in the S3 bucket.
         * \param client an S3 client.
         * \param logProgress enables verbose logging.
        */
        static bool
        DeleteObjectFromBucket(const Aws::String &bucketName, const Aws::String &key, Aws::S3::S3Client &client,
                               bool logProgress);
    }
}
```

```
///! Scenario to create, copy, and delete S3 buckets and objects.
/*
\sa S3_GettingStartedScenario()
\param uploadFilePath path to file to upload to an S3 bucket.
\param saveFilePath path for saving a downloaded S3 object.
\param clientConfig Aws client configuration.
\param logProgress enables verbose logging.
*/
bool AwsDoc::S3::S3_GettingStartedScenario(const Aws::String &uploadFilePath, const
                                             Aws::String &saveFilePath,
                                             const Aws::Client::ClientConfiguration
                                             &clientConfig,
                                             bool logProgress) {

    Aws::S3::S3Client client(clientConfig);

    // Create a unique bucket name which is only temporary and will be deleted.
    // Format: "doc-example-bucket-" + lowercase UUID.
    Aws::String uuid = Aws::Utils::UUID::RandomUUID();
    Aws::String bucketName = "doc-example-bucket-" +
        Aws::Utils::StringUtils::ToLower(uuid.c_str());

    // 1. Create a bucket.
    {
        Aws::S3::Model::CreateBucketRequest request;
        request.SetBucket(bucketName);

        if (clientConfig.region != Aws::Region::US_EAST_1) {
            Aws::S3::Model::CreateBucketConfiguration createBucketConfiguration;
            createBucketConfiguration.WithLocationConstraint(
                Aws::S3::Model::BucketLocationConstraintMapper::GetBucketLocationConstraintForName(
                    clientConfig.region));
            request.WithCreateBucketConfiguration(createBucketConfiguration);
        }

        Aws::S3::Model::CreateBucketOutcome outcome = client.CreateBucket(request);

        if (!outcome.IsSuccess()) {
            const Aws::S3::S3Error &err = outcome.GetError();
            std::cerr << "Error: CreateBucket: " <<
                err.GetExceptionName() << ":" << err.GetMessage() <<
            std::endl;
            return false;
        }
        else if (logProgress) {
            std::cout << "Created the bucket, '" << bucketName <<
                "', in the region, '" << clientConfig.region << "'." <<
            std::endl;
        }
    }

    // 2. Upload a local file to the bucket.
    Aws::String key = "key-for-test";
    {
        Aws::S3::Model::PutObjectRequest request;
        request.SetBucket(bucketName);
        request.SetKey(key);

        std::shared_ptr<Aws::FStream> input_data =
            Aws::MakeShared<Aws::FStream>("SampleAllocationTag",
                                         uploadFilePath,
                                         std::ios_base::in |
                                         std::ios_base::binary);

        if (!input_data->is_open()) {

```

```

        std::cout << "Error: unable to open file, '" << uploadFilePath << "'";
<< std::endl;
        AwsDoc::S3::DeleteBucket(bucketName, client, logProgress);
        return false;
    }

    request.SetBody(input_data);

    Aws::S3::Model::PutObjectOutcome outcome =
        client.PutObject(request);

    if (!outcome.IsSuccess()) {
        std::cout << "Error: PutObject: " <<
            outcome.GetError().GetMessage() << std::endl;
        AwsDoc::S3::DeleteObjectFromBucket(bucketName, key, client,
logProgress);
        AwsDoc::S3::DeleteBucket(bucketName, client, logProgress);
        return false;
    }
    else if (logProgress) {
        std::cout << "Added the object with the key, '" << key << "', to the
bucket, ''"
                << bucketName << "." << std::endl;
    }
}

// 3. Download the object to a local file.
{
    Aws::S3::Model::GetObjectRequest request;
    request.SetBucket(bucketName);
    request.SetKey(key);

    Aws::S3::Model::GetObjectOutcome outcome =
        client.GetObject(request);

    if (!outcome.IsSuccess()) {
        const Aws::S3::S3Error &err = outcome.GetError();
        std::cout << "Error: GetObject: " <<
            err.GetExceptionName() << ":" << err.GetMessage() <<
std::endl;
    }
    else {
        if (logProgress) {
            std::cout << "Downloaded the object with the key, '" << key << "'",
in the bucket, ''"
                    << bucketName << "." << std::endl;
        }

        Aws::IOStream &ioStream = outcome.GetResultWithOwnership().
            GetBody();
        Aws::OFStream outStream(saveFilePath);
        if (!outStream.is_open()) {
            std::cout << "Error: unable to open file, '" << saveFilePath <<
"'" << std::endl;
        }
        else if (logProgress) {
            outStream << ioStream.rdbuf();
            std::cout << "Wrote the downloaded object to the file ''"
                    << saveFilePath << "." << std::endl;
        }
    }
}

// 4. Copy the object to a different "folder" in the bucket.
Aws::String copiedToKey = "test-folder/" + key;
{

```

```

Aws::S3::Model::CopyObjectRequest request;
request.WithBucket(bucketName)
    .WithKey(copiedToKey)
    .WithCopySource(bucketName + "/" + key);

Aws::S3::Model::CopyObjectOutcome outcome =
    client.CopyObject(request);
if (!outcome.IsSuccess()) {
    std::cout << "Error: CopyObject: " <<
        outcome.GetError().GetMessage() << std::endl;
}
else if (logProgress) {
    std::cout << "Copied the object with the key, '" << key << "', to the
key, '" << copiedToKey
                << ", in the bucket, '" << bucketName << "'." << std::endl;
}
}

// 5. List objects in the bucket.
{
    Aws::S3::Model::ListObjectsRequest request;
    request.WithBucket(bucketName);

    Aws::S3::Model::ListObjectsOutcome outcome = client.ListObjects(request);

    if (!outcome.IsSuccess()) {
        std::cout << "Error: ListObjects: " <<
            outcome.GetError().GetMessage() << std::endl;
    }
    else if (logProgress) {
        Aws::Vector<Aws::S3::Model::Object> objects =
            outcome.GetResult().GetContents();

        std::cout << objects.size() << " objects in the bucket, '" <<
bucketName << ":" << std::endl;

        for (Aws::S3::Model::Object &object: objects) {
            std::cout << "      '" << object.GetKey() << "' " << std::endl;
        }
    }
}

// 6. Delete all objects in the bucket.
// All objects in the bucket must be deleted before deleting the bucket.
AwsDoc::S3::DeleteObjectFromBucket(bucketName, copiedToKey, client,
logProgress);
AwsDoc::S3::DeleteObjectFromBucket(bucketName, key, client, logProgress);

// 7. Delete the bucket.
return AwsDoc::S3::DeleteBucket(bucketName, client, logProgress);
}

bool AwsDoc::S3::DeleteObjectFromBucket(const Aws::String &bucketName, const
Aws::String &key,
                                         Aws::S3::S3Client &client, bool logProgress) {
    Aws::S3::Model::DeleteObjectRequest request;
    request.SetBucket(bucketName);
    request.SetKey(key);

    Aws::S3::Model::DeleteObjectOutcome outcome =
        client.DeleteObject(request);

    if (!outcome.IsSuccess()) {
        std::cout << "Error: DeleteObject: " <<
            outcome.GetError().GetMessage() << std::endl;
    return false;
}

```

```

        }
        else if (logProgress) {
            std::cout << "Deleted the object with the key, '" << key << "', from the
bucket, ''                                << bucketName << "." << std::endl;
        }

        return true;
    }

bool AwsDoc::S3::DeleteBucket(const Aws::String &bucketName, Aws::S3::S3Client
&client, bool logProgress) {
    Aws::S3::Model::DeleteBucketRequest request;
    request.SetBucket(bucketName);

    Aws::S3::Model::DeleteBucketOutcome outcome =
        client.DeleteBucket(request);

    if (!outcome.IsSuccess()) {
        const Aws::S3::S3Error &err = outcome.GetError();
        std::cerr << "Error: DeleteBucket: " <<
                    err.GetExceptionName() << ":" << err.GetMessage() << std::endl;
        return false;
    }
    else if (logProgress) {
        std::cout << "Deleted the bucket, '" << bucketName << "." << std::endl;
    }
    return true;
}

```

- For API details, see the following topics in *AWS SDK for C++ API Reference*.
 - [CopyObject](#)
 - [CreateBucket](#)
 - [DeleteBucket](#)
 - [DeleteObjects](#)
 - [GetObject](#)
 - [ListObjects](#)
 - [PutObject](#)

Go

SDK for Go V2

Tip

To learn how to set up and run this example, see [GitHub](#).

```

// This bucket name is 100% unique.
// Remember that bucket names must be globally unique among all buckets.

myBucketName := "mybucket-" + (xid.New()).String()
fmt.Printf("Bucket name: %v\n", myBucketName)

cfg, err := config.LoadDefaultConfig(context.TODO())

if err != nil {
    panic("Failed to load configuration")
}

```

```
s3client := s3.NewFromConfig(cfg)

MakeBucket(*s3client, myBucketName)
BucketOps(*s3client, myBucketName)
AccountBucketOps(*s3client, myBucketName)
BucketDelOps(*s3client, myBucketName)
```

- For API details, see the following topics in *AWS SDK for Go API Reference*.
 - [CopyObject](#)
 - [CreateBucket](#)
 - [DeleteBucket](#)
 - [DeleteObjects](#)
 - [GetObject](#)
 - [ListObjects](#)
 - [PutObject](#)

Java

SDK for Java 2.x

Tip

To learn how to set up and run this example, see [GitHub](#).

```
/** 
 * Before running this Java V2 code example, set up your development environment,
 * including your credentials.
 *
 * For more information, see the following documentation topic:
 *
 * https://docs.aws.amazon.com/sdk-for-java/latest/developer-guide/get-started.html
 */

public class S3Scenario {

    public static void main(String[] args) throws IOException {

        final String usage = "\n" +
            "Usage:\n" +
            "      <bucketName> <key> <objectPath> <savePath> <toBucket>\n\n" +
            "Where:\n" +
            "      bucketName - The Amazon S3 bucket to create.\n\n" +
            "      key - The key to use.\n\n" +
            "      objectPath - The path where the file is located (for example, C:/AWS/book2.pdf). +
            "      savePath - The path where the file is saved after it's downloaded
            (for example, C:/AWS/book2.pdf). " +
            "      toBucket - An Amazon S3 bucket to where an object is copied to
            (for example, C:/AWS/book2.pdf). ";

        if (args.length != 5) {
            System.out.println(usage);
            System.exit(1);
        }

        String bucketName = args[0];
        String key = args[1];
        String objectPath = args[2];
        String savePath = args[3];
```

```
String toBucket = args[4] ;

ProfileCredentialsProvider credentialsProvider =
ProfileCredentialsProvider.create();
Region region = Region.US_EAST_1;
S3Client s3 = S3Client.builder()
.region(region)
.credentialsProvider(credentialsProvider)
.build();

// Create an Amazon S3 bucket.
createBucket(s3, bucketName);

// Update a local file to the Amazon S3 bucket.
uploadLocalFile(s3, bucketName, key, objectPath);

// Download the object to another local file.
getBytes(s3, bucketName, key, savePath);

// Perform a multipart upload.
String multipartKey = "multiPartKey";
multipartUpload(s3, toBucket, multipartKey);

// List all objects located in the Amazon S3 bucket.
// Show 2 ways
listAllObjects(s3, bucketName);
anotherListExample(s3, bucketName) ;

// Copy the object to another Amazon S3 bucket
copyBucketObject(s3, bucketName, key, toBucket);

// Delete the object from the Amazon S3 bucket.
deleteObjectFromBucket(s3, bucketName, key);

// Delete the Amazon S3 bucket
deleteBucket(s3, bucketName);
System.out.println("All Amazon S3 operations were successfully performed");
s3.close();
}

// Create a bucket by using a S3Waiter object
public static void createBucket( S3Client s3Client, String bucketName) {

try {
S3Waiter s3Waiter = s3Client.waiter();
CreateBucketRequest bucketRequest = CreateBucketRequest.builder()
.bucket(bucketName)
.build();

s3Client.createBucket(bucketRequest);
HeadBucketRequest bucketRequestWait = HeadBucketRequest.builder()
.bucket(bucketName)
.build();

// Wait until the bucket is created and print out the response.
WaiterResponse<HeadBucketResponse> waiterResponse =
s3Waiter.waitUntilBucketExists(bucketRequestWait);
waiterResponse.matched().response().ifPresent(System.out::println);
System.out.println(bucketName + " is ready");

} catch (S3Exception e) {
System.err.println(e.awsErrorDetails().errorMessage());
System.exit(1);
}
}
```

```
public static void deleteBucket(S3Client client, String bucket) {
    DeleteBucketRequest deleteBucketRequest = DeleteBucketRequest.builder()
        .bucket(bucket)
        .build();

    client.deleteBucket(deleteBucketRequest);
    System.out.println(bucket + " was deleted.");
}

/**
 * Upload an object in parts
 */
private static void multipartUpload(S3Client s3, String bucketName, String key)
{
    int mB = 1024 * 1024;
    // First create a multipart upload and get the upload id
    CreateMultipartUploadRequest createMultipartUploadRequest =
CreateMultipartUploadRequest.builder()
    .bucket(bucketName)
    .key(key)
    .build();

    CreateMultipartUploadResponse response =
s3.createMultipartUpload(createMultipartUploadRequest);
    String uploadId = response.uploadId();
    System.out.println(uploadId);

    // Upload all the different parts of the object
    UploadPartRequest uploadPartRequest1 = UploadPartRequest.builder()
        .bucket(bucketName)
        .key(key)
        .uploadId(uploadId)
        .partNumber(1).build();

    String etag1 = s3.uploadPart(uploadPartRequest1,
RequestBody.fromByteBuffer(getRandomByteBuffer(5 * mB))).eTag();
    CompletedPart part1 =
CompletedPart.builder().partNumber(1).eTag(etag1).build();

    UploadPartRequest uploadPartRequest2 =
UploadPartRequest.builder().bucket(bucketName).key(key)
        .uploadId(uploadId)
        .partNumber(2).build();
    String etag2 = s3.uploadPart(uploadPartRequest2,
RequestBody.fromByteBuffer(getRandomByteBuffer(3 * mB))).eTag();
    CompletedPart part2 =
CompletedPart.builder().partNumber(2).eTag(etag2).build();

    // Call completeMultipartUpload operation to tell S3 to merge all uploaded
    // parts and finish the multipart operation.
    CompletedMultipartUpload completedMultipartUpload =
CompletedMultipartUpload.builder()
    .parts(part1, part2)
    .build();

    CompleteMultipartUploadRequest completeMultipartUploadRequest =
CompleteMultipartUploadRequest.builder()
    .bucket(bucketName)
    .key(key)
    .uploadId(uploadId)
    .multipartUpload(completedMultipartUpload)
    .build();

    s3.completeMultipartUpload(completeMultipartUploadRequest);
}
```

```
private static ByteBuffer getRandomByteBuffer(int size) {
    byte[] b = new byte[size];
    new Random().nextBytes(b);
    return ByteBuffer.wrap(b);
}

// Return a byte array
private static byte[] getObjectFile(String filePath) {

    FileInputStream fileInputStream = null;
    byte[] bytesArray = null;

    try {
        File file = new File(filePath);
        bytesArray = new byte[(int) file.length()];
        fileInputStream = new FileInputStream(file);
        fileInputStream.read(bytesArray);

    } catch (IOException e) {
        e.printStackTrace();
    } finally {
        if (fileInputStream != null) {
            try {
                fileInputStream.close();
            } catch (IOException e) {
                e.printStackTrace();
            }
        }
    }
    return bytesArray;
}

public static void getObjectBytes (S3Client s3, String bucketName, String
keyName, String path ) {

    try {
        GetObjectRequest objectRequest = GetObjectRequest
            .builder()
            .key(keyName)
            .bucket(bucketName)
            .build();

        ResponseBytes<GetObjectResponse> objectBytes =
s3.getObjectAsBytes(objectRequest);
        byte[] data = objectBytes.asByteArray();

        // Write the data to a local file.
        File myFile = new File(path );
        OutputStream os = new FileOutputStream(myFile);
        os.write(data);
        System.out.println("Successfully obtained bytes from an S3 object");
        os.close();

    } catch (IOException ex) {
        ex.printStackTrace();
    } catch (S3Exception e) {
        System.err.println(e.awsErrorDetails().errorMessage());
        System.exit(1);
    }
}

public static void uploadLocalFile(S3Client s3, String bucketName, String key,
String objectPath) {
```

```
PutObjectRequest objectRequest = PutObjectRequest.builder()
    .bucket(bucketName)
    .key(key)
    .build();

s3.putObject(objectRequest,
RequestBody.fromBytes(getObjectFile(objectPath)));
}

public static void listAllObjects(S3Client s3, String bucketName) {

    ListObjectsV2Request listObjectsReqManual = ListObjectsV2Request.builder()
        .bucket(bucketName)
        .maxKeys(1)
        .build();

    boolean done = false;
    while (!done) {
        ListObjectsV2Response listObjResponse =
s3.listObjectsV2(listObjectsReqManual);
        for (S3Object content : listObjResponse.contents()) {
            System.out.println(content.key());
        }

        if (listObjResponse.nextContinuationToken() == null) {
            done = true;
        }

        listObjectsReqManual = listObjectsReqManual.toBuilder()
            .continuationToken(listObjResponse.nextContinuationToken())
            .build();
    }
}

public static void anotherListExample(S3Client s3, String bucketName) {

    ListObjectsV2Request listReq = ListObjectsV2Request.builder()
        .bucket(bucketName)
        .maxKeys(1)
        .build();

    ListObjectsV2Iterable listRes = s3.listObjectsV2Paginator(listReq);

    // Process response pages
    listRes.stream()
        .flatMap(r -> r.contents().stream())
        .forEach(content -> System.out.println(" Key: " + content.key() + " size = " + content.size()));

    // Helper method to work with paginated collection of items directly
    listRes.contents().stream()
        .forEach(content -> System.out.println(" Key: " + content.key() + " size = " + content.size()));

    for (S3Object content : listRes.contents()) {
        System.out.println(" Key: " + content.key() + " size = " +
content.size());
    }
}

public static void deleteObjectFromBucket(S3Client s3, String bucketName,
String key) {

    DeleteObjectRequest deleteObjectRequest = DeleteObjectRequest.builder()
        .bucket(bucketName)
```

```
        .key(key)
        .build();

    s3.deleteObject(deleteObjectRequest);
    System.out.println(key + " was deleted");
}

public static String copyBucketObject (S3Client s3, String fromBucket, String
objectKey, String toBucket) {

    String encodedUrl = null;
    try {
        encodedUrl = URLEncoder.encode(fromBucket + "/" + objectKey,
StandardCharsets.UTF_8.toString());
    } catch (UnsupportedEncodingException e) {
        System.out.println("URL could not be encoded: " + e.getMessage());
    }
    CopyObjectRequest copyReq = CopyObjectRequest.builder()
        .copySource(encodedUrl)
        .destinationBucket(toBucket)
        .destinationKey(objectKey)
        .build();

    try {
        CopyObjectResponse copyRes = s3.copyObject(copyReq);
        System.out.println("The " + objectKey + " was copied to "+toBucket);
        return copyRes.copyObjectResult().toString();
    } catch (S3Exception e) {
        System.err.println(e.awsErrorDetails().errorMessage());
        System.exit(1);
    }

    return "";
}
}
```

- For API details, see the following topics in *AWS SDK for Java 2.x API Reference*.
 - [CopyObject](#)
 - [CreateBucket](#)
 - [DeleteBucket](#)
 - [DeleteObjects](#)
 - [GetObject](#)
 - [ListObjects](#)
 - [PutObject](#)

JavaScript

SDK for JavaScript V3

Tip

To learn how to set up and run this example, see [GitHub](#).

