aws

User Guide

AWS Elemental MediaStore

AWS Elemental MediaStore: User Guide

Table of Contents

End of support notice: On November 13, 2025, AWS will discontinue support for AWS Elemental MediaStore. After November 13, 2025, you will no longer be able to access the MediaStore console or MediaStore resources. For more information, visit this blog post.

What is AWS Elemental MediaStore?

AWS Elemental MediaStore is a video origination and storage service that offers the high performance and immediate consistency required for live origination. With MediaStore, you can manage video assets as objects in containers to build dependable, cloud-based media workflows.

To use the service, you upload your objects from a source, such as an encoder or data feed, to a container that you create in MediaStore.

MediaStore is a great choice for storing fragmented video files when you need strong consistency, low-latency reads and writes, and the ability to handle high volumes of concurrent requests. If you don't deliver live streaming videos, consider using Amazon Simple Storage Service (Amazon S3) instead.

Topics

AWS Elemental MediaStore concepts and terminology

ARN

An Amazon Resource Name.

Body

The data to be uploaded into an object.

(Byte) range

A subset of object data to be addressed. For more information, see range from the HTTP specification.

Container

A namespace that holds objects. A container has an endpoint that you can use for writing and retrieving objects and attaching access policies.

Endpoint

An entry point to the MediaStore service, given as an HTTPS root URL.

ETag

An entity tag, which is a hash of the object data.

Folder

A division of a container. A folder can hold objects and other folders.

Item

A term used to refer to objects and folders.

Object

An asset, similar to an Amazon S3 object. Objects are the fundamental entities that are stored in MediaStore. The service accepts all file types.

Origination service

MediaStore is considered an *origination service* because it is the point of distribution for media content delivery.

Path

A unique identifier for an object or folder, which indicates its location in the container.

Part

A subset of data (chunk) of an object.

Policy

An IAM policy.

Resource

An entity in AWS that you can work with. Each AWS resource is assigned an Amazon Resource Name (ARN) that acts as a unique identifier. In MediaStore, this is the resource and its ARN format:

- Container: `aws:mediastore:`*`region`*`:`*`account-id`*`:container/:`*`containerName`*

Related services

- **Amazon CloudFront** is a global content delivery network (CDN) service that securely delivers data and videos to your viewers. Use CloudFront to deliver content with the best possible performance. For more information, see the Amazon CloudFront Developer Guide.
- **AWS CloudFormation** is a service that helps you model and set up your AWS resources. You create a template that describes all the AWS resources that you want (like MediaStore containers), and AWS CloudFormation takes care of provisioning and configuring those resources for you. You don't need to individually create and configure AWS resources and figure out what's dependent on what; AWS CloudFormation handles all of that. For more information, see the AWS CloudFormation User Guide.
- **AWS CloudTrail** is a service that lets you monitor the calls made to the CloudTrail API for your account, including calls made by the AWS Management Console, AWS CLI, and other services. For more information, see the AWS CloudTrail User Guide.
- **Amazon CloudWatch** is a monitoring service for AWS Cloud resources and the applications that you run on AWS. Use CloudWatch Events to track changes in the status of containers and objects in MediaStore. For more information, see the Amazon CloudWatch documentation.
- **AWS Identity and Access Management (IAM)** is a web service that helps you securely control access to AWS resources for your users. Use IAM to control who can use your AWS resources (authentication) and what resources users can use in which ways (authorization). For more information, see *Setting Up AWS Elemental MediaStore*.
- **Amazon Simple Storage Service (Amazon S3)** is object storage built to store and retrieve any amount of data from anywhere. For more information, see the Amazon S3 documentation.