```
import {
  CreateBucketCommand,
  PutObjectCommand,
  CopyObjectCommand,
  DeleteObjectCommand,
```

```

DeleteBucketCommand,
GetObjectCommand
} from "@aws-sdk/client-s3";
import { s3Client, REGION } from "../libs/s3Client.js"; // Helper function that
creates an Amazon S3 service client module.

if (process.argv.length < 5) {
  console.log(
    "Usage: node s3_basics.js <the bucket name> <the AWS Region to use> <object
name> <object content>\n" +
    "Example: node s3_basics_full.js test-bucket 'test.txt' 'Test Content'"
  );
}
const bucket_name = process.argv[2];
const object_key = process.argv[3];
const object_content = process.argv[4];

export const run = async (bucket_name, object_key, object_content) => {
  try {
    const create_bucket_params = {
      Bucket: bucket_name
    };
    console.log("\nCreating the bucket, named " + bucket_name + "...\\n");
    console.log("about to create");
    const data = await s3Client.send(
      new CreateBucketCommand(create_bucket_params)
    );
    console.log("Bucket created at ", data.Location);
    try {
      console.log(
        "\nCreated and uploaded an object named " +
        object_key +
        " to first bucket " +
        bucket_name +
        "...\\n"
      );
      // Set the parameters for the object to upload.
      const object_upload_params = {
        Bucket: bucket_name,
        // Specify the name of the new object. For example, 'test.html'.
        // To create a directory for the object, use '/'. For example, 'myApp/
package.json'.
        Key: object_key,
        // Content of the new object.
        Body: object_content,
      };
      // Create and upload the object to the first S3 bucket.
      await s3Client.send(new PutObjectCommand(object_upload_params));
      console.log(
        "Successfully uploaded object: " +
        object_upload_params.Bucket +
        "/" +
        object_upload_params.Key
      );
      try {
        const download_bucket_params = {
          Bucket: bucket_name,
          Key: object_key
        };
        console.log(
          "\nDownloading " +
          object_key +
          " from" +
          bucket_name +
          "...\\n"
        );
      }
    }
  }
}

```

```
// Create a helper function to convert a ReadableStream into a string.
const streamToString = (stream) =>
  new Promise((resolve, reject) => {
    const chunks = [];
    stream.on("data", (chunk) => chunks.push(chunk));
    stream.on("error", reject);
    stream.on("end", () =>
      resolve(Buffer.concat(chunks).toString("utf8")));
  });

// Get the object from the Amazon S3 bucket. It is returned as a
ReadableStream.
const data = await s3Client.send(new
GetObjectCommand(download_bucket_params));
// Convert the ReadableStream to a string.
const bodyContents = await streamToString(data.Body);
console.log(bodyContents);
try {
  // Copy the object from the first bucket to the second bucket.
  const copy_object_params = {
    Bucket: bucket_name,
    CopySource: "/" + bucket_name + "/" + object_key,
    Key: "copy-destination/" + object_key,
  };
  console.log(
    "\nCopying " +
    object_key +
    " from" +
    bucket_name +
    " to " +
    bucket_name +
    "/" +
    copy_object_params.Key +
    " ...\n"
  );
  await s3Client.send(new CopyObjectCommand(copy_object_params));
  console.log("Success, object copied to folder.");
  try {
    console.log("\nDeleting " + object_key + " from" + bucket_name);
    const delete_object_from_bucket_params = {
      Bucket: bucket_name,
      Key: object_key,
    };
    await s3Client.send(
      new DeleteObjectCommand(delete_object_from_bucket_params)
    );
    console.log("Success. Object deleted from bucket.");
    try {
      console.log(
        "\nDeleting " +
        object_key +
        " from" +
        bucket_name +
        "/copy-destination folder"
      );
      const delete_object_from_folder_params = {
        Bucket: bucket_name,
        Key: "copy-destination/" + object_key,
      };
      await s3Client.send(
        new DeleteObjectCommand(delete_object_from_folder_params)
      );
      console.log("Success. Object deleted from folder.");
      try {
```

```
        console.log(
            "\nDeleting the bucket named " + bucket_name + "...\\n"
        );
        const delete_bucket_params = {Bucket: bucket_name};
        await s3Client.send(
            new DeleteBucketCommand(delete_bucket_params)
        );
        console.log("Success. First bucket deleted.");
        return "Run successfully"; // For unit tests.
    } catch (err) {
        console.log("Error deleting object from folder.", err);
        process.exit(1);
    }
} catch (err) {
    console.log("Error deleting bucket.", err);
    process.exit(1);
}
} catch (err) {
    console.log("Error deleting object from bucket.", err);
    process.exit(1);
}
} catch (err) {
    console.log("Error copying object from to folder", err);
    process.exit(1);
}
} catch (err) {
    console.log("Error downloading object", err);
    process.exit(1);
}
}catch (err) {
    console.log("Error creating and upload object to bucket", err);
    process.exit(1);
}
console.log("works");
} catch (err) {
    console.log("Error creating bucket", err);
}
};

run(bucket_name, object_key, object_content);
```

- For API details, see the following topics in *AWS SDK for JavaScript API Reference*.
 - [CopyObject](#)
 - [CreateBucket](#)
 - [DeleteBucket](#)
 - [DeleteObjects](#)
 - [GetObject](#)
 - [ListObjects](#)
 - [PutObject](#)

Kotlin

SDK for Kotlin

Note

This is prerelease documentation for a feature in preview release. It is subject to change.

Tip

To learn how to set up and run this example, see [GitHub](#).

```
suspend fun main(args: Array<String>) {

    val usage = """
    Usage:
        <bucketName> <key> <objectPath> <savePath> <toBucket>

    Where:
        bucketName - The Amazon S3 bucket to create.
        key - The key to use.
        objectPath - The path where the file is located (for example, C:/AWS/
book2.pdf).
        savePath - The path where the file is saved after it's downloaded (for
example, C:/AWS/book2.pdf).
        toBucket - An Amazon S3 bucket to where an object is copied to (for
example, C:/AWS/book2.pdf).
        """

    if (args.size != 4) {
        println(usage)
        exitProcess(1)
    }

    val bucketName = args[0]
    val key = args[1]
    val objectPath = args[2]
    val savePath = args[3]
    val toBucket = args[4]

    // Create an Amazon S3 bucket.
    createBucket(bucketName)

    // Update a local file to the Amazon S3 bucket.
    putObject(bucketName, key, objectPath)

    // Download the object to another local file.
    getObject(bucketName, key, savePath)

    // List all objects located in the Amazon S3 bucket.
    listBucketObs(bucketName)

    // Copy the object to another Amazon S3 bucket
    copyBucketOb(bucketName, key, toBucket)

    // Delete the object from the Amazon S3 bucket.
    deleteBucketObs(bucketName, key)

    // Delete the Amazon S3 bucket.
    deleteBucket(bucketName)
    println("All Amazon S3 operations were successfully performed")
}

suspend fun createBucket(bucketName: String) {

    val request = CreateBucketRequest {
        bucket = bucketName
    }

    S3Client { region = "us-east-1" }.use { s3 ->
        s3.createBucket(request)
        println("$bucketName is ready")
    }
}
```

```

    }

    suspend fun putObject(bucketName: String, objectKey: String, objectPath: String) {

        val metadataVal = mutableMapOf<String, String>()
        metadataVal["myVal"] = "test"

        val request = PutObjectRequest {
            bucket = bucketName
            key = objectKey
            metadata = metadataVal
            this.body = Paths.get(objectPath).asByteStream()
        }

        S3Client { region = "us-east-1" }.use { s3 ->
            val response = s3.putObject(request)
            println("Tag information is ${response.eTag}")
        }
    }

    suspend fun getObject(bucketName: String, keyName: String, path: String) {

        val request = GetObjectRequest {
            key = keyName
            bucket = bucketName
        }

        S3Client { region = "us-east-1" }.use { s3 ->
            s3.getObject(request) { resp ->
                val myFile = File(path)
                resp.body?.writeToFile(myFile)
                println("Successfully read $keyName from $bucketName")
            }
        }
    }

    suspend fun listBucketObs(bucketName: String) {

        val request = ListObjectsRequest {
            bucket = bucketName
        }

        S3Client { region = "us-east-1" }.use { s3 ->

            val response = s3.listObjects(request)
            response.contents?.forEach { myObject ->
                println("The name of the key is ${myObject.key}")
                println("The owner is ${myObject.owner}")
            }
        }
    }

    suspend fun copyBucketOb(fromBucket: String, objectKey: String, toBucket: String) {

        var encodedUrl = ""
        try {
            encodedUrl = URLEncoder.encode("$fromBucket/$objectKey",
                StandardCharsets.UTF_8.toString())
        } catch (e: UnsupportedEncodingException) {
            println("URL could not be encoded: " + e.message)
        }

        val request = CopyObjectRequest {
            copySource = encodedUrl
            bucket = toBucket
            key = objectKey
        }
    }
}

```

```
        }
        S3Client { region = "us-east-1" }.use { s3 ->
            s3.copyObject(request)
        }
    }

suspend fun deleteBucketObs(bucketName: String, objectName: String) {

    val objectId = ObjectIdentifier {
        key = objectName
    }

    val delOb = Delete {
        objects = listOf(objectId)
    }

    val request = DeleteObjectsRequest {
        bucket = bucketName
        delete = delOb
    }

    S3Client { region = "us-east-1" }.use { s3 ->
        s3.deleteObjects(request)
        println("$objectName was deleted from $bucketName")
    }
}

suspend fun deleteBucket(bucketName: String?) {

    val request = DeleteBucketRequest {
        bucket = bucketName
    }
    S3Client { region = "us-east-1" }.use { s3 ->
        s3.deleteBucket(request)
        println("The $bucketName was successfully deleted!")
    }
}
```

- For API details, see the following topics in *AWS SDK for Kotlin API reference*.
 - [CopyObject](#)
 - [CreateBucket](#)
 - [DeleteBucket](#)
 - [DeleteObjects](#)
 - [GetObject](#)
 - [ListObjects](#)
 - [PutObject](#)

PHP

SDK for PHP

Tip

To learn how to set up and run this example, see [GitHub](#).

```
require 'vendor/autoload.php';

use Aws\S3\S3Client;

echo("-----\n");
```

```
print("Welcome to the Amazon S3 getting started demo using PHP!\n");
echo("-----\n");

$region = 'us-west-2';
$version = 'latest';

$s3client = new S3Client([
    'region' => $region,
    'version' => $version
]);
/* Inline declaration example
$s3client = new Aws\S3\S3Client(['region' => 'us-west-2', 'version' => 'latest']);
*/

$bucket_name = "doc-example-bucket-" . uniqid();

try {
    $s3client->createBucket([
        'Bucket' => $bucket_name,
        'CreateBucketConfiguration' => ['LocationConstraint' => $region],
    ]);
    echo "Created bucket named: $bucket_name \n";
} catch (Exception $exception) {
    echo "Failed to create bucket $bucket_name with error: " . $exception-
>getMessage();
    exit("Please fix error with bucket creation before continuing.");
}

$file_name = "local-file-" . uniqid();
try {
    $s3client->putObject([
        'Bucket' => $bucket_name,
        'Key' => $file_name,
        'SourceFile' => 'testfile.txt'
    ]);
    echo "Uploaded $file_name to $bucket_name.\n";
} catch (Exception $exception) {
    echo "Failed to upload $file_name with error: " . $exception->getMessage();
    exit("Please fix error with file upload before continuing.");
}

try {
    $file = $s3client->getObject([
        'Bucket' => $bucket_name,
        'Key' => $file_name,
    ]);
    $body = $file->get('Body');
    $body->rewind();
    echo "Downloaded the file and it begins with: {$body->read(26)}.\n";
} catch (Exception $exception) {
    echo "Failed to download $file_name from $bucket_name with error: " .
    $exception->getMessage();
    exit("Please fix error with file downloading before continuing.");
}

try {
    $folder = "copied-folder";
    $s3client->copyObject([
        'Bucket' => $bucket_name,
        'CopySource' => "$bucket_name/$file_name",
        'Key' => "$folder/$file_name-copy",
    ]);
    echo "Copied $file_name to $folder/$file_name-copy.\n";
} catch (Exception $exception) {
    echo "Failed to copy $file_name with error: " . $exception->getMessage();
    exit("Please fix error with object copying before continuing.");
}
```

```

}

try {
    $contents = $s3client->listObjects([
        'Bucket' => $bucket_name,
    ]);
    echo "The contents of your bucket are: \n";
    foreach ($contents['Contents'] as $content) {
        echo $content['Key'] . "\n";
    }
} catch (Exception $exception) {
    echo "Failed to list objects in $bucket_name with error: " . $exception-
>getMessage();
    exit("Please fix error with listing objects before continuing.");
}

try {
    $objects = [];
    foreach ($contents['Contents'] as $content) {
        $objects[] = [
            'Key' => $content['Key'],
        ];
    }
    $s3client->deleteObjects([
        'Bucket' => $bucket_name,
        'Key' => $file_name,
        'Delete' => [
            'Objects' => $objects,
        ],
    ]);
    $check = $s3client->listObjects([
        'Bucket' => $bucket_name,
    ]);
    if (count($check) <= 0) {
        throw new Exception("Bucket wasn't empty.");
    }
    echo "Deleted all objects and folders from $bucket_name.\n";
} catch (Exception $exception) {
    echo "Failed to delete $file_name from $bucket_name with error: " . $exception-
>getMessage();
    exit("Please fix error with object deletion before continuing.");
}

try {
    $s3client->deleteBucket([
        'Bucket' => $bucket_name,
    ]);
    echo "Deleted bucket $bucket_name.\n";
} catch (Exception $exception) {
    echo "Failed to delete $bucket_name with error: " . $exception->getMessage();
    exit("Please fix error with bucket deletion before continuing.");
}

echo "Successfully ran the Amazon S3 with PHP demo.\n";

```

- For API details, see the following topics in *AWS SDK for PHP API Reference*.
 - [CopyObject](#)
 - [CreateBucket](#)
 - [DeleteBucket](#)
 - [DeleteObjects](#)
 - [GetObject](#)

- [ListObjects](#)
- [PutObject](#)

Python

SDK for Python (Boto3)

Tip

To learn how to set up and run this example, see [GitHub](#).

```
import io
import os
import uuid

import boto3
from boto3.s3.transfer import S3UploadFailedError
from botocore.exceptions import ClientError


def do_scenario(s3_resource):
    print('*'*88)
    print("Welcome to the Amazon S3 getting started demo!")
    print('*'*88)

    bucket_name = f'doc-example-bucket-{uuid.uuid4()}'  

    bucket = s3_resource.Bucket(bucket_name)
    try:
        bucket.create(
            CreateBucketConfiguration={
                'LocationConstraint': s3_resource.meta.client.meta.region_name})
        print(f"Created demo bucket named {bucket.name}.")
    except ClientError as err:
        print(f" Tried and failed to create demo bucket {bucket_name}.")
        print(f"\t{err.response['Error']['Code']}:{err.response['Error']"
        ['Message']}")
        print(f"\nCan't continue the demo without a bucket!")
        return

    file_name = None
    while file_name is None:
        file_name = input("\nEnter a file you want to upload to your bucket: ")
        if not os.path.exists(file_name):
            print(f"Couldn't find file {file_name}. Are you sure it exists?")
            file_name = None

    obj = bucket.Object(os.path.basename(file_name))
    try:
        obj.upload_file(file_name)
        print(f"Uploaded file {file_name} into bucket {bucket.name} with key
        {obj.key}.")
    except S3UploadFailedError as err:
        print(f"Couldn't upload file {file_name} to {bucket.name}.")
        print(f"\t{err}")

    answer = input(f"\nDo you want to download {obj.key} into memory (y/n)? ")
    if answer.lower() == 'y':
        data = io.BytesIO()
        try:
            obj.download_fileobj(data)
            data.seek(0)
            print(f"Got your object. Here are the first 20 bytes:\n")
            print(f"\t{data.read(20)}")
        except ClientError as err:
```

```
print(f"Couldn't download {obj.key}.")  
print(f"\t{err.response['Error']['Code']}:{err.response['Error']['Message']}")  
  
answer = input(  
    f"\nDo you want to copy {obj.key} to a subfolder in your bucket (y/n)? ")  
if answer.lower() == 'y':  
    dest_obj = bucket.Object(f'demo-folder/{obj.key}')  
    try:  
        dest_obj.copy({'Bucket': bucket.name, 'Key': obj.key})  
        print(f"Copied {obj.key} to {dest_obj.key}.")  
    except ClientError as err:  
        print(f"Couldn't copy {obj.key} to {dest_obj.key}.")  
        print(f"\t{err.response['Error']['Code']}:{err.response['Error']['Message']}")  
  
    print("\nYour bucket contains the following objects:")  
    try:  
        for o in bucket.objects.all():  
            print(f"\t{o.key}")  
    except ClientError as err:  
        print(f"Couldn't list the objects in bucket {bucket.name}.")  
        print(f"\t{err.response['Error']['Code']}:{err.response['Error']['Message']}")  
  
    answer = input(  
        "\nDo you want to delete all of the objects as well as the bucket (y/n)? ")  
    if answer.lower() == 'y':  
        try:  
            bucket.objects.delete()  
            bucket.delete()  
            print(f"Emptied and deleted bucket {bucket.name}.\n")  
        except ClientError as err:  
            print(f"Couldn't empty and delete bucket {bucket.name}.")  
            print(f"\t{err.response['Error']['Code']}:{err.response['Error']['Message']}")  
  
    print("Thanks for watching!")  
    print('*'*88)  
  
if __name__ == '__main__':  
    do_scenario(boto3.resource('s3'))
```

- For API details, see the following topics in *AWS SDK for Python (Boto3) API Reference*.
 - [CopyObject](#)
 - [CreateBucket](#)
 - [DeleteBucket](#)
 - [DeleteObjects](#)
 - [GetObject](#)
 - [ListObjects](#)
 - [PutObject](#)

Ruby

SDK for Ruby

Tip

To learn how to set up and run this example, see [GitHub](#).

```
require "aws-sdk-s3"

# Wraps the getting started scenario actions.
class ScenarioGettingStarted
  attr_reader :s3_resource

  # @param s3_resource [Aws::S3::Resource] An Amazon S3 resource.
  def initialize(s3_resource)
    @s3_resource = s3_resource
  end

  # Creates a bucket with a random name in the currently configured account and
  # AWS Region.
  #
  # @return [Aws::S3::Bucket] The newly created bucket.
  def create_bucket
    bucket = @s3_resource.create_bucket(
      bucket: "doc-example-bucket-#{Random.uuid}",
      create_bucket_configuration: {
        location_constraint: s3_resource.client.config.region
      }
    )
    puts("Created demo bucket named #{bucket.name}.")
  rescue Aws::Errors::ServiceError => e
    puts("Tried and failed to create demo bucket.")
    puts("\t#{e.code}: #{e.message}")
    puts("\nCan't continue the demo without a bucket!")
    raise
  else
    bucket
  end

  # Requests a file name from the user.
  #
  # @return The name of the file.
  def request_file_name
    file_name = ""
    while file_name.empty?
      puts("\nEnter a file you want to upload to your bucket: ")
      file_name = gets.chomp
      unless File.file?(file_name)
        puts("Couldn't find file #{file_name}. Are you sure it exists?")
        file_name = ""
      end
    end
    file_name
  end

  # Uploads a file from your computer to an Amazon S3 bucket.
  #
  # @param bucket [Aws::S3::Bucket] The bucket where the file is uploaded.
  # @return [Aws::S3::Object] The Amazon S3 object that contains the uploaded file.
  def upload_file(bucket)
    file_name = request_file_name
    s3_object = bucket.object(File.basename(file_name))
    s3_object.upload_file(file_name)
    puts("Uploaded file #{file_name} into bucket #{bucket.name} with key
#{s3_object.key}.")
  rescue Aws::Errors::ServiceError => e
    puts("Couldn't upload file #{file_name} to #{bucket.name}.")
    puts("\t#{e.code}: #{e.message}")
    raise
  else
    s3_object
  end
end
```

```

# Downloads an Amazon S3 object to a file.
#
# @param s3_object [Aws::S3::Object] The object to download.
def download_file(s3_object)
  puts("\nDo you want to download #{s3_object.key} to a local file (y/n)? ")
  answer = gets.chomp.downcase
  if answer == "y"
    puts("Enter a name for the downloaded file: ")
    file_name = gets.chomp
    s3_object.download_file(file_name)
    puts("Object #{s3_object.key} successfully downloaded to #{file_name}.")
  end
rescue Aws::Errors::ServiceError => e
  puts("Couldn't download #{s3_object.key}.")
  puts("\t#{e.code}: #{e.message}")
  raise
end

# Copies an Amazon S3 object to a subfolder within the same bucket.
#
# @param source_object [Aws::S3::Object] The source object to copy.
# @return [Aws::S3::Object, nil] The destination object.
def copy_object(source_object)
  dest_object = nil
  puts("\nDo you want to copy #{source_object.key} to a subfolder in your bucket
(y/n)? ")
  answer = gets.chomp.downcase
  if answer == "y"
    dest_object = source_object.bucket.object("demo-folder/#{source_object.key}")
    dest_object.copy_from(source_object)
    puts("Copied #{source_object.key} to #{dest_object.key}.")
  end
rescue Aws::Errors::ServiceError => e
  puts("Couldn't copy #{source_object.key}.")
  puts("\t#{e.code}: #{e.message}")
  raise
else
  dest_object
end

# Lists the objects in an Amazon S3 bucket.
#
# @param bucket [Aws::S3::Bucket] The bucket to query.
def list_objects(bucket)
  puts("\nYour bucket contains the following objects:")
  bucket.objects.each do |obj|
    puts("\t#{obj.key}")
  end
rescue Aws::Errors::ServiceError => e
  puts("Couldn't list the objects in bucket #{bucket.name}.")
  puts("\t#{e.code}: #{e.message}")
  raise
end

# Deletes the objects in an Amazon S3 bucket and deletes the bucket.
#
# @param bucket [Aws::S3::Bucket] The bucket to empty and delete.
def delete_bucket(bucket)
  puts("\nDo you want to delete all of the objects as well as the bucket (y/n)?
")
  answer = gets.chomp.downcase
  if answer == "y"
    bucket.objects.batch_delete!
    bucket.delete
    puts("Emptied and deleted bucket #{bucket.name}.\n")
  end
end

```

```
    end
  rescue Aws::Errors::ServiceError => e
    puts("Couldn't empty and delete bucket #{bucket.name}.")
    puts("\t#{e.code}: #{e.message}")
    raise
  end
end

# Runs the Amazon S3 getting started scenario.
def run_scenario(scenario)
  puts("-" * 88)
  puts("Welcome to the Amazon S3 getting started demo!")
  puts("-" * 88)

  bucket = scenario.create_bucket
  s3_object = scenario.upload_file(bucket)
  scenario.download_file(s3_object)
  scenario.copy_object(s3_object)
  scenario.list_objects(bucket)
  scenario.delete_bucket(bucket)

  puts("Thanks for watching!")
  puts("-" * 88)
rescue Aws::Errors::ServiceError
  puts("Something went wrong with the demo!")
end

run_scenario(ScenarioGettingStarted.new(Aws::S3::Resource.new)) if $PROGRAM_NAME ==
__FILE__
```

- For API details, see the following topics in *AWS SDK for Ruby API Reference*.
 - [CopyObject](#)
 - [CreateBucket](#)
 - [DeleteBucket](#)
 - [DeleteObjects](#)
 - [GetObject](#)
 - [ListObjects](#)
 - [PutObject](#)

Rust

SDK for Rust

Note

This documentation is for an SDK in preview release. The SDK is subject to change and should not be used in production.

Tip

To learn how to set up and run this example, see [GitHub](#).

Code for the binary crate which runs the scenario.