Accessing AWS Elemental MediaStore

You can access MediaStore using any of the following methods:

- **AWS Management Console** - The procedures throughout this guide explain how to use the AWS Management Console to perform tasks for MediaStore. To access MediaStore using the console:

  ```
  https://<region>.console.aws.amazon.com/mediastore/home
  ```

- **AWS Command Line Interface** – For more information, see the AWS Command Line Interface User Guide. To access MediaStore using the CLI endpoint:

```
aws mediastore
```

- **MediaStore API** – If you're using a programming language that an SDK isn't available for, see the AWS Elemental MediaStore API Reference for information about API actions and about how to make API requests. To access MediaStore using the REST API endpoint:

```
https://mediastore.<region>.amazonaws.com
```

- **AWS SDKs** – If you're using a programming language that AWS provides an SDK for, you can use an SDK to access MediaStore. SDKs simplify authentication, integrate easily with your development environment, and provide easy access to MediaStore commands. For more information, see Tools for Amazon Web Services.
- **AWS Tools for Windows PowerShell** – For more information, see the AWS Tools for PowerShell User Guide.

Pricing for AWS Elemental MediaStore

As with other AWS products, there are no contracts or minimum commitments for using MediaStore. You are charged a per GB ingest fee when content enters into the service and a per GB monthly fee for content that you store in the service. For more information, see AWS Elemental MediaStore Pricing.

Regions and endpoints for AWS Elemental MediaStore

To reduce data latency in your applications, MediaStore offers a regional endpoint to make your request:

```
https://mediastore.<region>.amazonaws.com
```

To view the complete list of AWS Regions where MediaStore is available, see AWS Elemental MediaStore endpoints and quotas in the AWS General Reference.

Setting Up AWS Elemental MediaStore

This section guides you through the steps required to configure users to access AWS Elemental MediaStore. For background and additional information about identity and access management for MediaStore, see Identity and Access Management for AWS Elemental MediaStore.

To start using AWS Elemental MediaStore, complete the following steps.

Topics

Sign up for an AWS account

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 or text message and entering a verification code on the phone keypad.

   When you sign up for an AWS account, an *AWS account root user* is created. The root user has access to all AWS services and resources in the account. As a security best practice, assign administrative access to a user, and use only the root user to perform tasks that require root user access.

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 a user with administrative access

After you sign up for an AWS account, secure your AWS account root user, enable AWS IAM Identity Center, and create an administrative user so that you don't use the root user for everyday tasks.

Secure your AWS account root user

1. Sign in to the AWS Management Console as the account owner by choosing **Root user** and entering your AWS account email address. On the next page, enter your password.

   For help signing in by using root user, see Signing in as the root user in the *AWS Sign-In User Guide*.

2. Turn on multi-factor authentication (MFA) for your root user.

   For instructions, see Enable a virtual MFA device for your AWS account root user (console) in the *IAM User Guide*.

Create a user with administrative access

1. Enable IAM Identity Center.

   For instructions, see Enabling AWS IAM Identity Center in the *AWS IAM Identity Center User Guide*.

2. In IAM Identity Center, grant administrative access to a user.

   For a tutorial about using the IAM Identity Center directory as your identity source, see Configure user access with the default IAM Identity Center directory in the *AWS IAM Identity Center User Guide*.

Sign in as the user with administrative access

- To sign in with your IAM Identity Center user, use the sign-in URL that was sent to your email address when you created the IAM Identity Center user.

  For help signing in using an IAM Identity Center user, see Signing in to the AWS access portal in the *AWS Sign-In User Guide*.

Assign access to additional users

1. In IAM Identity Center, create a permission set that follows the best practice of applying least-privilege permissions.

   For instructions, see Create a permission set in the *AWS IAM Identity Center User Guide*.

2. Assign users to a group, and then assign single sign-on access to the group.

   For instructions, see Add groups in the *AWS IAM Identity Center User Guide*.

Getting started with AWS Elemental MediaStore

This Getting Started tutorial shows you how to use AWS Elemental MediaStore to create a container and upload an object.

Topics

Step 1: Access AWS Elemental MediaStore

Once you have set up your AWS account and created users and roles, you sign in to the console for AWS Elemental MediaStore.

To access AWS Elemental MediaStore

- Sign in to the AWS Management Console and open the MediaStore console at https://console.aws.amazon.com/mediastore/.

  > **Note**
  >
  > You can login using any of the IAM credentials you have created for this account. For information about creating IAM credentials, see *Setting Up AWS Elemental MediaStore*.

Step 2: Create a container

You use containers in AWS Elemental MediaStore to store your folders and objects. You can use containers to group related objects in the same way that you use a directory to group files in a file system. You aren't charged when you create containers; you are charged only when you upload an object to a container.

To create a container

1. On the **Containers** page, choose **Create container**.

2. For **Container name**, type a name for your container. For more information, see Rules for container names.
3. Choose **Create container**. AWS Elemental MediaStore adds the new container to a list of containers. Initially, the status of the container is **Creating**, and then it changes to **Active**.

Step 3: Upload an object

You can upload objects (up to 25 MB each) to a container or to a folder within a container. To upload an object to a folder, you specify the path to the folder. If the folder already exists, AWS Elemental MediaStore stores the object in the folder. If the folder doesn't exist, the service creates it, and then stores the object in the folder.

> **Note**
>
> Object file names can contain only letters, numbers, periods (.), underscores (_), tildes (~), and hyphens (-).

To upload an object

1. On the **Containers** page, choose the name of the container that you just created. The details page for the container appears.
2. Choose **Upload object**.
3. For **Target path**, type a path for the folders. For example, `premium/canada`. If any of the folders in the path don't exist yet, AWS Elemental MediaStore creates them automatically.
4. For **Object**, choose **Browse**.
5. Navigate to the appropriate folder, and choose one object to upload.
6. Choose **Open**, and then choose **Upload**.

Step 4: Access an object

You can download your objects to a specified endpoint.

1. On the **Containers** page, choose the name of the container that has the object that you want to download.

2. If the object that you want to download is in a subfolder, continue choosing the folder names until you see the object.
3. Choose the name of the object.
4. On the details page for the object, choose **Download**.

Containers in AWS Elemental MediaStore

You use containers in MediaStore to store your folders and objects. Related objects can be grouped in containers in the same way that you use a directory to group files in a file system. You aren't charged when you create containers; you are charged only when you upload an object to a container. For more information about charges, see AWS Elemental MediaStore Pricing.

Topics

Rules for container names

When you choose a name for your container, remember the following:

- The name must be unique within the current account for the current AWS Region.
- The name can contain uppercase letters, lowercase letters, numbers, and underscores (_).
- The name must be from 1 to 255 characters long.
- Names are case sensitive. For example, you can have a container named `myContainer` and a folder named `mycontainer` because those names are unique.
- A container can't be renamed after it has been created.

Creating a container

You can create up to 100 containers for each AWS account. You can create as many folders as you want, as long as they are not nested more than 10 levels within a container. In addition, you can upload as many objects as you want to each container.

> **Tip**
>
> You can also create a container automatically by using an AWS CloudFormation template. The AWS CloudFormation template manages data for five API actions: creating a container, setting access logging, updating the default container policy, adding a cross-origin resource sharing (CORS) policy, and adding an object lifecycle policy. For more information, see the AWS CloudFormation User Guide.

To create a container (console)

1. Open the MediaStore console at https://console.aws.amazon.com/mediastore/.
2. On the **Containers** page, choose **Create container**.
3. For **Container** name, enter a name for the container. For more information, see Rules for container names.
4. Choose **Create container**. AWS Elemental MediaStore adds the new container to a list of containers. Initially, the status of the container is **Creating**, and then it changes to **Active**.

To create a container (AWS CLI)

- In the AWS CLI, use the `create-container` command:

```
aws mediastore create-container --container-name ExampleContainer --region us-
west-2
```

The following example shows the return value:

```
{
    "Container": {
        "AccessLoggingEnabled": false,
        "CreationTime": 1563557265.0,
        "Name": "ExampleContainer",
        "Status": "CREATING",
        "ARN": "arn:aws:mediastore:us-west-2:111122223333:container/
ExampleContainer"
    }
}
```

Viewing the details for a container

Details for a container include the container policy, endpoint, ARN, and creation time.

To view the details for a container (console)

1. Open the MediaStore console at https://console.aws.amazon.com/mediastore/.
2. On the **Containers** page, choose the name of the container.

   The container details page appears. This page is divided into two sections:

   - The **Objects** section, which lists the objects and folders in the container.
   - The **Container** policy section, which shows the resource-based policy that is associated with this container. For information about resource policies, see Container policies.