```
use aws_config::meta::region::RegionProviderChain;
use aws_sdk_s3::{Client, Error, Region};
use uuid::Uuid;

#[tokio::main]
```

```

async fn main() -> Result<(), Error> {
    let (region, client, bucket_name, file_name, key, target_key) =
        initialize_variables().await;

    if let Err(e) = run_s3_operations(region, client, bucket_name, file_name, key,
        target_key).await
    {
        println!("{}: {}", e);
    }

    Ok(())
}

async fn initialize_variables() -> (Region, Client, String, String, String, String)
{
    let region_provider = RegionProviderChain::first_try(Region::new("us-west-2"));
    let region = region_provider.region().await.unwrap();

    let shared_config =
        aws_config::from_env().region(region_provider).load().await;
    let client = Client::new(&shared_config);

    let bucket_name = format!("{}{}", "doc-example-bucket-", Uuid::new_v4());
    let file_name = "s3/testfile.txt".to_string();
    let key = "test file key name".to_string();
    let target_key = "target_key".to_string();

    (region, client, bucket_name, file_name, key, target_key)
}

async fn run_s3_operations(
    region: Region,
    client: Client,
    bucket_name: String,
    file_name: String,
    key: String,
    target_key: String,
) -> Result<(), Error> {
    s3_service::create_bucket(&client, &bucket_name, region.as_ref()).await?;
    s3_service::upload_object(&client, &bucket_name, &file_name, &key).await?;
    s3_service::download_object(&client, &bucket_name, &key).await?;
    s3_service::copy_object(&client, &bucket_name, &key, &target_key).await?;
    s3_service::list_objects(&client, &bucket_name).await?;
    s3_service::delete_objects(&client, &bucket_name).await?;
    s3_service::delete_bucket(&client, &bucket_name).await?;

    Ok(())
}

```

A library crate with common actions called by the binary.

```

use aws_sdk_s3::model::*;
use aws_sdk_s3::output::ListObjectsV2Output;
use aws_sdk_s3::types::ByteStream;
use aws_sdk_s3::{Client, Error};
use std::path::Path;
use std::str;

pub async fn delete_bucket(client: &Client, bucket_name: &str) -> Result<(), Error>
{

```

```

        client.delete_bucket().bucket(bucket_name).send().await?;
        println!("bucket deleted");
        Ok(())
    }

    pub async fn delete_objects(client: &Client, bucket_name: &str) -> Result<(), Error> {
        let objects = client.list_objects_v2().bucket(bucket_name).send().await?;

        let mut delete_objects: Vec<ObjectIdentifier> = vec![];
        for obj in objects.contents().unwrap_or_default() {
            let obj_id = ObjectIdentifier::builder()
                .set_key(Some(obj.key().unwrap().to_string()))
                .build();
            delete_objects.push(obj_id);
        }
        client
            .delete_objects()
            .bucket(bucket_name)
            .delete(Delete::builder().set_objects(Some(delete_objects)).build())
            .send()
            .await?;

        let objects: ListObjectsV2Output =
            client.list_objects_v2().bucket(bucket_name).send().await?;
        match objects.key_count {
            0 => Ok(()),
            _ => Err(Error::Unhandled(Box::from(
                "There were still objects left in the bucket.",
            ))),
        }
    }

    pub async fn list_objects(client: &Client, bucket_name: &str) -> Result<(), Error> {
        let objects = client.list_objects_v2().bucket(bucket_name).send().await?;
        println!("Objects in bucket:");
        for obj in objects.contents().unwrap_or_default() {
            println!("{}:{}", obj.key().unwrap());
        }
        Ok(())
    }

    pub async fn copy_object(
        client: &Client,
        bucket_name: &str,
        object_key: &str,
        target_key: &str,
    ) -> Result<(), Error> {
        let mut source_bucket_and_object: String = "".to_owned();
        source_bucket_and_object.push_str(bucket_name);
        source_bucket_and_object.push('/');
        source_bucket_and_object.push_str(object_key);

        client
            .copy_object()
            .copy_source(source_bucket_and_object)
            .bucket(bucket_name)
            .key(target_key)
            .send()
            .await?;

        Ok(())
    }
}

```

```
pub async fn download_object(client: &Client, bucket_name: &str, key: &str) ->
    Result<(), Error> {
    let resp = client
        .get_object()
        .bucket(bucket_name)
        .key(key)
        .send()
        .await?;

    let data = resp.body.collect().await;
    println!(
        "Data from downloaded object: {::?}",
        data.unwrap().into_bytes().slice(0..20)
    );
}

Ok(())
}

pub async fn upload_object(
    client: &Client,
    bucket_name: &str,
    file_name: &str,
    key: &str,
) -> Result<(), Error> {
    let body = ByteStream::from_path(Path::new(file_name)).await;
    client
        .put_object()
        .bucket(bucket_name)
        .key(key)
        .body(body.unwrap())
        .send()
        .await?;

    println!("Uploaded file: {}", file_name);
    Ok(())
}

pub async fn create_bucket(client: &Client, bucket_name: &str, region: &str) ->
    Result<(), Error> {
    let constraint = BucketLocationConstraint::from(region);
    let cfg = CreateBucketConfiguration::builder()
        .location_constraint(constraint)
        .build();
    client
        .create_bucket()
        .create_bucket_configuration(cfg)
        .bucket(bucket_name)
        .send()
        .await?;
    println!("{} created", bucket_name);
    Ok(())
}
```

- For API details, see the following topics in *AWS SDK for Rust API reference*.
 - [CopyObject](#)
 - [CreateBucket](#)
 - [DeleteBucket](#)
 - [DeleteObjects](#)
 - [GetObject](#)
 - [ListObjects](#)
 - [PutObject](#)

Swift

SDK for Swift

Note

This is prerelease documentation for an SDK in preview release. It is subject to change.

Tip

To learn how to set up and run this example, see [GitHub](#).

A Swift class that handles calls to the SDK for Swift.

```
import Foundation
import AWSS3
import ClientRuntime
import AWSClientRuntime

/// A class containing all the code that interacts with the AWS SDK for Swift.
public class ServiceHandler {
    let client: S3Client

    /// Initialize and return a new ``ServiceHandler`` object, which is used to
    /// drive the AWS calls
    /// used for the example.
    ///
    /// - Returns: A new ``ServiceHandler`` object, ready to be called to
    /// execute AWS operations.
    public init() async {
        do {
            client = try await S3Client()
        } catch {
            print("ERROR: ", dump(error, name: "Initializing s3 client"))
            exit(1)
        }
    }

    /// Create a new user given the specified name.
    ///
    /// - Parameters:
    ///   - name: Name of the bucket to create.
    /// Throws an exception if an error occurs.
    public func createBucket(name: String) async throws {
        let config = S3ClientTypes.CreateBucketConfiguration(
            locationConstraint: .usEast2
        )
        let input = CreateBucketInput(
            bucket: name,
            createBucketConfiguration: config
        )
        _ = try await client.createBucket(input: input)
    }

    /// Delete a bucket.
    /// - Parameter name: Name of the bucket to delete.
    public func deleteBucket(name: String) async throws {
        let input = DeleteBucketInput(
            bucket: name
        )
        _ = try await client.deleteBucket(input: input)
    }

    /// Upload a file from local storage to the bucket.
    /// - Parameters:
    ///   - bucket: Name of the bucket to upload the file to.
    ///   - key: Name of the file to create.
    ///   - file: Path name of the file to upload.
    
```

```

        public func uploadFile(bucket: String, key: String, file: String) async throws
    {
        let urlString = URL(fileURLWithPath: file)
        let fileData = try Data(contentsOf: urlString)
        let dataStream = ByteStream.from(data: fileData)

        let input = PutObjectInput(
            body: dataStream,
            bucket: bucket,
            key: key
        )
        _ = try await client.putObject(input: input)
    }

    /// Create a file in the specified bucket with the given name. The new
    /// file's contents are uploaded from a `Data` object.
    ///
    /// - Parameters:
    ///   - bucket: Name of the bucket to create a file in.
    ///   - key: Name of the file to create.
    ///   - data: A `Data` object to write into the new file.
    public func createFile(bucket: String, key: String, withData data: Data) async
throws {
    let dataStream = ByteStream.from(data: data)

    let input = PutObjectInput(
        body: dataStream,
        bucket: bucket,
        key: key
    )
    _ = try await client.putObject(input: input)
}

/// Download the named file to the given directory on the local device.
///
/// - Parameters:
///   - bucket: Name of the bucket that contains the file to be copied.
///   - key: The name of the file to copy from the bucket.
///   - to: The path of the directory on the local device where you want to
///     download the file.
public func downloadFile(bucket: String, key: String, to: String) async throws
{
    let urlString = URL(fileURLWithPath: to).appendingPathComponent(key)

    let input = GetObjectInput(
        bucket: bucket,
        key: key
    )
    let output = try await client.getObject(input: input)

    // Get the data stream object. Return immediately if there isn't one.
    guard let body = output.body else {
        return
    }
    let data = body.toBytes().toData()
    try data.write(to: urlString)
}

/// Read the specified file from the given S3 bucket into a Swift
/// `Data` object.
///
/// - Parameters:
///   - bucket: Name of the bucket containing the file to read.
///   - key: Name of the file within the bucket to read.
///
/// - Returns: A `Data` object containing the complete file data.

```

```
public func readFile(bucket: String, key: String) async throws -> Data {
    let input = GetObjectInput(
        bucket: bucket,
        key: key
    )
    let output = try await client.getObject(input: input)

    // Get the stream and return its contents in a `Data` object. If
    // there is no stream, return an empty `Data` object instead.
    guard let body = output.body else {
        return "".data(using: .utf8)!
    }
    let data = body.toBytes().toData()
    return data
}

/// Copy a file from one bucket to another.
///
/// - Parameters:
///   - sourceBucket: Name of the bucket containing the source file.
///   - name: Name of the source file.
///   - destBucket: Name of the bucket to copy the file into.
public func copyFile(from sourceBucket: String, name: String, to destBucket: String) async throws {
    let srcUrl = ("\"(sourceBucket)/
\$(name)").addingPercentEncoding(withAllowedCharacters: .urlPathAllowed)

    let input = CopyObjectInput(
        bucket: destBucket,
        copySource: srcUrl,
        key: name
    )
    _ = try await client.copyObject(input: input)
}

/// Deletes the specified file from Amazon S3.
///
/// - Parameters:
///   - bucket: Name of the bucket containing the file to delete.
///   - key: Name of the file to delete.
///
public func deleteFile(bucket: String, key: String) async throws {
    let input = DeleteObjectInput(
        bucket: bucket,
        key: key
    )

    do {
        _ = try await client.deleteObject(input: input)
    } catch {
        throw error
    }
}

/// Returns an array of strings, each naming one file in the
/// specified bucket.
///
/// - Parameter bucket: Name of the bucket to get a file listing for.
/// - Returns: An array of `String` objects, each giving the name of
///           one file contained in the bucket.
public func listBucketFiles(bucket: String) async throws -> [String] {
    let input = ListObjectsV2Input(
        bucket: bucket
    )
    let output = try await client.listObjectsV2(input: input)
    var names: [String] = []
    for object in output.contents {
        names.append(object.key)
    }
    return names
}
```

```

        guard let objList = output.contents else {
            return []
        }

        for obj in objList {
            if let objName = obj.key {
                names.append(objName)
            }
        }

        return names
    }
}

```

A Swift command-line program to manage the SDK calls.

```

import Foundation
import ServiceHandler
import ArgumentParser

/// The command-line arguments and options available for this
/// example command.
struct ExampleCommand: ParsableCommand {
    @Argument(help: "Name of the S3 bucket to create")
    var bucketName: String

    @Argument(help: "Pathname of the file to upload to the S3 bucket")
    var uploadSource: String

    @Argument(help: "The name (key) to give the file in the S3 bucket")
    var objName: String

    @Argument(help: "S3 bucket to copy the object to")
    var destBucket: String

    @Argument(help: "Directory where you want to download the file from the S3
bucket")
    var downloadDir: String

    static var configuration = CommandConfiguration(
        commandName: "s3-basics",
        abstract: "Demonstrates a series of basic AWS S3 functions.",
        discussion: """
        Performs the following Amazon S3 commands:

        * `CreateBucket`
        * `PutObject`
        * `GetObject`
        * `CopyObject`
        * `ListObjects`
        * `DeleteObjects`
        * `DeleteBucket`
        """
    )
}

/// Called by ``main()`` to do the actual running of the AWS
/// example.
func runAsync() async throws {
    let serviceHandler = await ServiceHandler()

    // 1. Create the bucket.
    print("Creating the bucket \(bucketName)...")

    try await serviceHandler.createBucket(name: bucketName)
}

```

```

// 2. Upload a file to the bucket.
print("Uploading the file \uploadSource...")
try await serviceHandler.uploadFile(bucket: bucketName, key: objName, file: uploadSource)

// 3. Download the file.
print("Downloading the file \objName to \downloadDir...")
try await serviceHandler.downloadFile(bucket: bucketName, key: objName, to: downloadDir)

// 4. Copy the file to another bucket.
print("Copying the file to the bucket \destBucket...")
try await serviceHandler.copyFile(from: bucketName, name: objName, to: destBucket)

// 5. List the contents of the bucket.

print("Getting a list of the files in the bucket \bucketName")
let fileList = try await serviceHandler.listBucketFiles(bucket: bucketName)
let numFiles = fileList.count
if numFiles != 0 {
    print("\numFiles file\((numFiles > 1) ? "s" : "") in bucket \bucketName:")
    for name in fileList {
        print(" \name")
    }
} else {
    print("No files found in bucket \bucketName")
}

// 6. Delete the objects from the bucket.

print("Deleting the file \objName from the bucket \bucketName...")
try await serviceHandler.deleteFile(bucket: bucketName, key: objName)
print("Deleting the file \objName from the bucket \destBucket...")
try await serviceHandler.deleteFile(bucket: destBucket, key: objName)

// 7. Delete the bucket.
print("Deleting the bucket \bucketName...")
try await serviceHandler.deleteBucket(name: bucketName)

print("Done.")
}

// Main program entry point.
// @main
struct Main {
    static func main() async {
        let args = Array(CommandLine.arguments.dropFirst())

        do {
            let command = try ExampleCommand.parse(args)
            try await command.runAsync()
        } catch {
            ExampleCommand.exit(withError: error)
        }
    }
}

```

- For API details, see the following topics in *AWS SDK for Swift API reference*.

- [CopyObject](#)
- [CreateBucket](#)
- [DeleteBucket](#)
- [DeleteObjects](#)
- [GetObject](#)
- [ListObjects](#)
- [PutObject](#)

For a complete list of AWS SDK developer guides and code examples, see [Using this service with an AWS SDK \(p. 1185\)](#). This topic also includes information about getting started and details about previous SDK versions.

Manage versioned Amazon S3 objects in batches with a Lambda function using an AWS SDK

The following code example shows how to manage versioned S3 objects in batches with a Lambda function.

Python

SDK for Python (Boto3)

Shows how to manipulate Amazon Simple Storage Service (Amazon S3) versioned objects in batches by creating jobs that call AWS Lambda functions to perform processing. This example creates a version-enabled bucket, uploads the stanzas from the poem *You Are Old, Father William* by Lewis Carroll, and uses Amazon S3 batch jobs to twist the poem in various ways.

Learn how to:

- Create Lambda functions that operate on versioned objects.
- Create a manifest of objects to update.
- Create batch jobs that invoke Lambda functions to update objects.
- Delete Lambda functions.
- Empty and delete a versioned bucket.

This example is best viewed on GitHub. For complete source code and instructions on how to set up and run, see the full example on [GitHub](#).

Services used in this example

- Amazon S3

For a complete list of AWS SDK developer guides and code examples, see [Using this service with an AWS SDK \(p. 1185\)](#). This topic also includes information about getting started and details about previous SDK versions.

Use a transfer manager to upload and download files to and from Amazon S3 using an AWS SDK

The following code examples show how to use a transfer manager to upload and download files to and from S3.

For more information, see [Uploading an object using multipart upload](#).

.NET

AWS SDK for .NET

Tip

To learn how to set up and run this example, see [GitHub](#).

Call functions that transfer files to and from an S3 bucket using the Amazon S3 TransferUtility.