To view the details for a container (AWS CLI)

- In the AWS CLI, use the `describe-container` command:

  ```
  aws mediastore describe-container --container-name ExampleContainer --region us-west-2
  ```

  The following example shows the return value:

  ```
  {
      "Container": {
          "CreationTime": 1563558086.0,
          "AccessLoggingEnabled": false,
          "ARN": "arn:aws:mediastore:us-west-2:111122223333:container/
  ExampleContainer",
          "Status": "ACTIVE",
          "Name": "ExampleContainer",
          "Endpoint": "https://aaabbbcccdddee.data.mediastore.us-
  west-2.amazonaws.com"
      }
  }
  ```

Viewing a list of containers

You can view a list of all the containers that are associated with your account.

To view a list of containers (console)

- Open the MediaStore console at https://console.aws.amazon.com/mediastore/.

  The **Containers** page appears, listing all the containers that are associated with your account.

To view a list of containers (AWS CLI)

- In the AWS CLI, use the `list-containers` command.

```
aws mediastore list-containers --region us-west-2
```

The following example shows the return value:

```
{
    "Containers": [
        {
            "CreationTime": 1505317931.0,
            "Endpoint": "https://aaabbbcccdddee.data.mediastore.us-
west-2.amazonaws.com",
            "Status": "ACTIVE",
            "ARN": "arn:aws:mediastore:us-west-2:111122223333:container/
ExampleLiveDemo",
            "AccessLoggingEnabled": false,
            "Name": "ExampleLiveDemo"
        },
        {
            "CreationTime": 1506528818.0,
            "Endpoint": "https://fffgggghhhiiijj.data.mediastore.us-
west-2.amazonaws.com",
            "Status": "ACTIVE",
            "ARN": "arn:aws:mediastore:us-west-2:111122223333:container/
ExampleContainer",
            "AccessLoggingEnabled": false,
            "Name": "ExampleContainer"
        }
    ]
```

```
}
```

Deleting a container

You can delete a container only if it has no objects.

To delete a container (console)

1. Open the MediaStore console at https://console.aws.amazon.com/mediastore/.
2. On the **Containers** page, choose the option to the left of the container name.
3. Choose **Delete**.

To delete a container (AWS CLI)

- In the AWS CLI, use the `delete-container` command:

  ```
  aws mediastore delete-container --container-name=ExampleLiveDemo --region us-west-2
  ```

  This command has no return value.

Policies in AWS Elemental MediaStore

You can apply one or more of these policies to your AWS Elemental MediaStore container:

- Container policy - Sets access rights to all folders and objects within the container. MediaStore sets a default policy that allows users to perform all MediaStore operations on the container. This policy specifies that all operations must be performed over HTTPS. After you create a container, you can edit the container policy.
- Cross-origin resource sharing (CORS) policy - Allows client web applications from one domain to interact with resources in a different domain. MediaStore does not set a default CORS policy.
- Metrics policy - Allows MediaStore to send metrics to Amazon CloudWatch. MediaStore does not set a default metric policy.
- Object lifecycle policy - Controls how long objects remain in a MediaStore container. MediaStore does not set a default object lifecycle policy.

Container policies in AWS Elemental MediaStore

Each container has a resource-based policy that governs access rights to all folders and objects in that container. The default policy, which is automatically attached to all new containers, allows access to all AWS Elemental MediaStore operations on the container. It specifies that this access has the condition of requiring HTTPS for the operations. After you create a container, you can edit the policy that is attached to that container.

You can also specify an object lifecycle policy that governs the expiration date of objects in a container. After objects reach the maximum age that you specify, the service deletes the objects from the container.

Topics

Viewing a container policy

You can use the console or the AWS CLI to view the resource-based policy of a container.

To view a container policy (console)

1. Open the MediaStore console at https://console.aws.amazon.com/mediastore/.
2. On the **Containers** page, choose the container name.

   The container details page appears. The policy is displayed in the **Container policy** section.

To view a container policy (AWS CLI)

- In the AWS CLI, use the `get-container-policy` command:

```
aws mediastore get-container-policy --container-name ExampleLiveDemo --region us-
west-2
```

The following example shows the return value:

```
{
  "Policy": {
    "Version": "2012-10-17",
    "Statement": [
      {
        "Sid": "PublicReadOverHttps",
        "Effect": "Allow",
        "Principal": {
          "AWS": "arn:aws:iam::111122223333:root",
        },
        "Action": [
          "mediastore:GetObject",
          "mediastore:DescribeObject",
        ],
        "Resource": "arn:aws:mediastore:us-
west-2:111122223333:container/ExampleLiveDemo/*",
        "Condition": {
          "Bool": {
            "aws:SecureTransport": "true"
          }
        }
      }
    ]
  }
}
```

Editing a container policy

You can edit the permissions in the default container policy, or you can create a new policy that replaces the default policy. It takes up to five minutes for the new policy to take effect.

To edit a container policy (console)

1. Open the MediaStore console at https://console.aws.amazon.com/mediastore/.
2. On the **Containers** page, choose the container name.
3. Choose **Edit policy**. For examples that show how to set different permissions, see the section called “Example container policies”.
4. Make the appropriate changes, and then choose **Save**.

To edit a container policy (AWS CLI)

1. Create a file that defines the container policy:

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "PublicReadOverHttps",
      "Effect": "Allow",
      "Action": ["mediastore:GetObject", "mediastore:DescribeObject"],
      "Principal": "*",
      "Resource": "arn:aws:mediastore:us-
west-2:111122223333:container/ExampleLiveDemo/*",
      "Condition": {
        "Bool": {
            "aws:SecureTransport": "true"
        }
      }
    }
  ]
}
```

2. In the AWS CLI, use the `put-container-policy` command:

```
aws mediastore put-container-policy --container-name ExampleLiveDemo --
policy file://ExampleContainerPolicy.json --region us-west-2
```

This command has no return value.

Example container policies

The following examples show container policies that are constructed for different user groups.

Topics

Example container policy: Default

When you create a container, AWS Elemental MediaStore automatically attaches the following resource-based policy:

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "MediaStoreFullAccess",
      "Action": [ "mediastore:*" ],
      "Principal":{
          "AWS" : "arn:aws:iam::<aws_account_number>:root"},
      "Effect": "Allow",
      "Resource": "arn:aws:mediastore:<region>:<owner acct number>:container/<container
 name>/*",
      "Condition": {
        "Bool": { "aws:SecureTransport": "true" }
      }
    }
  ]
```

```
}
```

The policy is built into the service, so you don't have to create it. However, you can edit the policy on the container if the permissions in the default policy don't align with the permissions that you want to use for the container.

The default policy that is assigned to all new containers allows access to all MediaStore operations on the container. It specifies that this access has the condition of requiring HTTPS for the operations.

Example container policy: Public read access over HTTPS

This example policy allows users to retrieve an object through an HTTPS request. It allows read access to anyone over a secured SSL/TLS connection: authenticated users and anonymous users (users who are not logged in). The statement has the name `PublicReadOverHttps`. It allows access to the `GetObject` and `DescribeObject` operations on any object (as specified by the * at the end of the resource path). It specifies that this access has the condition of requiring HTTPS for the operations:

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "PublicReadOverHttps",
      "Effect": "Allow",
      "Action": ["mediastore:GetObject", "mediastore:DescribeObject"],
      "Principal": "*",
      "Resource": "arn:aws:mediastore:<region>:<owner acct number>:container/<container
 name>/*",
      "Condition": {
        "Bool": {
            "aws:SecureTransport": "true"
        }
      }
    }
  ]
}
```

Example container policy: Public read access over HTTP or HTTPS

This example policy allows access to the `GetObject` and `DescribeObject` operations on any object (as specified by the * at the end of the resource path). It allows read access to anyone, including all authenticated users and anonymous users (users who are not logged in):

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "PublicReadOverHttpOrHttps",
      "Effect": "Allow",
      "Action": ["mediastore:GetObject", "mediastore:DescribeObject"],
      "Principal": "*",
      "Resource": "arn:aws:mediastore:<region>:<owner acct number>:container/<container
 name>/*",
      "Condition": {
        "Bool": { "aws:SecureTransport": ["true", "false"] }
      }
    }
  ]
}
```