```
global using System.Text;
global using Amazon;
global using Amazon.S3;
global using Amazon.S3.Model;
global using Amazon.S3.Transfer;
global using TransferUtilityBasics;

// This Amazon S3 client uses the default user credentials
// defined for this computer.
IAmazonS3 client = new AmazonS3Client();
var transferUtil = new TransferUtility(client);

// Change the following values to an Amazon S3 bucket that
// exists in your AWS account.
var bucketName = "doc-example-bucket1";
var localPath =
    $"{Environment.GetFolderPath(Environment.SpecialFolder.ApplicationData)}\TransferFolder";

DisplayInstructions();

PressEnter();

// Upload a single file to an S3 bucket.
var fileToUpload = "UploadTest.docx";

Console.WriteLine($"Uploading {fileToUpload} to the S3 bucket, {bucketName}.");

var success = await TransferMethods.UploadSingleFileAsync(transferUtil, bucketName,
    fileToUpload, localPath);
if (success)
{
    Console.WriteLine($"Successfully uploaded the file, {fileToUpload} to
    {bucketName}.");
}

PressEnter();

// Upload a local directory to an S3 bucket.
var keyPrefix = "UploadFolder";
var uploadPath = $"{localPath}\UploadFolder";

Console.WriteLine($"Uploading the files in {uploadPath} to {bucketName}");
Console.WriteLine($"{uploadPath} contains the following files:");
DisplayLocalFiles(uploadPath);
Console.WriteLine();

success = await TransferMethods.UploadFullDirectoryAsync(transferUtil, bucketName,
    keyPrefix, uploadPath);
if (success)
{
    Console.WriteLine($"Successfully uploaded the files in {uploadPath} to
    {bucketName}.");
    Console.WriteLine($"{bucketName} currently contains the following files:");
}
```

```
        await DisplayBucketFiles(client, bucketName, keyPrefix);
        Console.WriteLine();
    }

    PressEnter();

    // Download a single file from an S3 bucket.
    var keyName = "FileToDelete.docx";

    Console.WriteLine($"Downloading {keyName} from {bucketName}.");

    success = await TransferMethods.DownloadSingleFileAsync(transferUtil, bucketName,
        keyName, localPath);
    if (success)
    {
        Console.WriteLine($"Successfully downloaded the file, {keyName} from
        {bucketName}.");
    }

    PressEnter();

    // Download the contents of a directory from an S3 bucket.
    var s3Path = "DownloadFolder";
    var downloadPath = $"{localPath}\\DownloadFolder";

    Console.WriteLine($"Downloading the contents of {bucketName}\\{s3Path}");
    Console.WriteLine($"{bucketName}\\{s3Path} contains the following files:");
    await DisplayBucketFiles(client, bucketName, s3Path);
    Console.WriteLine();

    success = await TransferMethods.DownloadS3DirectoryAsync(transferUtil, bucketName,
        s3Path, downloadPath);
    if (success)
    {
        Console.WriteLine($"Downloaded the files in {bucketName} to {downloadPath}.");
        Console.WriteLine($"{downloadPath} now contains the following files:");
        DisplayLocalFiles(downloadPath);
    }

    Console.WriteLine("\nThe TransferUtility Basics application has completed.");
    PressEnter();

    static void DisplayInstructions()
    {
        var sepBar = new string('-', 80);

        Console.Clear();
        Console.WriteLine(sepBar);
        Console.WriteLine(CenterText("Amazon S3 Transfer Utility Basics"));
        Console.WriteLine(sepBar);
        Console.WriteLine("This program shows how to use the Amazon S3 Transfer
Utility.");
        Console.WriteLine("It performs the following actions:");
        Console.WriteLine("\t1. Upload a single object to an S3 bucket.");
        Console.WriteLine("\t2. Upload all an entire directory from the local computer
to an\n\t S3 bucket.");
        Console.WriteLine("\t3. Download a single object from an S3 bucket.");
        Console.WriteLine("\t4. Download the objects in an S3 bucket to a local
directory.");
        Console.WriteLine($"{sepBar}");
    }

    static void PressEnter()
    {
        Console.WriteLine("Press <Enter> to continue.");
    }
}
```

```
        _ = Console.ReadLine();
        Console.WriteLine("\n");
    }

    static string CenterText(string textToCenter)
    {
        var centeredText = new StringBuilder();
        centeredText.Append(new string(' ', (int)(80 - textToCenter.Length) / 2));
        centeredText.Append(textToCenter);
        return centeredText.ToString();
    }

    static void DisplayLocalFiles(string localPath)
    {
        var fileList = Directory.GetFiles(localPath);
        if (fileList is not null)
        {
            foreach (var fileName in fileList)
            {
                Console.WriteLine(fileName);
            }
        }
    }

    static async Task DisplayBucketFiles(IAmazonS3 client, string bucketName, string s3Path)
    {
        ListObjectsV2Request request = new()
        {
            BucketName = bucketName,
            Prefix = s3Path,
            MaxKeys = 5,
        };

        var response = new ListObjectsV2Response();

        do
        {
            response = await client.ListObjectsV2Async(request);

            response.S3Objects
                .ForEach(obj => Console.WriteLine($"{{obj.Key}}"));

            // If the response is truncated, set the request ContinuationToken
            // from the NextContinuationToken property of the response.
            request.ContinuationToken = response.NextContinuationToken;
        } while (response.IsTruncated);
    }
}
```

Upload a single file.

```
public static async Task<bool> UploadSingleFileAsync(
    TransferUtility transferUtil,
    string bucketName,
    string fileName,
    string localPath)
{
    if (File.Exists($"{localPath}\\{fileName}"))
    {
        try
        {
            await transferUtil.UploadAsync(new TransferUtilityUploadRequest
```

```
        BucketName = bucketName,
        Key = fileName,
        FilePath = $"{localPath}\\\{fileName}",
    });

        return true;
}
catch (AmazonS3Exception s3Ex)
{
    Console.WriteLine($"Could not upload {fileName} from
{localPath} because:");
    Console.WriteLine(s3Ex.Message);
    return false;
}
else
{
    Console.WriteLine($"{fileName} does not exist in {localPath}");
    return false;
}
}
```

Upload an entire local directory.

```
public static async Task<bool> UploadFullDirectoryAsync(
    TransferUtility transferUtil,
    string bucketName,
    string keyPrefix,
    string localPath)
{
    if (Directory.Exists(localPath))
    {
        try
        {
            await transferUtil.UploadDirectoryAsync(new
TransferUtilityUploadDirectoryRequest
{
    BucketName = bucketName,
    KeyPrefix = keyPrefix,
    Directory = localPath,
});

            return true;
        }
        catch (AmazonS3Exception s3Ex)
        {
            Console.WriteLine($"Can't upload the contents of {localPath}
because:");
            Console.WriteLine(s3Ex?.Message);
            return false;
        }
    }
    else
    {
        Console.WriteLine($"The directory {localPath} does not exist.");
        return false;
    }
}
```

Download a single file.

```
public static async Task<bool> DownloadSingleFileAsync(
    TransferUtility transferUtil,
    string bucketName,
    string keyName,
    string localPath)
{
    await transferUtil.DownloadAsync(new TransferUtilityDownloadRequest
    {
        BucketName = bucketName,
        Key = keyName,
        FilePath = $"{localPath}\\{keyName}",
    });

    if (File.Exists($"{localPath}\\{keyName}"))
    {
        return true;
    }
    else
    {
        return false;
    }
}
```

Download contents of an S3 bucket.

```
public static async Task<bool> DownloadS3DirectoryAsync(
    TransferUtility transferUtil,
    string bucketName,
    string s3Path,
    string localPath)
{
    await transferUtil.DownloadDirectoryAsync(new
TransferUtilityDownloadDirectoryRequest
    {
        BucketName = bucketName,
        LocalDirectory = localPath,
        S3Directory = s3Path,
    });

    if (Directory.Exists(localPath))
    {
        return true;
    }
    else
    {
        return false;
    }
}
```

Python

SDK for Python (Boto3)

Tip

To learn how to set up and run this example, see [GitHub](#).

Create functions that transfer files using several of the available transfer manager settings. Use a callback class to write callback progress during file transfer.

```
import sys
import threading

import boto3
from boto3.s3.transfer import TransferConfig

MB = 1024 * 1024
s3 = boto3.resource('s3')

class TransferCallback:
    """
    Handle callbacks from the transfer manager.

    The transfer manager periodically calls the __call__ method throughout
    the upload and download process so that it can take action, such as
    displaying progress to the user and collecting data about the transfer.
    """

    def __init__(self, target_size):
        self._target_size = target_size
        self._total_transferred = 0
        self._lock = threading.Lock()
        self.thread_info = {}

    def __call__(self, bytes_transferred):
        """
        The callback method that is called by the transfer manager.

        Display progress during file transfer and collect per-thread transfer
        data. This method can be called by multiple threads, so shared instance
        data is protected by a thread lock.
        """
        thread = threading.current_thread()
        with self._lock:
            self._total_transferred += bytes_transferred
            if thread.ident not in self.thread_info.keys():
                self.thread_info[thread.ident] = bytes_transferred
            else:
                self.thread_info[thread.ident] += bytes_transferred

            target = self._target_size * MB
            sys.stdout.write(
                f"\r{self._total_transferred} of {target} transferred "
                f"({(self._total_transferred / target) * 100:.2f}%).")
            sys.stdout.flush()

    def upload_with_default_configuration(local_file_path, bucket_name,
                                          object_key, file_size_mb):
        """
        Upload a file from a local folder to an Amazon S3 bucket, using the default
        configuration.
        """
        transfer_callback = TransferCallback(file_size_mb)
        s3.Bucket(bucket_name).upload_file(
            local_file_path,
            object_key,
            Callback=transfer_callback)
        return transfer_callback.thread_info

def upload_with_chunksize_and_meta(local_file_path, bucket_name, object_key,
                                   file_size_mb, metadata=None):
```

```
"""
Upload a file from a local folder to an Amazon S3 bucket, setting a
multipart chunk size and adding metadata to the Amazon S3 object.

The multipart chunk size controls the size of the chunks of data that are
sent in the request. A smaller chunk size typically results in the transfer
manager using more threads for the upload.

The metadata is a set of key-value pairs that are stored with the object
in Amazon S3.
"""
transfer_callback = TransferCallback(file_size_mb)

config = TransferConfig(multipart_chunksize=1 * MB)
extra_args = {'Metadata': metadata} if metadata else None
s3.Bucket(bucket_name).upload_file(
    local_file_path,
    object_key,
    Config=config,
    ExtraArgs=extra_args,
    Callback=transfer_callback)
return transfer_callback.thread_info


def upload_with_high_threshold(local_file_path, bucket_name, object_key,
                               file_size_mb):
"""
Upload a file from a local folder to an Amazon S3 bucket, setting a
multipart threshold larger than the size of the file.

Setting a multipart threshold larger than the size of the file results
in the transfer manager sending the file as a standard upload instead of
a multipart upload.
"""
transfer_callback = TransferCallback(file_size_mb)
config = TransferConfig(multipart_threshold=file_size_mb * 2 * MB)
s3.Bucket(bucket_name).upload_file(
    local_file_path,
    object_key,
    Config=config,
    Callback=transfer_callback)
return transfer_callback.thread_info


def upload_with_sse(local_file_path, bucket_name, object_key,
                   file_size_mb, sse_key=None):
"""
Upload a file from a local folder to an Amazon S3 bucket, adding server-side
encryption with customer-provided encryption keys to the object.

When this kind of encryption is specified, Amazon S3 encrypts the object
at rest and allows downloads only when the expected encryption key is
provided in the download request.
"""
transfer_callback = TransferCallback(file_size_mb)
if sse_key:
    extra_args = {
        'SSECustomerAlgorithm': 'AES256',
        'SSECustomerKey': sse_key}
else:
    extra_args = None
s3.Bucket(bucket_name).upload_file(
    local_file_path,
    object_key,
    ExtraArgs=extra_args,
    Callback=transfer_callback)
```

```
        return transfer_callback.thread_info

    def download_with_default_configuration(bucket_name, object_key,
                                              download_file_path, file_size_mb):
        """
        Download a file from an Amazon S3 bucket to a local folder, using the
        default configuration.
        """
        transfer_callback = TransferCallback(file_size_mb)
        s3.Bucket(bucket_name).Object(object_key).download_file(
            download_file_path,
            Callback=transfer_callback)
        return transfer_callback.thread_info

    def download_with_single_thread(bucket_name, object_key,
                                    download_file_path, file_size_mb):
        """
        Download a file from an Amazon S3 bucket to a local folder, using a
        single thread.
        """
        transfer_callback = TransferCallback(file_size_mb)
        config = TransferConfig(use_threads=False)
        s3.Bucket(bucket_name).Object(object_key).download_file(
            download_file_path,
            Config=config,
            Callback=transfer_callback)
        return transfer_callback.thread_info

    def download_with_high_threshold(bucket_name, object_key,
                                     download_file_path, file_size_mb):
        """
        Download a file from an Amazon S3 bucket to a local folder, setting a
        multipart threshold larger than the size of the file.

        Setting a multipart threshold larger than the size of the file results
        in the transfer manager sending the file as a standard download instead
        of a multipart download.
        """
        transfer_callback = TransferCallback(file_size_mb)
        config = TransferConfig(multipart_threshold=file_size_mb * 2 * MB)
        s3.Bucket(bucket_name).Object(object_key).download_file(
            download_file_path,
            Config=config,
            Callback=transfer_callback)
        return transfer_callback.thread_info

    def download_with_sse(bucket_name, object_key, download_file_path,
                          file_size_mb, sse_key):
        """
        Download a file from an Amazon S3 bucket to a local folder, adding a
        customer-provided encryption key to the request.

        When this kind of encryption is specified, Amazon S3 encrypts the object
        at rest and allows downloads only when the expected encryption key is
        provided in the download request.
        """
        transfer_callback = TransferCallback(file_size_mb)

        if sse_key:
            extra_args = {
                'SSECustomerAlgorithm': 'AES256',
                'SSECustomerKey': sse_key}
```

```
        else:
            extra_args = None
            s3.Bucket(bucket_name).Object(object_key).download_file(
                download_file_path,
                ExtraArgs=extra_args,
                Callback=transfer_callback)
            return transfer_callback.thread_info
```

Demonstrate the transfer manager functions and report results.

```
import hashlib
import os
import platform
import shutil
import time

import boto3
from boto3.s3.transfer import TransferConfig
from botocore.exceptions import ClientError
from botocore.exceptions import ParamValidationError
from botocore.exceptions import NoCredentialsError

import file_transfer

MB = 1024 * 1024
# These configuration attributes affect both uploads and downloads.
CONFIG_ATTRS = ('multipart_threshold', 'multipart_chunksize', 'max_concurrency',
                 'use_threads')
# These configuration attributes affect only downloads.
DOWNLOAD_CONFIG_ATTRS = ('max_io_queue', 'io_chunksize', 'num_download_attempts')

class TransferDemoManager:
    """
    Manages the demonstration. Collects user input from a command line, reports
    transfer results, maintains a list of artifacts created during the
    demonstration, and cleans them up after the demonstration is completed.
    """

    def __init__(self):
        self._s3 = boto3.resource('s3')
        self._chore_list = []
        self._create_file_cmd = None
        self._size_multiplier = 0
        self.file_size_mb = 30
        self.demo_folder = None
        self.demo_bucket = None
        self._setup_platform_specific()
        self._terminal_width = shutil.get_terminal_size(fallback=(80, 80))[0]

    def collect_user_info(self):
        """
        Collect local folder and Amazon S3 bucket name from the user. These
        locations are used to store files during the demonstration.
        """
        while not self.demo_folder:
            self.demo_folder = input(
                "Which file folder do you want to use to store "
                "demonstration files? ")
            if not os.path.isdir(self.demo_folder):
                print(f"{self.demo_folder} isn't a folder!")
                self.demo_folder = None

        while not self.demo_bucket:
```

```
self.demo_bucket = input(
    "Which Amazon S3 bucket do you want to use to store "
    "demonstration files? ")
try:
    self._s3.meta.client.head_bucket(Bucket=self.demo_bucket)
except ParamValidationError as err:
    print(err)
    self.demo_bucket = None
except ClientError as err:
    print(err)
    print(
        f"Either {self.demo_bucket} doesn't exist or you don't "
        f"have access to it.")
    self.demo_bucket = None

def demo(self, question, upload_func, download_func,
         upload_args=None, download_args=None):
    """Run a demonstration.

    Ask the user if they want to run this specific demonstration.
    If they say yes, create a file on the local path, upload it
    using the specified upload function, then download it using the
    specified download function.
    """
    if download_args is None:
        download_args = {}
    if upload_args is None:
        upload_args = {}
    question = question.format(self.file_size_mb)
    answer = input(f"{question} (y/n)")
    if answer.lower() == 'y':
        local_file_path, object_key, download_file_path = \
            self._create_demo_file()

        file_transfer.TransferConfig = \
            self._config_wrapper(TransferConfig, CONFIG_ATTRS)
        self._report_transfer_params('Uploading', local_file_path,
                                      object_key, **upload_args)
        start_time = time.perf_counter()
        thread_info = upload_func(local_file_path, self.demo_bucket,
                                  object_key, self.file_size_mb,
                                  **upload_args)
        end_time = time.perf_counter()
        self._report_transfer_result(thread_info, end_time - start_time)

        file_transfer.TransferConfig = \
            self._config_wrapper(TransferConfig,
                               CONFIG_ATTRS + DOWNLOAD_CONFIG_ATTRS)
        self._report_transfer_params('Downloading', object_key,
                                      download_file_path, **download_args)
        start_time = time.perf_counter()
        thread_info = download_func(self.demo_bucket, object_key,
                                    download_file_path, self.file_size_mb,
                                    **download_args)
        end_time = time.perf_counter()
        self._report_transfer_result(thread_info, end_time - start_time)

    def last_name_set(self):
        """Get the name set used for the last demo."""
        return self._chore_list[-1]

    def cleanup(self):
        """
        Remove files from the demo folder, and uploaded objects from the
        Amazon S3 bucket.
        """

```

```
print('-' * self._terminal_width)
for local_file_path, s3_object_key, downloaded_file_path \
    in self._chore_list:
    print(f"Removing {local_file_path}")
    try:
        os.remove(local_file_path)
    except FileNotFoundError as err:
        print(err)

    print(f"Removing {downloaded_file_path}")
    try:
        os.remove(downloaded_file_path)
    except FileNotFoundError as err:
        print(err)

    if self.demo_bucket:
        print(f"Removing {self.demo_bucket}:{s3_object_key}")
        try:
            self._s3.Bucket(self.demo_bucket).Object(
                s3_object_key).delete()
        except ClientError as err:
            print(err)

def _setup_platform_specific(self):
    """Set up platform-specific command used to create a large file."""
    if platform.system() == "Windows":
        self._create_file_cmd = "fsutil file createnew {} {}"
        self._size_multiplier = MB
    elif platform.system() == "Linux" or platform.system() == "Darwin":
        self._create_file_cmd = f"dd if=/dev/urandom of={} " \
            f"bs={MB} count={{}"
        self._size_multiplier = 1
    else:
        raise EnvironmentError(
            f"Demo of platform {platform.system()} isn't supported.")

def _create_demo_file(self):
    """
    Create a file in the demo folder specified by the user. Store the local
    path, object name, and download path for later cleanup.

    Only the local file is created by this method. The Amazon S3 object and
    download file are created later during the demonstration.

    Returns:
    A tuple that contains the local file path, object name, and download
    file path.
    """
    file_name_template = "TestFile{}-{}.demo"
    local_suffix = "local"
    object_suffix = "s3object"
    download_suffix = "downloaded"
    file_tag = len(self._chore_list) + 1

    local_file_path = os.path.join(
        self.demo_folder,
        file_name_template.format(file_tag, local_suffix))

    s3_object_key = file_name_template.format(file_tag, object_suffix)

    downloaded_file_path = os.path.join(
        self.demo_folder,
        file_name_template.format(file_tag, download_suffix))

    filled_cmd = self._create_file_cmd.format(
        local_file_path,
```

```
        self.file_size_mb * self._size_multiplier)

    print(f"Creating file of size {self.file_size_mb} MB "
          f"in {self.demo_folder} by running:")
    print(f"'{':4}{filled_cmd}'")
    os.system(filled_cmd)

    chore = (local_file_path, s3_object_key, downloaded_file_path)
    self._chore_list.append(chore)
    return chore

def _report_transfer_params(self, verb, source_name, dest_name, **kwargs):
    """Report configuration and extra arguments used for a file transfer."""
    print('-' * self._terminal_width)
    print(f'{verb} {source_name} ({self.file_size_mb} MB) to {dest_name}')
    if kwargs:
        print('With extra args:')
        for arg, value in kwargs.items():
            print(f'{"":4}{arg:<20}: {value}')

@staticmethod
def ask_user(question):
    """
    Ask the user a yes or no question.

    Returns:
    True when the user answers 'y' or 'Y'; otherwise, False.
    """
    answer = input(f'{question} (y/n) ')
    return answer.lower() == 'y'

@staticmethod
def _config_wrapper(func, config_attrs):
    def wrapper(*args, **kwargs):
        config = func(*args, **kwargs)
        print('With configuration:')
        for attr in config_attrs:
            print(f'{"":4}{attr:<20}: {getattr(config, attr)}')
        return config

    return wrapper

@staticmethod
def _report_transfer_result(thread_info, elapsed):
    """Report the result of a transfer, including per-thread data."""
    print(f'\nUsed {len(thread_info)} threads.')
    for ident, byte_count in thread_info.items():
        print(f'{':4}Thread {ident} copied {byte_count} bytes.")
    print(f'Your transfer took {elapsed:.2f} seconds.')

def main():
    """
    Run the demonstration script for s3_file_transfer.
    """
    demo_manager = TransferDemoManager()
    demo_manager.collect_user_info()

    # Upload and download with default configuration. Because the file is 30 MB
    # and the default multipart_threshold is 8 MB, both upload and download are
    # multipart transfers.
    demo_manager.demo(
        "Do you want to upload and download a {} MB file "
        "using the default configuration?",
        file_transfer.upload_with_default_configuration,
        file_transfer.download_with_default_configuration)
```

```
# Upload and download with multipart_threshold set higher than the size of
# the file. This causes the transfer manager to use standard transfers
# instead of multipart transfers.
demo_manager.demo(
    "Do you want to upload and download a {} MB file "
    "as a standard (not multipart) transfer?",
    file_transfer.upload_with_high_threshold,
    file_transfer.download_with_high_threshold)

# Upload with specific chunk size and additional metadata.
# Download with a single thread.
demo_manager.demo(
    "Do you want to upload a {} MB file with a smaller chunk size and "
    "then download the same file using a single thread?",
    file_transfer.upload_with_chunksizes_and_meta,
    file_transfer.download_with_single_thread,
    upload_args={
        'metadata': {
            'upload_type': 'chunky',
            'favorite_color': 'aqua',
            'size': 'medium'}})

# Upload using server-side encryption with customer-provided
# encryption keys.
# Generate a 256-bit key from a passphrase.
sse_key = hashlib.sha256('demo_passphrase'.encode('utf-8')).digest()
demo_manager.demo(
    "Do you want to upload and download a {} MB file using "
    "server-side encryption?",
    file_transfer.upload_with_sse,
    file_transfer.download_with_sse,
    upload_args={'sse_key': sse_key},
    download_args={'sse_key': sse_key})

# Download without specifying an encryption key to show that the
# encryption key must be included to download an encrypted object.
if demo_manager.ask_user("Do you want to try to download the encrypted "
    "object without sending the required key?"):
    try:
        _, object_key, download_file_path = \
            demo_manager.last_name_set()
        file_transfer.download_with_default_configuration(
            demo_manager.demo_bucket, object_key, download_file_path,
            demo_manager.file_size_mb)
    except ClientError as err:
        print("Got expected error when trying to download an encrypted "
              "object without specifying encryption info:")
        print(f"'{err}'")

# Remove all created and downloaded files, remove all objects from
# S3 storage.
if demo_manager.ask_user(
    "Demonstration complete. Do you want to remove local files "
    "and S3 objects?"):
    demo_manager.cleanup()

if __name__ == '__main__':
    try:
        main()
    except NoCredentialsError as error:
        print(error)
        print("To run this example, you must have valid credentials in "
              "a shared credential file or set in environment variables.")
```

For a complete list of AWS SDK developer guides and code examples, see [Using this service with an AWS SDK \(p. 1185\)](#). This topic also includes information about getting started and details about previous SDK versions.

Work with Amazon S3 versioned objects using an AWS SDK

The following code example shows how to:

- Create a versioned S3 bucket.
- Get all versions of an object.
- Roll an object back to a previous version.
- Delete and restore a versioned object.
- Permanently delete all versions of an object.

Python

SDK for Python (Boto3)

Tip

To learn how to set up and run this example, see [GitHub](#).

Create functions that wrap S3 actions.