Example container policy: Cross-account read access—HTTP enabled

This example policy allows users to retrieve an object through an HTTP request. It allows this access to authenticated users with cross-account access. The object is not required to be hosted on a server with an SSL/TLS certificate:

```
{
  "Version" : "2012-10-17",
  "Statement" : [ {
    "Sid" : "CrossAccountReadOverHttpOrHttps",
    "Effect" : "Allow",
    "Principal" : {
      "AWS" : "arn:aws:iam::<other acct number>:root"
    },
    "Action" : [ "mediastore:GetObject", "mediastore:DescribeObject" ],
    "Resource" : "arn:aws:mediastore:<region>:<owner acct number>:container/<container
 name>/*",
    "Condition" : {
      "Bool" : {
```

```
        "aws:SecureTransport" : [ "true", "false" ]
      }
    }
  } ]
}
```

Example container policy: Cross-account read access over HTTPS

This example policy allows access to the `GetObject` and `DescribeObject` operations on any object (as specified by the * at the end of the resource path) that is owned by root user user of the specified <other acct number>. It specifies that this access has the condition of requiring HTTPS for the operations:

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "CrossAccountReadOverHttps",
      "Effect": "Allow",
      "Action": ["mediastore:GetObject", "mediastore:DescribeObject"],
      "Principal":{
        "AWS": "arn:aws:iam::<other acct number>:root"},
      "Resource": "arn:aws:mediastore:<region>:<owner acct number>:container/<container
 name>/*",
      "Condition": {
        "Bool": {
            "aws:SecureTransport": "true"
        }
      }
    }
  ]
}
```

Example container policy: Cross-account read access to a role

The example policy allows access to the `GetObject` and `DescribeObject` operations on any object (as specified by the * at the end of the resource path) that is owned by the <owner acct number>. It allows this access to any user of the <other acct number> if that account has assumed the role that is specified in <role name>:

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

```
  "Statement": [
    {
      "Sid": "CrossAccountRoleRead",
      "Effect": "Allow",
      "Action": ["mediastore:GetObject", "mediastore:DescribeObject"],
      "Principal":{
        "AWS": "arn:aws:iam::<other acct number>:role/<role name>"},
      "Resource": "arn:aws:mediastore:<region>:<owner acct number>:container/<container
 name>/*",
    }
  ]
}
```

Example container policy: Cross-account full access to a role

This example policy allows cross-account access to update any object in the account, as long as the user is logged in over HTTP. It also allows cross-account access to delete, download, and describe objects over HTTP or HTTPS to an account that has assumed the specified role:

- The first statement is `CrossAccountRolePostOverHttps`. It allows access to the `PutObject` operation on any object and allows this access to any user of the specified account if that account has assumed the role that is specified in <role name>. It specifies that this access has the condition of requiring HTTPS for the operation (this condition must always be included when providing access to `PutObject`).

  In other words, any principal that has cross-account access can access `PutObject`, but only over HTTPS.

- The second statement is `CrossAccountFullAccessExceptPost`. It allows access to all operations except `PutObject` on any object. It allows this access to any user of the specified account if that account has assumed the role that is specified in <role name>. This access does not have the condition of requiring HTTPS for the operations.

  In other words, any account that has cross-account access can access `DeleteObject`, `GetObject`, and so on (but not `PutObject`), and can do this over HTTP or HTTPS.

  If you don't exclude `PutObject` from the second statement, the statement won't be valid (because if you include `PutObject` you must explicitly set HTTPS as a condition).