```
def create_versioned_bucket(bucket_name, prefix):
    """
    Creates an Amazon S3 bucket, enables it for versioning, and configures a
    lifecycle
    that expires noncurrent object versions after 7 days.

    Adding a lifecycle configuration to a versioned bucket is a best practice.
    It helps prevent objects in the bucket from accumulating a large number of
    noncurrent versions, which can slow down request performance.

    Usage is shown in the usage_demo_single_object function at the end of this
    module.

    :param bucket_name: The name of the bucket to create.
    :param prefix: Identifies which objects are automatically expired under the
                  configured lifecycle rules.
    :return: The newly created bucket.
    """
    try:
        bucket = s3.create_bucket(
            Bucket=bucket_name,
            CreateBucketConfiguration={
                'LocationConstraint': s3.meta.client.meta.region_name
            }
        )
        logger.info("Created bucket %s.", bucket.name)
    except ClientError as error:
        if error.response['Error']['Code'] == 'BucketAlreadyOwnedByYou':
            logger.warning("Bucket %s already exists! Using it.", bucket_name)
            bucket = s3.Bucket(bucket_name)
        else:
            logger.exception("Couldn't create bucket %s.", bucket_name)
```

```

        raise

    try:
        bucket.Versioning().enable()
        logger.info("Enabled versioning on bucket %s.", bucket.name)
    except ClientError:
        logger.exception("Couldn't enable versioning on bucket %s.", bucket.name)
        raise

    try:
        expiration = 7
        bucket.LifecycleConfiguration().put(
            LifecycleConfiguration={
                'Rules': [
                    {
                        'Status': 'Enabled',
                        'Prefix': prefix,
                        'NoncurrentVersionExpiration': {'NoncurrentDays': expiration}
                    }
                ]
            }
        )
        logger.info("Configured lifecycle to expire noncurrent versions after %s days"
days "
        "on bucket %s.", expiration, bucket.name)
    except ClientError as error:
        logger.warning("Couldn't configure lifecycle on bucket %s because %s. "
                      "Continuing anyway.", bucket.name, error)

    return bucket

def rollback_object(bucket, object_key, version_id):
    """
    Rolls back an object to an earlier version by deleting all versions that
    occurred after the specified rollback version.

    Usage is shown in the usage_demo_single_object function at the end of this
    module.

    :param bucket: The bucket that holds the object to roll back.
    :param object_key: The object to roll back.
    :param version_id: The version ID to roll back to.
    """
    # Versions must be sorted by last_modified date because delete markers are
    # at the end of the list even when they are interspersed in time.
    versions = sorted(bucket.object_versions.filter(Prefix=object_key),
                      key=attrgetter('last_modified'), reverse=True)

    logger.debug(
        "Got versions:\n%s",
        '\n'.join([f"\t{version.version_id}, last modified {version.last_modified}"
                  for version in versions]))

    if version_id in [ver.version_id for ver in versions]:
        print(f"Rolling back to version {version_id}")
        for version in versions:
            if version.version_id != version_id:
                version.delete()
                print(f"Deleted version {version.version_id}")
            else:
                break

        print(f"Active version is now {bucket.Object(object_key).version_id}")
    else:
        raise KeyError(f"{version_id} was not found in the list of versions for "
                      f"{object_key}.")

def revive_object(bucket, object_key):

```

```

"""
Revives a versioned object that was deleted by removing the object's active
delete marker.
A versioned object presents as deleted when its latest version is a delete
marker.
By removing the delete marker, we make the previous version the latest version
and the object then presents as *not* deleted.

Usage is shown in the usage_demo_single_object function at the end of this
module.

:param bucket: The bucket that contains the object.
:param object_key: The object to revive.
"""

# Get the latest version for the object.
response = s3.meta.client.list_object_versions(
    Bucket=bucket.name, Prefix=object_key, MaxKeys=1)

if 'DeleteMarkers' in response:
    latest_version = response['DeleteMarkers'][0]
    if latest_version['IsLatest']:
        logger.info("Object %s was indeed deleted on %s. Let's revive it.",
                    object_key, latest_version['LastModified'])
        obj = bucket.Object(object_key)
        obj.Version(latest_version['VersionId']).delete()
        logger.info("Revived %s, active version is now %s with body '%s'",
                    object_key, obj.version_id, obj.get()['Body'].read())
    else:
        logger.warning("Delete marker is not the latest version for %s!",
                      object_key)
elif 'Versions' in response:
    logger.warning("Got an active version for %s, nothing to do.", object_key)
else:
    logger.error("Couldn't get any version info for %s.", object_key)

def permanently_delete_object(bucket, object_key):
"""
Permanently deletes a versioned object by deleting all of its versions.

Usage is shown in the usage_demo_single_object function at the end of this
module.

:param bucket: The bucket that contains the object.
:param object_key: The object to delete.
"""

try:
    bucket.object_versions.filter(Prefix=object_key).delete()
    logger.info("Permanently deleted all versions of object %s.", object_key)
except ClientError:
    logger.exception("Couldn't delete all versions of %s.", object_key)
    raise

```

Upload the stanza of a poem to a versioned object and perform a series of actions on it.

```

def usage_demo_single_object(obj_prefix='demo-versioning/'):
"""
Demonstrates usage of versioned object functions. This demo uploads a stanza
of a poem and performs a series of revisions, deletions, and revivals on it.

:param obj_prefix: The prefix to assign to objects created by this demo.
"""

with open('father_william.txt') as file:
    stanzas = file.read().split('\n\n')

```

```

width = get_terminal_size((80, 20))[0]
print('*'*width)
print("Welcome to the usage demonstration of Amazon S3 versioning.")
print("This demonstration uploads a single stanza of a poem to an Amazon "
      "S3 bucket and then applies various revisions to it.")
print('*'*width)
print("Creating a version-enabled bucket for the demo...")
bucket = create_versioned_bucket('bucket-' + str(uuid.uuid1()), obj_prefix)

print("\nThe initial version of our stanza:")
print(stanzas[0])

# Add the first stanza and revise it a few times.
print("\nApplying some revisions to the stanza...")
obj_stanza_1 = bucket.Object(f'{obj_prefix}stanza-1')
obj_stanza_1.put(Body=bytes(stanzas[0], 'utf-8'))
obj_stanza_1.put(Body=bytes(stanzas[0].upper(), 'utf-8'))
obj_stanza_1.put(Body=bytes(stanzas[0].lower(), 'utf-8'))
obj_stanza_1.put(Body=bytes(stanzas[0][::-1], 'utf-8'))
print("The latest version of the stanza is now:",
      obj_stanza_1.get()['Body'].read().decode('utf-8'),
      sep='\n')

# Versions are returned in order, most recent first.
obj_stanza_1_versions = bucket.object_versions.filter(Prefix=obj_stanza_1.key)
print(
    "The version data of the stanza revisions:",
    *[f" {version.version_id}, last modified {version.last_modified}"
      for version in obj_stanza_1_versions],
    sep='\n'
)

# Rollback two versions.
print("\nRolling back two versions...")
rollback_object(bucket, obj_stanza_1.key, list(obj_stanza_1_versions)[2].version_id)
print("The latest version of the stanza:",
      obj_stanza_1.get()['Body'].read().decode('utf-8'),
      sep='\n')

# Delete the stanza
print("\nDeleting the stanza...")
obj_stanza_1.delete()
try:
    obj_stanza_1.get()
except ClientError as error:
    if error.response['Error']['Code'] == 'NoSuchKey':
        print("The stanza is now deleted (as expected).")
    else:
        raise

# Revive the stanza
print("\nRestoring the stanza...")
revive_object(bucket, obj_stanza_1.key)
print("The stanza is restored! The latest version is again:",
      obj_stanza_1.get()['Body'].read().decode('utf-8'),
      sep='\n')

# Permanently delete all versions of the object. This cannot be undone!
print("\nPermanently deleting all versions of the stanza...")
permanently_delete_object(bucket, obj_stanza_1.key)
obj_stanza_1_versions = bucket.object_versions.filter(Prefix=obj_stanza_1.key)
if len(list(obj_stanza_1_versions)) == 0:
    print("The stanza has been permanently deleted and now has no versions.")
else:
    print("Something went wrong. The stanza still exists!")

```

```
print(f"\nRemoving {bucket.name}...")
bucket.delete()
print(f"{bucket.name} deleted.")
print("Demo done!")
```

- For API details, see the following topics in *AWS SDK for Python (Boto3) API Reference*.
 - [CreateBucket](#)
 - [DeleteObject](#)
 - [ListObjectVersions](#)
 - [PutBucketLifecycleConfiguration](#)

For a complete list of AWS SDK developer guides and code examples, see [Using this service with an AWS SDK \(p. 1185\)](#). This topic also includes information about getting started and details about previous SDK versions.

Cross-service examples for Amazon S3 using AWS SDKs

The following sample applications use AWS SDKs to combine Amazon S3 with other AWS services. Each example includes a link to GitHub, where you can find instructions on how to set up and run the application.

Examples

- [Build an Amazon Transcribe app \(p. 1510\)](#)
- [Convert text to speech and back to text using an AWS SDK \(p. 1511\)](#)
- [Create a long-lived Amazon EMR cluster and run several steps using an AWS SDK \(p. 1512\)](#)
- [Create a short-lived Amazon EMR cluster and run a step using an AWS SDK \(p. 1512\)](#)
- [Create an Amazon Textract explorer application \(p. 1513\)](#)
- [Detect PPE in images with Amazon Rekognition using an AWS SDK \(p. 1514\)](#)
- [Detect entities in text extracted from an image using an AWS SDK \(p. 1515\)](#)
- [Detect faces in an image using an AWS SDK \(p. 1515\)](#)
- [Detect objects in images with Amazon Rekognition using an AWS SDK \(p. 1516\)](#)
- [Detect people and objects in a video with Amazon Rekognition using an AWS SDK \(p. 1518\)](#)
- [Save EXIF and other image information using an AWS SDK \(p. 1519\)](#)

Build an Amazon Transcribe app

The following code example shows how to use Amazon Transcribe to transcribe and display voice recordings in the browser.

JavaScript

SDK for JavaScript V3

Create an app that uses Amazon Transcribe to transcribe and display voice recordings in the browser. The app uses two Amazon Simple Storage Service (Amazon S3) buckets, one to host

the application code, and another to store transcriptions. The app uses an Amazon Cognito user pool to authenticate your users. Authenticated users have AWS Identity and Access Management (IAM) permissions to access the required AWS services.

For complete source code and instructions on how to set up and run, see the full example on [GitHub](#).

This example is also available in the [AWS SDK for JavaScript v3 developer guide](#).

Services used in this example

- Amazon Cognito Identity
- Amazon S3
- Amazon Transcribe

For a complete list of AWS SDK developer guides and code examples, see [Using this service with an AWS SDK \(p. 1185\)](#). This topic also includes information about getting started and details about previous SDK versions.

Convert text to speech and back to text using an AWS SDK

The following code example shows how to:

- Use Amazon Polly to synthesize a plain text (UTF-8) input file to an audio file.
- Upload the audio file to an Amazon Simple Storage Service (Amazon S3) bucket.
- Use Amazon Transcribe to convert the audio file to text.
- Display the text.

Rust

SDK for Rust

Note

This documentation is for an SDK in preview release. The SDK is subject to change and should not be used in production.

Use Amazon Polly to synthesize a plain text (UTF-8) input file to an audio file, upload the audio file to an Amazon Simple Storage Service bucket, use Amazon Transcribe to convert that audio file to text, and display the text.

For complete source code and instructions on how to set up and run, see the full example on [GitHub](#).

Services used in this example

- Amazon Polly
- Amazon S3
- Amazon Transcribe

For a complete list of AWS SDK developer guides and code examples, see [Using this service with an AWS SDK \(p. 1185\)](#). This topic also includes information about getting started and details about previous SDK versions.

Create a long-lived Amazon EMR cluster and run several steps using an AWS SDK

The following code example shows how to create a long-lived Amazon EMR cluster and run several steps.

Python

SDK for Python (Boto3)

Create a long-lived Amazon EMR cluster that uses Apache Spark to query historical Amazon review data from the [Amazon Customer Reviews Dataset](#). Run a job that gets data for top-rated products in specific categories that contain keywords in their product titles. Job results are written to an Amazon Simple Storage Service (Amazon S3) bucket.

- Create an Amazon S3 bucket and upload a job script.
- Create AWS Identity and Access Management (IAM) roles.
- Create Amazon Elastic Compute Cloud (Amazon EC2) security groups.
- Create a long-lived cluster and run several job steps.

This example is best viewed on GitHub. For complete source code and instructions on how to set up and run, see the full example on [GitHub](#).

Services used in this example

- Amazon EC2
- Amazon EMR
- IAM
- Amazon S3

For a complete list of AWS SDK developer guides and code examples, see [Using this service with an AWS SDK \(p. 1185\)](#). This topic also includes information about getting started and details about previous SDK versions.

Create a short-lived Amazon EMR cluster and run a step using an AWS SDK

The following code example shows how to create a short-lived Amazon EMR cluster that runs a step and automatically terminates after the step completes.

Python

SDK for Python (Boto3)

Create a short-lived Amazon EMR cluster that estimates the value of pi using Apache Spark to parallelize a large number of calculations. The job writes output to Amazon EMR logs and to an Amazon Simple Storage Service (Amazon S3) bucket. The cluster terminates itself after completing the job.

- Create an Amazon S3 bucket and upload a job script.
- Create AWS Identity and Access Management (IAM) roles.
- Create Amazon Elastic Compute Cloud (Amazon EC2) security groups.
- Create a short-lived cluster and run a single job step.

This example is best viewed on GitHub. For complete source code and instructions on how to set up and run, see the full example on [GitHub](#).

Services used in this example

- Amazon EC2
- Amazon EMR
- IAM
- Amazon S3

For a complete list of AWS SDK developer guides and code examples, see [Using this service with an AWS SDK \(p. 1185\)](#). This topic also includes information about getting started and details about previous SDK versions.

Create an Amazon Textract explorer application

The following code examples show how to explore Amazon Textract output through an interactive application.

JavaScript

SDK for JavaScript V3

Shows how to use the AWS SDK for JavaScript to build a React application that uses Amazon Textract to extract data from a document image and display it in an interactive web page. This example runs in a web browser and requires an authenticated Amazon Cognito identity for credentials. It uses Amazon Simple Storage Service (Amazon S3) for storage, and for notifications it polls an Amazon Simple Queue Service (Amazon SQS) queue that is subscribed to an Amazon Simple Notification Service (Amazon SNS) topic.

For complete source code and instructions on how to set up and run, see the full example on [GitHub](#).

Services used in this example

- Amazon Cognito Identity
- Amazon S3
- Amazon SNS
- Amazon SQS
- Amazon Textract

Python

SDK for Python (Boto3)

Shows how to use the AWS SDK for Python (Boto3) with Amazon Textract to detect text, form, and table elements in a document image. The input image and Amazon Textract output are shown in a Tkinter application that lets you explore the detected elements.

- Submit a document image to Amazon Textract and explore the output of detected elements.
- Submit images directly to Amazon Textract or through an Amazon Simple Storage Service (Amazon S3) bucket.
- Use asynchronous APIs to start a job that publishes a notification to an Amazon Simple Notification Service (Amazon SNS) topic when the job completes.
- Poll an Amazon Simple Queue Service (Amazon SQS) queue for a job completion message and display the results.

For complete source code and instructions on how to set up and run, see the full example on [GitHub](#).

Services used in this example

- Amazon S3
- Amazon SNS
- Amazon SQS
- Amazon Textract

For a complete list of AWS SDK developer guides and code examples, see [Using this service with an AWS SDK \(p. 1185\)](#). This topic also includes information about getting started and details about previous SDK versions.

Detect PPE in images with Amazon Rekognition using an AWS SDK

The following code examples show how to build an app that uses Amazon Rekognition to detect Personal Protective Equipment (PPE) in images.

Java

SDK for Java 2.x

Shows how to create an AWS Lambda function that detects images with Personal Protective Equipment.

For complete source code and instructions on how to set up and run, see the full example on [GitHub](#).

Services used in this example

- DynamoDB
- Amazon Rekognition
- Amazon S3
- Amazon SES

JavaScript

SDK for JavaScript V3

Shows how to use Amazon Rekognition with the AWS SDK for JavaScript to create an application to detect personal protective equipment (PPE) in images located in an Amazon Simple Storage Service (Amazon S3) bucket. The app saves the results to an Amazon DynamoDB table, and sends the admin an email notification with the results using Amazon Simple Email Service (Amazon SES).

Learn how to:

- Create an unauthenticated user using Amazon Cognito.
- Analyze images for PPE using Amazon Rekognition.
- Verify an email address for Amazon SES.
- Update a DynamoDB table with results.
- Send an email notification using Amazon SES.

For complete source code and instructions on how to set up and run, see the full example on [GitHub](#).

Services used in this example

- DynamoDB
- Amazon Rekognition
- Amazon S3
- Amazon SES

For a complete list of AWS SDK developer guides and code examples, see [Using this service with an AWS SDK \(p. 1185\)](#). This topic also includes information about getting started and details about previous SDK versions.

Detect entities in text extracted from an image using an AWS SDK

The following code example shows how to use Amazon Comprehend to detect entities in text extracted by Amazon Textract from an image that is stored in Amazon S3.

Python

SDK for Python (Boto3)

Shows how to use the AWS SDK for Python (Boto3) in a Jupyter notebook to detect entities in text that is extracted from an image. This example uses Amazon Textract to extract text from an image stored in Amazon Simple Storage Service (Amazon S3) and Amazon Comprehend to detect entities in the extracted text.

This example is a Jupyter notebook and must be run in an environment that can host notebooks. For instructions on how to run the example using Amazon SageMaker, see the directions in [TextractAndComprehendNotebook.ipynb](#).

For complete source code and instructions on how to set up and run, see the full example on [GitHub](#).

Services used in this example

- Amazon Comprehend
- Amazon S3
- Amazon Textract

For a complete list of AWS SDK developer guides and code examples, see [Using this service with an AWS SDK \(p. 1185\)](#). This topic also includes information about getting started and details about previous SDK versions.

Detect faces in an image using an AWS SDK

The following code example shows how to:

- Save an image in an Amazon Simple Storage Service Amazon S3 bucket.
- Use Amazon Rekognition (Amazon Rekognition) to detect facial details, such as age range, gender, and emotion (smiling, etc.).

- Display those details.

Rust

SDK for Rust

Note

This documentation is for an SDK in preview release. The SDK is subject to change and should not be used in production.

Save the image in an Amazon Simple Storage Service bucket with an **uploads** prefix, use Amazon Rekognition to detect facial details, such as age range, gender, and emotion (smiling, etc.), and display those details.

For complete source code and instructions on how to set up and run, see the full example on [GitHub](#).

Services used in this example

- Amazon Rekognition
- Amazon S3

For a complete list of AWS SDK developer guides and code examples, see [Using this service with an AWS SDK \(p. 1185\)](#). This topic also includes information about getting started and details about previous SDK versions.

Detect objects in images with Amazon Rekognition using an AWS SDK

The following code examples show how to build an app that uses Amazon Rekognition to detect objects by category in images.

.NET

AWS SDK for .NET

Shows how to use Amazon Rekognition .NET API to create an app that uses Amazon Rekognition to identify objects by category in images located in an Amazon Simple Storage Service (Amazon S3) bucket. The app sends the admin an email notification with the results using Amazon Simple Email Service (Amazon SES).

For complete source code and instructions on how to set up and run, see the full example on [GitHub](#).

Services used in this example

- Amazon Rekognition
- Amazon S3
- Amazon SES

Java

SDK for Java 2.x

Shows how to use Amazon Rekognition Java API to create an app that uses Amazon Rekognition to identify objects by category in images located in an Amazon Simple Storage Service (Amazon

S3) bucket. The app sends the admin an email notification with the results using Amazon Simple Email Service (Amazon SES).

For complete source code and instructions on how to set up and run, see the full example on [GitHub](#).

Services used in this example

- Amazon Rekognition
- Amazon S3
- Amazon SES

JavaScript

SDK for JavaScript V3

Shows how to use Amazon Rekognition with the AWS SDK for JavaScript to create an app that uses Amazon Rekognition to identify objects by category in images located in an Amazon Simple Storage Service (Amazon S3) bucket. The app sends the admin an email notification with the results using Amazon Simple Email Service (Amazon SES).

Learn how to:

- Create an unauthenticated user using Amazon Cognito.
- Analyze images for objects using Amazon Rekognition.
- Verify an email address for Amazon SES.
- Send an email notification using Amazon SES.

For complete source code and instructions on how to set up and run, see the full example on [GitHub](#).

Services used in this example

- Amazon Rekognition
- Amazon S3
- Amazon SES

Kotlin

SDK for Kotlin

Note

This is prerelease documentation for a feature in preview release. It is subject to change.

Shows how to use Amazon Rekognition Kotlin API to create an app that uses Amazon Rekognition to identify objects by category in images located in an Amazon Simple Storage Service (Amazon S3) bucket. The app sends the admin an email notification with the results using Amazon Simple Email Service (Amazon SES).

For complete source code and instructions on how to set up and run, see the full example on [GitHub](#).

Services used in this example

- Amazon Rekognition
- Amazon S3
- Amazon SES

Python

SDK for Python (Boto3)

Shows you how to use the AWS SDK for Python (Boto3) to create a web application that lets you do the following:

- Upload photos to an Amazon Simple Storage Service (Amazon S3) bucket.
- Use Amazon Rekognition to analyze and label the photos.
- Use Amazon Simple Email Service (Amazon SES) to send email reports of image analysis.

This example contains two main components: a webpage written in JavaScript that is built with React, and a REST service written in Python that is built with Flask-RESTful.

You can use the React webpage to:

- Display a list of images that are stored in your S3 bucket.
- Upload images from your computer to your S3 bucket.
- Display images and labels that identify items that are detected in the image.
- Get a report of all images in your S3 bucket and send an email of the report.

The webpage calls the REST service. The service sends requests to AWS to perform the following actions:

- Get and filter the list of images in your S3 bucket.
- Upload photos to your S3 bucket.
- Use Amazon Rekognition to analyze individual photos and get a list of labels that identify items that are detected in the photo.
- Analyze all photos in your S3 bucket and use Amazon SES to email a report.

For complete source code and instructions on how to set up and run, see the full example on [GitHub](#).

Services used in this example

- Amazon Rekognition
- Amazon S3
- Amazon SES

For a complete list of AWS SDK developer guides and code examples, see [Using this service with an AWS SDK \(p. 1185\)](#). This topic also includes information about getting started and details about previous SDK versions.

Detect people and objects in a video with Amazon Rekognition using an AWS SDK

The following code examples show how to detect people and objects in a video with Amazon Rekognition.

Java

SDK for Java 2.x

Shows how to use Amazon Rekognition Java API to create an app to detect faces and objects in videos located in an Amazon Simple Storage Service (Amazon S3) bucket. The app sends the admin an email notification with the results using Amazon Simple Email Service (Amazon SES).

For complete source code and instructions on how to set up and run, see the full example on [GitHub](#).

Services used in this example

- Amazon Rekognition
- Amazon S3
- Amazon SES

JavaScript

SDK for JavaScript V3

Shows how to use Amazon Rekognition with the AWS SDK for JavaScript to create an app to detect faces and objects in videos located in an Amazon Simple Storage Service (Amazon S3) bucket. The app sends the admin an email notification with the results using Amazon Simple Email Service (Amazon SES).

Learn how to:

- Create an unauthenticated user using Amazon Cognito.
- Analyze images for PPE using Amazon Rekognition.
- Verify an email address for Amazon SES.
- Send an email notification using Amazon SES.

For complete source code and instructions on how to set up and run, see the full example on [GitHub](#).

Services used in this example

- Amazon Rekognition
- Amazon S3
- Amazon SES

For a complete list of AWS SDK developer guides and code examples, see [Using this service with an AWS SDK \(p. 1185\)](#). This topic also includes information about getting started and details about previous SDK versions.

Save EXIF and other image information using an AWS SDK

The following code example shows how to:

- Get EXIF information from a a JPG, JPEG, or PNG file.
- Upload the image file to an Amazon Simple Storage Service (Amazon S3) bucket.
- Use Amazon Rekognition (Amazon Rekognition) to identify the three top attributes (labels in Amazon Rekognition) in the file.
- Add the EXIF and label information to a Amazon DynamoDB (DynamoDB) table in the Region.

Rust

SDK for Rust

Note

This documentation is for an SDK in preview release. The SDK is subject to change and should not be used in production.

Get EXIF information from a JPG, JPEG, or PNG file, upload the image file to an Amazon Simple Storage Service bucket, use Amazon Rekognition to identify the three top attributes (*labels* in Amazon Rekognition) in the file, and add the EXIF and label information to a Amazon DynamoDB table in the Region.

For complete source code and instructions on how to set up and run, see the full example on [GitHub](#).

Services used in this example

- DynamoDB
- Amazon Rekognition
- Amazon S3

For a complete list of AWS SDK developer guides and code examples, see [Using this service with an AWS SDK \(p. 1185\)](#). This topic also includes information about getting started and details about previous SDK versions.

Troubleshooting

This section describes how to troubleshoot Amazon S3 and explains how to get request IDs that you'll need when you contact AWS Support.

Topics

- [Troubleshooting Amazon S3 by Symptom \(p. 1521\)](#)
- [Getting Amazon S3 Request IDs for AWS Support \(p. 1522\)](#)
- [Related Topics \(p. 1524\)](#)

For other troubleshooting and support topics, see the following:

- [Troubleshooting CORS \(p. 583\)](#)
- [Handling REST and SOAP errors \(p. 1201\)](#)
- [AWS Support Documentation](#)

For troubleshooting information regarding third-party tools, see [Getting Amazon S3 request IDs](#) in the AWS Developer Forums.

Troubleshooting Amazon S3 by Symptom

The following topics list symptoms to help you troubleshoot some of the issues that you might encounter when working with Amazon S3.

Symptoms

- [Significant Increases in HTTP 503 Responses to Amazon S3 Requests to Buckets with Versioning Enabled \(p. 1521\)](#)
- [Unexpected Behavior When Accessing Buckets Set with CORS \(p. 1522\)](#)

Significant Increases in HTTP 503 Responses to Amazon S3 Requests to Buckets with Versioning Enabled

If you notice a significant increase in the number of HTTP 503-slow down responses received for Amazon S3 PUT or DELETE object requests to a bucket that has versioning enabled, you might have one or more objects in the bucket for which there are millions of versions. When you have objects with millions of versions, Amazon S3 automatically throttles requests to the bucket to protect the customer from an excessive amount of request traffic, which could potentially impede other requests made to the same bucket.

To determine which S3 objects have millions of versions, use the Amazon S3 Inventory tool. The inventory tool generates a report that provides a flat file list of the objects in a bucket. For more information, see [Amazon S3 Inventory \(p. 739\)](#).

The Amazon S3 team encourages customers to investigate applications that repeatedly overwrite the same S3 object, potentially creating millions of versions for that object, to determine whether the application is working as intended. If you have a use case that requires millions of versions for one or more S3 objects, contact the AWS Support team at [AWS Support](#) to discuss your use case and to help us assist you in determining the optimal solution for your use case scenario.

To help prevent this issue, consider the following best practices:

- Enable a lifecycle management "NonCurrentVersion" expiration policy and an "ExpiredObjectDeleteMarker" policy to expire the previous versions of objects and delete markers without associated data objects in the bucket.
- Keep your directory structure as flat as possible and make each directory name unique.

Unexpected Behavior When Accessing Buckets Set with CORS

If you encounter unexpected behavior when accessing buckets set with the cross-origin resource sharing (CORS) configuration, see [Troubleshooting CORS \(p. 583\)](#).

Getting Amazon S3 Request IDs for AWS Support

Whenever you need to contact AWS Support due to encountering errors or unexpected behavior in Amazon S3, you will need to get the request IDs associated with the failed action. Getting these request IDs enables AWS Support to help you resolve the problems you're experiencing. Request IDs come in pairs, are returned in every response that Amazon S3 processes (even the erroneous ones), and can be accessed through verbose logs. There are a number of common methods for getting your request IDs including, S3 access logs and CloudTrail events/data events.

After you've recovered these logs, copy and retain those two values, because you'll need them when you contact AWS Support. For information about contacting AWS Support, see [Contact Us](#).

Topics

- [Using HTTP to Obtain Request IDs \(p. 1522\)](#)
- [Using a Web Browser to Obtain Request IDs \(p. 1523\)](#)
- [Using AWS SDKs to Obtain Request IDs \(p. 1523\)](#)
- [Using the AWS CLI to Obtain Request IDs \(p. 1524\)](#)

Using HTTP to Obtain Request IDs

You can obtain your request IDs, `x-amz-request-id` and `x-amz-id-2` by logging the bits of an HTTP request before it reaches the target application. There are a variety of third-party tools that can be used to recover verbose logs for HTTP requests. Choose one you trust, and run the tool, listening on the port that your Amazon S3 traffic travels on, as you send out another Amazon S3 HTTP request.

For HTTP requests, the pair of request IDs will look like the following examples.

```
x-amz-request-id: 79104EXAMPLEB723
x-amz-id-2: IOWQ4fDEXAMPLEQM+ey7N9WgVhSnQ6JEXAMPLEZb7hSQDASK+Jd1vEXAMPLEa3Km
```

Note

HTTPS requests are encrypted and hidden in most packet captures.

Using a Web Browser to Obtain Request IDs

Most web browsers have developer tools that allow you to view request headers.

For web browser-based requests that return an error, the pair of requests IDs will look like the following examples.

```
<Error><Code>AccessDenied</Code><Message>Access Denied</Message>
<RequestId>79104EXAMPLEB723</RequestId><HostId>IOWQ4fDEXAMPLEQM
+ey7N9WgVhSnQ6JEXAMPLEZb7hSQDASK+Jd1vEXAMPLEa3Km</HostId></Error>
```

For obtaining the request ID pair from successful requests, you'll need to use the developer tools to look at the HTTP response headers. For information about developer tools for specific browsers, see [Amazon S3 Troubleshooting - How to recover your S3 request IDs](#) in the AWS Developer Forums.

Using AWS SDKs to Obtain Request IDs

The following sections include information for configuring logging using an AWS SDK. While you can enable verbose logging on every request and response, you should not enable logging in production systems since large requests/responses can cause significant slowdown in an application.

For AWS SDK requests, the pair of request IDs will look like the following examples.

```
Status Code: 403, AWS Service: Amazon S3, AWS Request ID: 79104EXAMPLEB723
AWS Error Code: AccessDenied AWS Error Message: Access Denied
S3 Extended Request ID: IOWQ4fDEXAMPLEQM+ey7N9WgVhSnQ6JEXAMPLEZb7hSQDASK+Jd1vEXAMPLEa3Km
```

Using the SDK for PHP to Obtain Request IDs

You can configure logging using PHP. For more information, see [How can I see what data is sent over the wire?](#) in the FAQ for the [AWS SDK for PHP](#).

Using the SDK for Java to Obtain Request IDs

You can enable logging for specific requests or responses, allowing you to catch and return only the relevant headers. To do this, import the `com.amazonaws.services.s3.S3ResponseMetadata` class. Afterwards, you can store the request in a variable before performing the actual request. Call `getCachedResponseMetadata(AmazonWebServiceRequest request).getRequestID()` to get the logged request or response.

Example

```
PutObjectRequest req = new PutObjectRequest(bucketName, key, createSampleFile());
s3.putObject(req);
S3ResponseMetadata md = s3.getCachedResponseMetadata(req);
System.out.println("Host ID: " + md.getHostId() + " RequestID: " + md.getRequestId());
```

Alternatively, you can use verbose logging of every Java request and response. For more information, see [Verbose Wire Logging](#) in the [Logging AWS SDK for Java Calls](#) topic in the [AWS SDK for Java Developer Guide](#).

Using the AWS SDK for .NET to Obtain Request IDs

You can configure logging in AWS SDK for .NET using the built-in `System.Diagnostics` logging tool. For more information, see the [Logging with the AWS SDK for .NET](#) AWS Developer Blog post.

Note

By default, the returned log contains only error information. The config file needs to have `AWSLogMetrics` (and optionally, `AWSResponseLogging`) added to get the request IDs.

Using the SDK for Python (Boto3) to Obtain Request IDs

With SDK for Python (Boto3), you can log specific responses, which enables you to capture only the relevant headers. The following code shows you how to log parts of the response to a file:

```
import logging
import boto3
logging.basicConfig(filename='logfile.txt', level=logging.INFO)
logger = logging.getLogger(__name__)
s3 = boto3.resource('s3')
response = s3.Bucket(bucket_name).Object(object_key).put()
logger.info("HTTPStatusCode: %s", response['ResponseMetadata']['HTTPStatusCode'])
logger.info("RequestId: %s", response['ResponseMetadata']['RequestId'])
logger.info("HostId: %s", response['ResponseMetadata']['HostId'])
logger.info("Date: %s", response['ResponseMetadata']['HTTPHeaders']['date'])
```

You can also catch exceptions and log relevant information when an exception is raised. For details, see [Discerning useful information from error responses](#) in the *Boto3 developer guide*.

Additionally, you can configure Boto3 to output verbose debugging logs using the following code:

```
import boto3
boto3.set_stream_logger('', logging.DEBUG)
```

For more information, see [set_stream_logger](#) in the *Boto3 reference*.

Using the SDK for Ruby to Obtain Request IDs

You can get your request IDs using either the SDK for Ruby - Version 1, Version 2, or Version 3.

- **Using the SDK for Ruby - Version 1**– You can enable HTTP wire logging globally with the following line of code.

```
s3 = AWS::S3.new(:logger => Logger.new($stdout), :http_wire_trace => true)
```

- **Using the SDK for Ruby - Version 2 or Version 3**– You can enable HTTP wire logging globally with the following line of code.

```
s3 = Aws::S3::Client.new(:logger => Logger.new($stdout), :http_wire_trace => true)
```

Using the AWS CLI to Obtain Request IDs

You can get your request IDs in the AWS CLI by adding `--debug` to your command.

Related Topics

For other troubleshooting and support topics, see the following:

- [Troubleshooting CORS \(p. 583\)](#)

- [Handling REST and SOAP errors \(p. 1201\)](#)
- [AWS Support Documentation](#)

For troubleshooting information regarding third-party tools, see [Getting Amazon S3 request IDs](#) in the AWS Developer Forums.

Document history

- **Current API version:** 2006-03-01

The following table describes the important changes in each release of the *Amazon Simple Storage Service API Reference* and the *Amazon S3 User Guide*. For notification about updates to this documentation, you can subscribe to an RSS feed.

update-history-change	update-history-description	update-history-date
AWS Backup for Amazon S3 (p. 1526)	AWS Backup is a fully managed, policy-based service that you can use to define a central backup policy to protect your Amazon S3 data. For more information see, Using AWS Backup for Amazon S3 .	February 18, 2022
Use S3 Batch Replication to replicate existing objects (p. 1526)	S3 Batch Replication provides you a way to replicate objects that existed before a replication configuration was in place. This is done through the use of a Batch Operations job. This differs from live replication which continuously and automatically copies new objects across Amazon S3 buckets. For more information, see Replicating existing objects with S3 Batch Replication .	February 8, 2022
Rename of S3 Glacier Flexible Retrieval (p. 1526)	The Glacier storage class has been renamed to S3 Glacier Flexible Retrieval. This change does not impact the API.	November 30, 2021
New S3 Object Ownership setting to disable ACLs (p. 1526)	Apply the bucket owner enforced setting for Object Ownership to disable ACLs for your bucket and the objects in it and take ownership of every object in your bucket, simplifying access management for data stored in Amazon S3. For more information, see Controlling ownership of objects and disabling ACLs for your bucket .	November 30, 2021
New S3 Intelligent-Tiering storage class (p. 1526)	S3 Intelligent-Tiering Archive Instant Access is an additional storage class under S3 Intelligent-Tiering. For more	November 30, 2021

	information see How S3 Intelligent-Tiering works .	
New S3 Glacier Instant Retrieval storage class (p. 1526)	You can now place objects in the S3 Glacier Instant Retrieval storage class. For more information about this storage class, see Using Amazon S3 storage classes .	November 30, 2021
AWS Backup for Amazon S3 Preview (p. 1526)	AWS Backup is a fully managed, policy-based service that you can use to define a central backup policy to protect your Amazon S3 data. For more information see, Using AWS Backup for Amazon S3 .	November 30, 2021
AWS Identity and Access Management Access Analyzer for Amazon S3 (p. 1526)	IAM Access Analyzer runs policy checks to validate your policy against IAM policy grammar and best practices. To learn more about validating policies using IAM Access Analyzer, see IAM Access Analyzer policy validation in the IAM User Guide .	November 30, 2021
New event types (p. 1526)	New event types added to Amazon S3 Event Notifications, see Amazon S3 Event Notifications .	November 29, 2021
Enable Amazon EventBridge on buckets (p. 1526)	You can enable EventBridge on Amazon S3 buckets to send events to Amazon EventBridge, see Using EventBridge .	November 29, 2021
New S3 Lifecycle filters (p. 1526)	You can create lifecycle rules based on object size or specify how many noncurrent object version to keep, see Examples of S3 Lifecycle configuration .	November 23, 2021

Publish Amazon S3 Storage Lens metrics to Amazon CloudWatch (p. 1526)	You can publish S3 Storage Lens usage and activity metrics to Amazon CloudWatch to create a unified view of your operational health in CloudWatch dashboards. You can also use CloudWatch features, like alarms and triggered actions, metric math, and anomaly detection, to monitor and take action on S3 Storage Lens metrics. In addition, the CloudWatch APIs enable applications, including third-party providers, to access your S3 Storage Lens metrics. For more information, see the Monitor S3 Storage Lens metrics in CloudWatch .	November 22, 2021
Multi-Region Access Points (p. 1526)	You can use Multi-Region Access Points to create a global endpoint that applications can use to fulfill requests from Amazon S3 buckets located in multiple AWS Regions. You can use this Multi-Region Access Point to route data to a bucket with the lowest latency. For more information about Multi-Region Access Points and how to use them, see Multi-Region Access Point in Amazon S3 .	September 2, 2021
Amazon S3 on Outposts adds direct local access for applications (p. 1526)	Run your applications outside the AWS Outposts virtual private cloud (VPC) and access your S3 on Outposts data. You can also access S3 on Outposts objects directly from your on-premises network. For more information about configuring S3 on Outposts endpoints using customer-owned IP (CoIP) addresses and accessing your objects by creating a local gateway from your on-premises network, see Accessing Amazon S3 on Outposts using VPC-only access points .	July 29, 2021

Amazon S3 access point alias (p. 1526)	When you create an access point, Amazon S3 automatically generates an alias that you can use instead of a bucket name for data access. You can use this access point alias instead of an Amazon Resource Name (ARN) for any access point data plane operation. For more information, see Using a bucket-style alias for your access point .	July 26, 2021
Amazon S3 Inventory and S3 Batch Operations support Bucket Key status (p. 1526)	Amazon S3 Inventory and Batch Operations support identifying and copying existing objects with S3 Bucket Keys. S3 Bucket Keys accelerate the reduction of server-side encryption costs for existing objects. For more information, see Amazon S3 Inventory and Batch Operations Copy object .	June 3, 2021
Amazon S3 Storage Lens metrics account snapshot (p. 1526)	The S3 Storage Lens account snapshot displays your total storage, object count, and average object size on the S3 console home (Buckets) page by summarizing metrics from your default dashboard. For more information, see S3 Storage Lens metrics account snapshot .	May 5, 2021
Increased Amazon S3 on Outposts endpoint support (p. 1526)	S3 on Outposts now supports up to 100 endpoints per Outpost. For more information, see S3 on Outposts network restrictions .	April 29, 2021
Amazon S3 on Outposts event notifications in Amazon CloudWatch Events (p. 1526)	You can use CloudWatch Events to create a rule to capture any S3 on Outposts API event and get notified via all supported CloudWatch targets. For more information, see Receiving S3 on Outposts event notifications using CloudWatch Events .	April 19, 2021

S3 Object Lambda (p. 1526)	With S3 Object Lambda, you can add your own code to Amazon S3 GET requests to modify and process data as it is returned to an application. You can use custom code to modify the data returned by standard S3 GET requests to filter rows, dynamically resize images, redact confidential data, and more. For more information, see Transforming objects .	March 18, 2021
AWS PrivateLink (p. 1526)	With AWS PrivateLink for Amazon S3, you can connect directly to S3 using an interface endpoint in your VPC instead of connecting over the internet. Interface endpoints are directly accessible from applications that are on premises or in a different AWS Region. For more information, see AWS PrivateLink for Amazon S3 .	February 2, 2021
Managing Amazon S3 on Outposts capacity with AWS CloudTrail (p. 1526)	S3 on Outposts management events are available via CloudTrail logs. For more information, see Managing S3 on Outposts capacity with CloudTrail .	December 21, 2020
Strong consistency (p. 1526)	Amazon S3 provides strong read-after-write consistency for PUTs and DELETEs of objects in your Amazon S3 bucket in all AWS Regions. In addition, read operations on Amazon S3 Select, Amazon S3 access control lists, Amazon S3 Object Tags, and object metadata (e.g., HEAD object) are strongly consistent. For more information, see Amazon S3 data consistency model .	December 1, 2020

Amazon S3 replica modification sync (p. 1526)	Amazon S3 replica modification sync keeps object metadata such as tags, ACLs, and Object Lock settings in sync between source objects and replicas. When this feature is enabled, Amazon S3 replicates metadata changes made to either the source object or the replica copies. For more information, see Replicating metadata changes with replica modification sync .	December 1, 2020
Amazon S3 Bucket Keys (p. 1526)	Amazon S3 Bucket Keys reduce the cost of Amazon S3 server-side encryption using AWS Key Management Service (SSE-KMS). This new bucket-level key for server-side encryption can reduce AWS KMS request costs by up to 99 percent by decreasing the request traffic from Amazon S3 to AWS KMS. For more information, see Reducing the cost of SSE-KMS using S3 Bucket Keys .	December 1, 2020
Amazon S3 Storage Lens (p. 1526)	Amazon S3 Storage Lens aggregates your usage and activity metrics and displays the information in the account snapshot on the Amazon S3 console home (Buckets) page, interactive dashboards, or through a metrics export that you can download in CSV or Parquet format. You can use the dashboard to visualize insights and trends, flag outliers, and receive recommendations for optimizing storage costs and applying data protection best practices. You can use S3 Storage Lens through the AWS Management Console, AWS CLI, AWS SDKs, or REST API. For more information, see Assessing your storage activity and usage with S3 Storage Lens .	November 18, 2020

Tracing S3 requests using AWS X-Ray (p. 1526)	Amazon S3 integrates with X-Ray to get one request chain integrates with X-Ray to propagate trace context and give you one request chain with upstream and downstream nodes. For more information, see Tracing requests using X-Ray .	November 16, 2020
S3 replication metrics (p. 1526)	S3 replication metrics provide detailed metrics for the replication rules in your replication configuration. For more information, see Replication metrics and Amazon S3 event notifications .	November 9, 2020
S3 Intelligent-Tiering Archive Access and Deep Archive Access (p. 1526)	S3 Intelligent-Tiering Archive Access and Deep Archive Access are additional storage tiers under S3 Intelligent-Tiering. For more information, see Storage class for automatically optimizing frequently and infrequently accessed objects .	November 9, 2020
Delete marker replication (p. 1526)	With delete marker replication you can ensure that delete markers are copied to your destination buckets for your replication rules. For more information, see Using delete marker replication .	November 9, 2020
S3 Object Ownership (p. 1526)	Object Ownership is an S3 bucket setting that you can use to control ownership of new objects that are uploaded to your buckets. For more information, see Using S3 Object Ownership .	October 2, 2020
Amazon S3 on Outposts (p. 1526)	With Amazon S3 on Outposts, you can create S3 buckets on your AWS Outposts and easily store and retrieve objects on-premises for applications that require local data access, local data processing, and data residency. You can use S3 on Outposts through the AWS Management Console, AWS CLI, AWS SDKs, or REST API. For more information, see Using Amazon S3 on Outposts .	September 30, 2020

Bucket owner condition (p. 1526)	You can use Amazon S3 bucket owner condition to ensure that the buckets you use in your S3 operations belong to the AWS accounts you expect. For more information, see Bucket owner condition .	September 11, 2020
S3 Batch Operations support for Object Lock Retention (p. 1526)	You can now use Batch Operations with S3 Object Lock to apply retention settings to many Amazon S3 objects at once. For more information, see Setting S3 Object Lock Retention Dates with S3 Batch Operations .	May 4, 2020
S3 Batch Operations support for Object Lock Legal Hold (p. 1526)	You can now use Batch Operations with S3 Object Lock to add legal hold to many Amazon S3 objects at once. For more information, see Using S3 Batch Operations for setting S3 Object Lock Legal Hold .	May 4, 2020
Job Tags for S3 Batch Operations (p. 1526)	You can add tags to your S3 Batch Operations jobs to control and label those jobs. For more information, see Tags for S3 Batch Operations Jobs .	March 16, 2020
Amazon S3 access points (p. 1526)	Amazon S3 access points simplify managing data access at scale for shared datasets in S3. Access points are named network endpoints that are attached to buckets that you can use to perform S3 object operations. For more information, see Managing Data Access with Amazon S3 access points .	December 2, 2019
Access Analyzer for Amazon S3 (p. 1526)	Access Analyzer for Amazon S3 alerts you to S3 buckets that are configured to allow access to anyone on the internet or other AWS accounts, including accounts outside of your organization. For more information, see Using Access Analyzer for Amazon S3 .	December 2, 2019

S3 Replication Time Control (S3 RTC) (p. 1526)	S3 Replication Time Control (S3 RTC) replicates most objects that you upload to Amazon S3 in seconds, and 99.99 percent of those objects within 15 minutes. For more information, see Replicating Objects Using S3 Replication Time Control (S3 RTC) .	November 20, 2019
Same-Region Replication (p. 1526)	Same-Region Replication (SRR) is used to copy objects across Amazon S3 buckets in the same AWS Region. For information about both cross-Region and same-Region replication, see Replication .	September 18, 2019
Cross-Region Replication support for S3 Object Lock (p. 1526)	Cross-Region Replication now supports Object Lock. For more information, see What Does Amazon S3 Replicate? .	May 28, 2019
S3 Batch Operations (p. 1526)	Using S3 Batch Operations you can perform large-scale Batch Operations on Amazon S3 objects. S3 Batch Operations can run a single operation on lists of objects that you specify. A single job can perform the specified operation on billions of objects containing exabytes of data. For more information, see Performing S3 Batch Operations .	April 30, 2019
Asia Pacific (Hong Kong) Region (p. 1526)	Amazon S3 is now available in the Asia Pacific (Hong Kong) Region. For more information about Amazon S3 Regions and endpoints, see Regions and Endpoints in the <i>AWS General Reference</i> .	April 24, 2019
Added a new field to the server access logs (p. 1526)	Amazon S3 added the following new field to the server access logs: Transport Layer Security (TLS) version. For more information, see Server Access Log Format .	March 28, 2019
New archive storage class (p. 1526)	Amazon S3 now offers a new archive storage class, DEEP_ARCHIVE, for storing rarely accessed objects. For more information, see Storage Classes .	March 27, 2019

Added new fields to the server access logs (p. 1526)	Amazon S3 added the following new fields to the server access logs: Host Id, Signature Version, Cipher Suite, Authentication Type, and Host Header. For more information, see Server Access Log Format .	March 5, 2019
Support for Parquet-formatted Amazon S3 Inventory files (p. 1526)	Amazon S3 now supports the Apache Parquet (Parquet) format in addition to the Apache optimized row columnar (ORC) and comma-separated values (CSV) file formats for inventory output files. For more information, see Inventory .	December 4, 2018
S3 Object Lock (p. 1526)	Amazon S3 now offers Object Lock functionality that provides Write Once Read Many protections for Amazon S3 objects. For more information, see Locking Objects .	November 26, 2018
Restore speed upgrade (p. 1526)	Using Amazon S3 restore speed upgrade you can change the speed of a restoration from the S3 Glacier Flexible Retrieval storage class to a faster speed while the restoration is in progress. For more information, see Restoring Archived Objects .	November 26, 2018
Restore event notifications (p. 1526)	Amazon S3 event notifications now support initiation and completion events when restoring objects from the S3 Glacier Flexible Retrieval storage class. For more information, see Event Notifications .	November 26, 2018

PUT directly to the S3 Glacier Flexible Retrieval storage class (p. 1526)	The Amazon S3 PUT operation now supports specifying S3 Glacier Flexible Retrieval as the storage class when creating objects. Previously, you had to transition objects to the S3 Glacier Flexible Retrieval storage class from another Amazon S3 storage class. Also, when using S3 Cross-Region Replication (CRR), you can now specify S3 Glacier Flexible Retrieval as the storage class for replicated objects. For more information about the S3 Glacier Flexible Retrieval storage class, see Storage Classes . For more information about specifying the storage class for replicated objects, Replication Configuration Overview . For more information about the direct PUT to S3 Glacier Flexible Retrieval REST API changes, see Document History: PUT directly to S3 Glacier Flexible Retrieval .	November 26, 2018
New storage class (p. 1526)	Amazon S3 now offers a new storage class named INTELLIGENT_TIERING that is designed for long-lived data with changing or unknown access patterns. For more information, see Storage Classes .	November 26, 2018
Amazon S3 Block Public Access (p. 1526)	Amazon S3 now includes the ability to block public access to buckets and objects on a per-bucket or account-wide basis. For more information, see Using Amazon S3 Block Public Access .	November 15, 2018
Filtering enhancements in Cross-Region Replication (CRR) rules (p. 1526)	In a CRR rule configuration, you can specify an object filter to choose a subset of objects to apply the rule to. Previously, you could filter only on an object key prefix. In this release, you can filter on an object key prefix, one or more object tags, or both. For more information, see CRR Setup: Replication Configuration Overview .	September 19, 2018

New Amazon S3 Select features (p. 1526)	Amazon S3 Select now supports Apache Parquet input, queries on nested JSON objects, and two new Amazon CloudWatch monitoring metrics (<code>SelectScannedBytes</code> and <code>SelectReturnedBytes</code>).	September 5, 2018
Updates now available over RSS (p. 1526)	You can now subscribe to an RSS feed to receive notifications about updates to the Amazon S3 User Guide.	June 19, 2018

Earlier updates

The following table describes the important changes in each release of the *Amazon S3 User Guide* before June 19, 2018.

Change	Description	Date
Code examples update	<p>Code examples updated:</p> <ul style="list-style-type: none"> • C#—Updated all of the examples to use the task-based asynchronous pattern. For more information, see Amazon Web Services Asynchronous APIs for .NET in the <i>AWS SDK for .NET Developer Guide</i>. Code examples are now compliant with version 3 of the AWS SDK for .NET. • Java—Updated all of the examples to use the client builder model. For more information about the client builder model, see Creating Service Clients. • PHP—Updated all of the examples to use the AWS SDK for PHP 3.0. For more information about the AWS SDK for PHP 3.0, see AWS SDK for PHP. • Ruby—Updated example code so that the examples work with the AWS SDK for Ruby version 3. 	April 30, 2018
Amazon S3 now reports S3 Glacier Flexible Retrieval and ONEZONE_IA storage classes to Amazon CloudWatch Logs storage metrics	<p>In addition to reporting actual bytes, these storage metrics include per-object overhead bytes for applicable storage classes (ONEZONE_IA, STANDARD_IA, and S3 Glacier Flexible Retrieval):</p> <ul style="list-style-type: none"> • For ONEZONE_IA and STANDARD_IA storage class objects, Amazon S3 reports objects smaller than 128 KB as 128 KB. For more information, see Using Amazon S3 storage classes (p. 688). • For S3 Glacier Flexible Retrieval storage class objects, the storage metrics report the following overheads: <ul style="list-style-type: none"> • A 32 KB per-object overhead, charged at S3 Glacier Flexible Retrieval storage class pricing • An 8 KB per-object overhead, charged at STANDARD storage class pricing <p>For more information, see Transitioning objects using Amazon S3 Lifecycle (p. 703).</p>	April 30, 2018

Change	Description	Date
	For more information about storage metrics, see Monitoring metrics with Amazon CloudWatch (p. 1002) .	
New storage class	Amazon S3 now offers a new storage class, ONEZONE_IA (IA, for infrequent access) for storing objects. For more information, see Using Amazon S3 storage classes (p. 688) .	April 4, 2018
Amazon S3 Select	Amazon S3 now supports retrieving object content based on an SQL expression. For more information, see Filtering and retrieving data using Amazon S3 Select (p. 852) .	April 4, 2018
Asia Pacific (Osaka-Local) Region	Amazon S3 is now available in the Asia Pacific (Osaka-Local) Region. For more information about Amazon S3 Regions and endpoints, see Regions and Endpoints in the <i>AWS General Reference</i> . Important You can use the Asia Pacific (Osaka-Local) Region only in conjunction with the Asia Pacific (Tokyo) Region. To request access to Asia Pacific (Osaka-Local) Region, contact your sales representative.	February 12, 2018
Amazon S3 Inventory creation timestamp	Amazon S3 Inventory now includes a timestamp of the date and start time of the creation of the Amazon S3 Inventory report. You can use the timestamp to determine changes in your Amazon S3 storage from the start time of when the inventory report was generated.	January 16, 2018
Europe (Paris) Region	Amazon S3 is now available in the Europe (Paris) Region. For more information about Amazon S3 Regions and endpoints, see Regions and Endpoints in the <i>AWS General Reference</i> .	December 18, 2017
China (Ningxia) Region	Amazon S3 is now available in the China (Ningxia) Region. For more information about Amazon S3 Regions and endpoints, see Regions and Endpoints in the <i>AWS General Reference</i> .	December 11, 2017
Querying archives with SQL	Amazon S3 now supports querying S3 Glacier data archives with SQL. For more information, see Querying archived objects (p. 677) .	November 29, 2017
Support for ORC-formatted Amazon S3 Inventory files	Amazon S3 now supports the Apache optimized row columnar (ORC) format in addition to comma-separated values (CSV) file format for inventory output files. Also, you can now query Amazon S3 inventory using standard SQL by using Amazon Athena, Amazon Redshift Spectrum, and other tools such as Presto , Apache Hive , and Apache Spark . For more information, see Amazon S3 Inventory (p. 739) .	November 17, 2017
Default encryption for S3 buckets	Amazon S3 default encryption provides a way to set the default encryption behavior for an S3 bucket. You can set default encryption on a bucket so that all objects are encrypted when they are stored in the bucket. The objects are encrypted using server-side encryption with either Amazon S3-managed keys (SSE-S3) or AWS KMS-managed keys (SSE-KMS). For more information, see Setting default server-side encryption behavior for Amazon S3 buckets (p. 132) .	November 06, 2017

Change	Description	Date
Encryption status in Amazon S3 Inventory	<p>Amazon S3 now supports including encryption status in Amazon S3 Inventory so you can see how your objects are encrypted at rest for compliance auditing or other purposes. You can also configure to encrypt Amazon S3 Inventory with server-side encryption (SSE) or SSE-KMS so that all inventory files are encrypted accordingly. For more information, see Amazon S3 Inventory (p. 739).</p>	November 06, 2017
Cross-Region Replication (CRR) enhancements	<p>Cross-Region Replication now supports the following:</p> <ul style="list-style-type: none"> In a cross-account scenario, you can add a CRR configuration to change replica ownership to the AWS account that owns the destination bucket. For more information, see Changing the replica owner (p. 812). By default, Amazon S3 does not replicate objects in your source bucket that are created using server-side encryption using keys stored in AWS KMS. In your CRR configuration, you can now direct Amazon S3 to replicate these objects. For more information, see Replicating objects created with server-side encryption (SSE) using KMS keys (p. 814). 	November 06, 2017
Europe (London) Region	<p>Amazon S3 is now available in the Europe (London) Region. For more information about Amazon S3 Regions and endpoints, see Regions and Endpoints in the <i>AWS General Reference</i>.</p>	December 13, 2016
Canada (Central) Region	<p>Amazon S3 is now available in the Canada (Central) Region. For more information about Amazon S3 Regions and endpoints, see Regions and Endpoints in the <i>AWS General Reference</i>.</p>	December 8, 2016
Object tagging	<p>Amazon S3 now supports object tagging. Object tagging enables you to categorize storage. Object key name prefixes also enable you to categorize storage, object tagging adds another dimension to it.</p> <p>There are added benefits tagging offers. These include:</p> <ul style="list-style-type: none"> Object tags enable fine-grained access control of permissions (for example, you could grant an IAM user permissions to read-only objects with specific tags). Fine-grained control in specifying lifecycle configuration. You can specify tags to select a subset of objects to which lifecycle rule applies. If you have Cross-Region Replication (CRR) configured, Amazon S3 can replicate the tags. You must grant necessary permission to the IAM role created for Amazon S3 to assume to replicate objects on your behalf. You can also customize CloudWatch metrics and CloudTrail events to display information by specific tag filters. <p>For more information, see Categorizing your storage using tags (p. 825).</p>	November 29, 2016

Change	Description	Date
Amazon S3 Lifecycle now supports tag-based filters	Amazon S3 now supports tag-based filtering in lifecycle configuration. You can now specify lifecycle rules in which you can specify a key prefix, one or more object tags, or a combination of both to select a subset of objects to which the lifecycle rule applies. For more information, see Managing your storage lifecycle (p. 701) .	November 29, 2016
CloudWatch request metrics for buckets	Amazon S3 now supports CloudWatch metrics for requests made on buckets. When you enable these metrics for a bucket, the metrics report at 1-minute intervals. You can also configure which objects in a bucket will report these request metrics. For more information, see Monitoring metrics with Amazon CloudWatch (p. 1002) .	November 29, 2016
Amazon S3 Inventory	Amazon S3 now supports storage inventory. Amazon S3 Inventory provides a flat-file output of your objects and their corresponding metadata on a daily or weekly basis for an S3 bucket or a shared prefix (that is, objects that have names that begin with a common string). For more information, see Amazon S3 Inventory (p. 739) .	November 29, 2016
Amazon S3 Analytics – Storage Class Analysis	The new Amazon S3 analytics – storage class analysis feature observes data access patterns to help you determine when to transition less frequently accessed STANDARD storage to the STANDARD_IA (IA, for infrequent access) storage class. After storage class analysis observes the infrequent access patterns of a filtered set of data over a period of time, you can use the analysis results to help you improve your lifecycle policies. This feature also includes a detailed daily analysis of your storage usage at the specified bucket, prefix, or tag level that you can export to an S3 bucket. For more information, see Amazon S3 analytics – Storage Class Analysis (p. 1048) in the <i>Amazon S3 User Guide</i> .	November 29, 2016
New Expedited and Bulk data retrievals when restoring archived objects from S3 Glacier	Amazon S3 now supports Expedited and Bulk data retrievals in addition to Standard retrievals when restoring objects archived to S3 Glacier. For more information, see Restoring an archived object (p. 673) .	November 21, 2016
CloudTrail object logging	CloudTrail supports logging Amazon S3 object level API operations such as GetObject, PutObject, and DeleteObject. You can configure your event selectors to log object level API operations. For more information, see Logging Amazon S3 API calls using AWS CloudTrail (p. 962) .	November 21, 2016
US East (Ohio) Region	Amazon S3 is now available in the US East (Ohio) Region. For more information about Amazon S3 Regions and endpoints, see Regions and Endpoints in the <i>AWS General Reference</i> .	October 17, 2016
IPv6 support for Amazon S3 Transfer Acceleration	Amazon S3 now supports Internet Protocol version 6 (IPv6) for Amazon S3 Transfer Acceleration. You can connect to Amazon S3 over IPv6 by using the new dual-stack for Transfer Acceleration endpoint. For more information, see Getting started with Amazon S3 Transfer Acceleration (p. 137) .	October 6, 2016

Change	Description	Date
IPv6 support	Amazon S3 now supports Internet Protocol version 6 (IPv6). You can access Amazon S3 over IPv6 by using dual-stack endpoints. For more information, see Making requests to Amazon S3 over IPv6 (p. 1140) .	August 11, 2016
Asia Pacific (Mumbai) Region	Amazon S3 is now available in the Asia Pacific (Mumbai) Region. For more information about Amazon S3 Regions and endpoints, see Regions and Endpoints in the <i>AWS General Reference</i> .	June 27, 2016
Amazon S3 Transfer Acceleration	Amazon S3 Transfer Acceleration enables fast, easy, and secure transfers of files over long distances between your client and an S3 bucket. Transfer Acceleration takes advantage of Amazon CloudFront globally distributed edge locations. For more information, see Configuring fast, secure file transfers using Amazon S3 Transfer Acceleration (p. 136) .	April 19, 2016
Lifecycle support to remove expired object delete markers	Lifecycle configuration <code>Expiration</code> action now allows you to direct Amazon S3 to remove expired object delete markers in a versioned bucket. For more information, see Elements to describe lifecycle actions (p. 725) .	March 16, 2016
Bucket lifecycle configuration now supports action to stop incomplete multipart uploads	<p>Bucket lifecycle configuration now supports the <code>AbortIncompleteMultipartUpload</code> action that you can use to direct Amazon S3 to stop multipart uploads that don't complete within a specified number of days after being initiated. When a multipart upload becomes eligible for a stop operation, Amazon S3 deletes any uploaded parts and stops the multipart upload.</p> <p>For conceptual information, see the following topics in the <i>Amazon S3 User Guide</i>:</p> <ul style="list-style-type: none"> • Aborting a multipart upload (p. 193) • Elements to describe lifecycle actions (p. 725) <p>The following API operations have been updated to support the new action:</p> <ul style="list-style-type: none"> • PUT Bucket lifecycle – The XML configuration now allows you to specify the <code>AbortIncompleteMultipartUpload</code> action in a lifecycle configuration rule. • List Parts and Initiate Multipart Upload – Both of these API operations now return two additional response headers (<code>x-amz-abort-date</code>, and <code>x-amz-abort-rule-id</code>) if the bucket has a lifecycle rule that specifies the <code>AbortIncompleteMultipartUpload</code> action. These headers in the response indicate when the initiated multipart upload becomes eligible for a stop operation and which lifecycle rule is applicable. 	March 16, 2016

Change	Description	Date
Asia Pacific (Seoul) Region	Amazon S3 is now available in the Asia Pacific (Seoul) Region. For more information about Amazon S3 Regions and endpoints, see Regions and Endpoints in the <i>AWS General Reference</i> .	January 6, 2016
New condition key and a Multipart Upload change	<p>IAM policies now support an Amazon S3 <code>s3:x-amz-storage-class</code> condition key. For more information, see Amazon S3 condition key examples (p. 420).</p> <p>You no longer need to be the initiator of a multipart upload to upload parts and complete the upload. For more information, see Multipart upload API and permissions (p. 170).</p>	December 14, 2015
Renamed the US Standard Region	Changed the Region name string from "US Standard" to "US East (N. Virginia)." This is only a Region name update, there is no change in the functionality.	December 11, 2015
New storage class	<p>Amazon S3 now offers a new storage class, STANDARD_IA (IA, for infrequent access) for storing objects. This storage class is optimized for long-lived and less frequently accessed data. For more information, see Using Amazon S3 storage classes (p. 688).</p> <p>Lifecycle configuration feature updates now allow you to transition objects to the STANDARD_IA storage class. For more information, see Managing your storage lifecycle (p. 701).</p> <p>Previously, the Cross-Region Replication feature used the storage class of the source object for object replicas. Now, when you configure Cross-Region Replication, you can specify a storage class for the object replica created in the destination bucket. For more information, see Replicating objects (p. 753).</p>	September 16, 2015
AWS CloudTrail integration	New AWS CloudTrail integration allows you to record Amazon S3 API activity in your S3 bucket. You can use CloudTrail to track S3 bucket creations or deletions, access control modifications, or lifecycle policy changes. For more information, see Logging Amazon S3 API calls using AWS CloudTrail (p. 962) .	September 1, 2015
Bucket limit increase	Amazon S3 now supports bucket limit increases. By default, customers can create up to 100 buckets in their AWS account. Customers who need additional buckets can increase that limit by submitting a service limit increase. For information about how to increase your bucket limit, go to AWS service quotas in the <i>AWS General Reference</i> . For more information, see Using the AWS SDKs (p. 121) and Bucket restrictions and limitations (p. 147) .	August 4, 2015

Change	Description	Date
Consistency model update	Amazon S3 now supports read-after-write consistency for new objects added to Amazon S3 in the US East (N. Virginia) Region. Prior to this update, all Regions except US East (N. Virginia) Region supported read-after-write consistency for new objects uploaded to Amazon S3. With this enhancement, Amazon S3 now supports read-after-write consistency in all Regions for new objects added to Amazon S3. Read-after-write consistency allows you to retrieve objects immediately after creation in Amazon S3. For more information, see Regions (p. 6) .	August 4, 2015
Event notifications	Amazon S3 event notifications have been updated to add notifications when objects are deleted and to add filtering on object names with prefix and suffix matching. For more information, see Amazon S3 Event Notifications (p. 1017) .	July 28, 2015
Amazon CloudWatch integration	New Amazon CloudWatch integration allows you to monitor and set alarms on your Amazon S3 usage through CloudWatch metrics for Amazon S3. Supported metrics include total bytes for standard storage, total bytes for reduced-redundancy storage, and total number of objects for a given S3 bucket. For more information, see Monitoring metrics with Amazon CloudWatch (p. 1002) .	July 28, 2015
Support for deleting and emptying non-empty buckets	Amazon S3 now supports deleting and emptying non-empty buckets. For more information, see Emptying a bucket (p. 127) .	July 16, 2015
Bucket policies for Amazon VPC endpoints	Amazon S3 has added support for bucket policies for virtual private cloud (VPC) (VPC) endpoints. You can use S3 bucket policies to control access to buckets from specific VPC endpoints, or specific VPCs. VPC endpoints are easy to configure, are highly reliable, and provide a secure connection to Amazon S3 without requiring a gateway or a NAT instance. For more information, see Controlling access from VPC endpoints with bucket policies (p. 489) .	April 29, 2015
Event notifications	Amazon S3 event notifications have been updated to support the switch to resource-based permissions for AWS Lambda functions. For more information, see Amazon S3 Event Notifications (p. 1017) .	April 9, 2015
Cross-Region Replication	Amazon S3 now supports Cross-Region Replication. Cross-Region Replication is the automatic, asynchronous copying of objects across buckets in different AWS Regions. For more information, see Replicating objects (p. 753) .	March 24, 2015
Event notifications	Amazon S3 now supports new event types and destinations in a bucket notification configuration. Prior to this release, Amazon S3 supported only the <code>s3:ReducedRedundancyLostObject</code> event type and an Amazon SNS topic as the destination. For more information about the new event types, see Amazon S3 Event Notifications (p. 1017) .	November 13, 2014

Change	Description	Date
Server-side encryption with customer-provided encryption keys	<p>Server-side encryption with AWS Key Management Service (AWS KMS)</p> <p>Amazon S3 now supports server-side encryption using AWS KMS. This feature allows you to manage the envelope key through AWS KMS, and Amazon S3 calls AWS KMS to access the envelope key within the permissions you set.</p> <p>For more information about server-side encryption with AWS KMS, see Protecting Data Using Server-Side Encryption with AWS Key Management Service.</p>	November 12, 2014
Europe (Frankfurt) Region	Amazon S3 is now available in the Europe (Frankfurt) Region.	October 23, 2014
Server-side encryption with customer-provided encryption keys	<p>Amazon S3 now supports server-side encryption using customer-provided encryption keys (SSE-C). Server-side encryption enables you to request Amazon S3 to encrypt your data at rest. When using SSE-C, Amazon S3 encrypts your objects with the custom encryption keys that you provide. Since Amazon S3 performs the encryption for you, you get the benefits of using your own encryption keys without the cost of writing or executing your own encryption code.</p> <p>For more information about SSE-C, see Server-Side Encryption (Using Customer-Provided Encryption Keys).</p>	June 12, 2014
Lifecycle support for versioning	Prior to this release, lifecycle configuration was supported only on nonversioned buckets. Now you can configure lifecycle on both nonversioned and versioning-enabled buckets. For more information, see Managing your storage lifecycle (p. 701) .	May 20, 2014
Access control topics revised	Revised Amazon S3 access control documentation. For more information, see Identity and access management in Amazon S3 (p. 394) .	April 15, 2014
Server access logging topic revised	Revised server access logging documentation. For more information, see Logging requests using server access logging (p. 978) .	November 26, 2013
.NET SDK samples updated to version 2.0	.NET SDK samples in this guide are now compliant to version 2.0.	November 26, 2013
SOAP Support Over HTTP Deprecated	SOAP support over HTTP is deprecated, but it is still available over HTTPS. New Amazon S3 features will not be supported for SOAP. We recommend that you use either the REST API or the AWS SDKs.	September 20, 2013

Change	Description	Date
IAM policy variable support	<p>The IAM access policy language now supports variables. When a policy is evaluated, any policy variables are replaced with values that are supplied by context-based information from the authenticated user's session. You can use policy variables to define general purpose policies without explicitly listing all the components of the policy. For more information about policy variables, see IAM Policy Variables Overview in the <i>IAM User Guide</i>.</p> <p>For examples of policy variables in Amazon S3, see User policy examples (p. 516).</p>	April 3, 2013
Console support for Requester Pays	<p>You can now configure your bucket for Requester Pays by using the Amazon S3 console. For more information, see Using Requester Pays buckets for storage transfers and usage (p. 144).</p>	December 31, 2012
Root domain support for website hosting	<p>Amazon S3 now supports hosting static websites at the root domain. Visitors to your website can access your site from their browser without specifying "www" in the web address (e.g., "example.com"). Many customers already host static websites on Amazon S3 that are accessible from a "www" subdomain (e.g., "www.example.com"). Previously, to support root domain access, you needed to run your own web server to proxy root domain requests from browsers to your website on Amazon S3. Running a web server to proxy requests introduces additional costs, operational burden, and another potential point of failure. Now, you can take advantage of the high availability and durability of Amazon S3 for both "www" and root domain addresses. For more information, see Hosting a static website using Amazon S3 (p. 1116).</p>	December 27, 2012
Console revision	<p>Amazon S3 console has been updated. The documentation topics that refer to the console have been revised accordingly.</p>	December 14, 2012
Support for Archiving Data to S3 Glacier	<p>Amazon S3 now supports a storage option that enables you to utilize S3 Glacier's low-cost storage service for data archival. To archive objects, you define archival rules identifying objects and a timeline when you want Amazon S3 to archive these objects to S3 Glacier. You can easily set the rules on a bucket using the Amazon S3 console or programmatically using the Amazon S3 API or AWS SDKs.</p> <p>For more information, see Managing your storage lifecycle (p. 701).</p>	November 13, 2012
Support for Website Page Redirects	<p>For a bucket that is configured as a website, Amazon S3 now supports redirecting a request for an object to another object in the same bucket or to an external URL. For more information, see (Optional) Configuring a webpage redirect (p. 1130).</p> <p>For information about hosting websites, see Hosting a static website using Amazon S3 (p. 1116).</p>	October 4, 2012

Change	Description	Date
Support for Cross-Origin Resource Sharing (CORS)	Amazon S3 now supports Cross-Origin Resource Sharing (CORS). CORS defines a way in which client web applications that are loaded in one domain can interact with or access resources in a different domain. With CORS support in Amazon S3, you can build rich client-side web applications on top of Amazon S3 and selectively allow cross-domain access to your Amazon S3 resources. For more information, see Using cross-origin resource sharing (CORS) (p. 573) .	August 31, 2012
Support for Cost Allocation Tags	Amazon S3 now supports cost allocation tagging, which allows you to label S3 buckets so you can more easily track their cost against projects or other criteria. For more information about using tagging for buckets, see Using cost allocation S3 bucket tags (p. 834) .	August 21, 2012
Support for MFA-protected API access in bucket policies	Amazon S3 now supports MFA-protected API access, a feature that can enforce AWS Multi-Factor Authentication for an extra level of security when accessing your Amazon S3 resources. It is a security feature that requires users to prove physical possession of an MFA device by providing a valid MFA code. For more information, go to AWS Multi-Factor Authentication . You can now require MFA authentication for any requests to access your Amazon S3 resources. To enforce MFA authentication, Amazon S3 now supports the <code>aws:MultiFactorAuthAge</code> key in a bucket policy. For an example bucket policy, see Adding a bucket policy to require MFA (p. 495) .	July 10, 2012
Object Expiration support	You can use Object Expiration to schedule automatic removal of data after a configured time period. You set object expiration by adding lifecycle configuration to a bucket.	27 December 2011
New Region supported	Amazon S3 now supports the South America (São Paulo) Region. For more information, see Methods for accessing a bucket (p. 125) .	December 14, 2011
Multi-Object Delete	Amazon S3 now supports Multi-Object Delete API that enables you to delete multiple objects in a single request. With this feature, you can remove large numbers of objects from Amazon S3 more quickly than using multiple individual DELETE requests. For more information, see Deleting Amazon S3 objects (p. 223) .	December 7, 2011
New Region supported	Amazon S3 now supports the US West (Oregon) Region. For more information, see Buckets and Regions (p. 125) .	November 8, 2011
Documentation Update	Documentation bug fixes.	November 8, 2011

Change	Description	Date
Documentation Update	<p>In addition to documentation bug fixes, this release includes the following enhancements:</p> <ul style="list-style-type: none"> • New server-side encryption sections using the AWS SDK for PHP and the AWS SDK for Ruby (see Specifying Amazon S3 encryption (p. 357)). • New section on creating and testing Ruby samples (see Using the AWS SDK for Ruby - Version 3 (p. 1194)). 	October 17, 2011
Server-side encryption support	<p>Amazon S3 now supports server-side encryption. It enables you to request Amazon S3 to encrypt your data at rest, that is, encrypt your object data when Amazon S3 writes your data to disks in its data centers. In addition to REST API updates, the AWS SDK for Java and .NET provide necessary functionality to request server-side encryption. You can also request server-side encryption when uploading objects using the AWS Management Console. To learn more about data encryption, go to Using Data Encryption.</p>	October 4, 2011
Documentation Update	<p>In addition to documentation bug fixes, this release includes the following enhancements:</p> <ul style="list-style-type: none"> • Added Ruby and PHP samples to the Making requests (p. 1138) section. • Added sections describing how to generate and use presigned URLs. For more information, see Sharing objects using presigned URLs (p. 258) and Sharing objects using presigned URLs (p. 258). • Updated an existing section to introduce AWS Explorers for Eclipse and Visual Studio. For more information, see Developing with Amazon S3 using the AWS SDKs, and explorers (p. 1184). 	September 22, 2011
Support for sending requests using temporary security credentials	<p>In addition to using your AWS account and IAM user security credentials to send authenticated requests to Amazon S3, you can now send requests using temporary security credentials you obtain from AWS Identity and Access Management (IAM). You can use the AWS Security Token Service API or the AWS SDK wrapper libraries to request these temporary credentials from IAM. You can request these temporary security credentials for your own use or hand them out to federated users and applications. This feature enables you to manage your users outside AWS and provide them with temporary security credentials to access your AWS resources.</p> <p>For more information, see Making requests (p. 1138).</p> <p>For more information about IAM support for temporary security credentials, see Temporary Security Credentials in the IAM User Guide.</p>	August 3, 2011

Change	Description	Date
Multipart Upload API extended to enable copying objects up to 5 TB	<p>Prior to this release, Amazon S3 API supported copying objects of up to 5 GB in size. To enable copying objects larger than 5 GB, Amazon S3 now extends the multipart upload API with a new operation, <code>Upload Part (Copy)</code>. You can use this multipart upload operation to copy objects up to 5 TB in size. For more information, see Copying objects (p. 201).</p> <p>For conceptual information about multipart upload API, see Uploading and copying objects using multipart upload (p. 167).</p>	June 21, 2011
SOAP API calls over HTTP disabled	<p>To increase security, SOAP API calls over HTTP are disabled. Authenticated and anonymous SOAP requests must be sent to Amazon S3 using SSL.</p>	June 6, 2011
IAM enables cross-account delegation	<p>Previously, to access an Amazon S3 resource, an IAM user needed permissions from both the parent AWS account and the Amazon S3 resource owner. With cross-account access, the IAM user now only needs permission from the owner account. That is, If a resource owner grants access to an AWS account, the AWS account can now grant its IAM users access to these resources.</p> <p>For more information, see Creating a Role to Delegate Permissions to an IAM User in the <i>IAM User Guide</i>.</p> <p>For more information on specifying principals in a bucket policy, see Principals (p. 414).</p>	June 6, 2011
New link	<p>This service's endpoint information is now located in the AWS General Reference. For more information, go to Regions and Endpoints in AWS General Reference.</p>	March 1, 2011
Support for hosting static websites in Amazon S3	<p>Amazon S3 introduces enhanced support for hosting static websites. This includes support for index documents and custom error documents. When using these features, requests to the root of your bucket or a subfolder (e.g., <code>http://mywebsite.com/subfolder</code>) returns your index document instead of the list of objects in your bucket. If an error is encountered, Amazon S3 returns your custom error message instead of an Amazon S3 error message. For more information, see Hosting a static website using Amazon S3 (p. 1116).</p>	February 17, 2011
Response Header API Support	<p>The GET Object REST API now allows you to change the response headers of the REST GET Object request for each request. That is, you can alter object metadata in the response, without altering the object itself. For more information, see Downloading an object (p. 209).</p>	January 14, 2011

Change	Description	Date
Large object support	Amazon S3 has increased the maximum size of an object you can store in an S3 bucket from 5 GB to 5 TB. If you are using the REST API you can upload objects of up to 5 GB size in a single PUT operation. For larger objects, you must use the Multipart Upload REST API to upload objects in parts. For more information, see Uploading and copying objects using multipart upload (p. 167) .	December 9, 2010
Multipart upload	Multipart upload enables faster, more flexible uploads into Amazon S3. It allows you to upload a single object as a set of parts. For more information, see Uploading and copying objects using multipart upload (p. 167) .	November 10, 2010
Canonical ID support in bucket policies	You can now specify canonical IDs in bucket policies. For more information, see Bucket policies and user policies (p. 411)	September 17, 2010
Amazon S3 works with IAM	This service now integrates with AWS Identity and Access Management (IAM). For more information, go to AWS Services That Work with IAM in the <i>IAM User Guide</i> .	September 2, 2010
Notifications	The Amazon S3 notifications feature enables you to configure a bucket so that Amazon S3 publishes a message to an Amazon Simple Notification Service (Amazon SNS) topic when Amazon S3 detects a key event on a bucket. For more information, see Setting Up Notification of Bucket Events (p. 1017) .	July 14, 2010
Bucket policies	Bucket policies is an access management system you use to set access permissions across buckets, objects, and sets of objects. This functionality supplements and in many cases replaces access control lists. For more information, see Bucket policies and user policies (p. 411) .	July 6, 2010
Path-style syntax available in all Regions	Amazon S3 now supports the path-style syntax for any bucket in the US Classic Region, or if the bucket is in the same Region as the endpoint of the request. For more information, see Virtual Hosting (p. 1175) .	June 9, 2010
New endpoint for Europe (Ireland)	Amazon S3 now provides an endpoint for Europe (Ireland): http://s3-eu-west-1.amazonaws.com .	June 9, 2010
Console	You can now use Amazon S3 through the AWS Management Console. You can read about all of the Amazon S3 functionality in the console in the Amazon Simple Storage Service User Guide.	June 9, 2010
Reduced Redundancy	Amazon S3 now enables you to reduce your storage costs by storing objects in Amazon S3 with reduced redundancy. For more information, see Reduced Redundancy Storage (p. 1) .	May 12, 2010
New Region supported	Amazon S3 now supports the Asia Pacific (Singapore) Region. For more information, see Buckets and Regions (p. 125) .	April 28, 2010

Change	Description	Date
Object Versioning	This release introduces object versioning. All objects now can have a key and a version. If you enable versioning for a bucket, Amazon S3 gives all objects added to a bucket a unique version ID. This feature enables you to recover from unintended overwrites and deletions. For more information, see Versioning (p. 5) and Using Versioning (p. 638) .	February 8, 2010
New Region supported	Amazon S3 now supports the US West (N. California) Region. The new endpoint for requests to this Region is <code>s3-us-west-1.amazonaws.com</code> . For more information, see Buckets and Regions (p. 125) .	December 2, 2009
AWS SDK for .NET	AWS now provides libraries, sample code, tutorials, and other resources for software developers who prefer to build applications using .NET language-specific API operations instead of REST or SOAP. These libraries provide basic functions (not included in the REST or SOAP APIs), such as request authentication, request retries, and error handling so that it's easier to get started. For more information about language-specific libraries and resources, see Developing with Amazon S3 using the AWS SDKs, and explorers (p. 1184) .	November 11, 2009

AWS glossary

For the latest AWS terminology, see the [AWS glossary](#) in the *AWS General Reference*.