```
{
```

```
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "CrossAccountRolePostOverHttps",
      "Effect": "Allow",
      "Action": "mediastore:PutObject",
      "Principal":{
        "AWS": "arn:aws:iam::<other acct number>:role/<role name>"},
      "Resource": "arn:aws:mediastore:<region>:<owner acct number>:container/<container
 name>/*",
      "Condition": {
        "Bool": {
            "aws:SecureTransport": "true"
        }
      }
    },
    {
      "Sid": "CrossAccountFullAccessExceptPost",
      "Effect": "Allow",
      "NotAction": "mediastore:PutObject",
      "Principal":{
        "AWS": "arn:aws:iam::<other acct number>:role/<role name>"},
      "Resource": "arn:aws:mediastore:<region>:<owner acct number>:container/<container
 name>/*"
    }
  ]
}
```

Example container policy: Access restricted to specific IP addresses

This example policy allows access to all AWS Elemental MediaStore operations on objects in the specified container. However, the request must originate from the range of IP addresses specified in the condition.

The condition in this statement identifies the 198.51.100.* range of allowed Internet Protocol version 4 (IPv4) IP addresses, with one exception: 198.51.100.188.

The `Condition` block uses the `IpAddress` and `NotIpAddress` conditions and the `aws:SourceIp` condition key, which is an AWS-wide condition key. The `aws:sourceIp` IPv4 values use the standard CIDR notation. For more information, see IP Address Condition Operators in the IAM User Guide.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "AccessBySpecificIPAddress",
            "Effect": "Allow",
            "Action": [
                "mediastore:GetObject",
                "mediastore:DescribeObject"
            ],
            "Principal": "*",
            "Resource": "arn:aws:mediastore:<region>:<owner acct number>:container/
<container name>/*",
            "Condition": {
                "IpAddress": {
                    "aws:SourceIp": [
                        "198.51.100.0/24"
                    ]
                },
                "NotIpAddress": {
                    "aws:SourceIp": "198.51.100.188/32"
                }
            }
        }
    ]
}
```

Cross-origin resource sharing (CORS) policies in AWS Elemental MediaStore

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 in AWS Elemental MediaStore, you can build rich client-side web applications with MediaStore and selectively allow cross-origin access to your MediaStore resources.

> **Note**
>
> If you are using Amazon CloudFront to distribute content from a container that has a CORS policy, be sure to configure the distribution for AWS Elemental MediaStore (including the step to edit the cache behavior to set up CORS).

This section provides an overview of CORS. The subtopics describe how you can enable CORS using the AWS Elemental MediaStore console, or programmatically using the MediaStore REST API and the AWS SDKs.

Topics

CORS use-case scenarios

The following are example scenarios for using CORS:

- Scenario 1: Suppose you are distributing live streaming video in an AWS Elemental MediaStore container named *LiveVideo*. Your users load the video manifest endpoint `http://livevideo.mediastore.ap-southeast-2.amazonaws.com` from a specific origin like `www.example.com`. You want to use a JavaScript video player to access videos that are originated from this container via unauthenticated `GET` and `PUT` requests. A browser would typically block JavaScript from allowing those requests, but you can set a CORS policy on your container to explicitly enable these requests from `www.example.com`.
- Scenario 2: Suppose you want to host the same live stream as in Scenario 1 from your MediaStore container, but want to allow requests from any origin. You can configure a CORS policy to allow wildcard (*) origins, so that requests from any origin can access the video.

Adding a CORS policy to a container

This section explains how to add a cross-origin resource sharing (CORS) configuration to an AWS Elemental MediaStore container. CORS allows client web applications that are loaded in one domain to interact with resources in another domain.

To configure your container to allow cross-origin requests, you add a CORS policy to the container. A CORS policy defines rules that identify the origins that you allow to access your container, the operations (HTTP methods) supported for each origin, and other operation-specific information.

When you add a CORS policy to the container, the container policies (that govern access rights to the container) continue to apply.

To add a CORS policy (console)

1. Open the MediaStore console at https://console.aws.amazon.com/mediastore/.
2. On the **Containers** page, choose the name of the container that you want to create a CORS policy for.

   The container details page appears.
3. In the **Container CORS policy** section, choose **Create CORS policy**.
4. Insert the policy in JSON format, and then choose **Save**.

To add a CORS policy (AWS CLI)

1. Create a file that defines the CORS policy:

```
[
  {
    "AllowedHeaders": [
      "*"
    ],
    "AllowedMethods": [
      "GET",
      "HEAD"
    ],
    "AllowedOrigins": [
      "*"
    ],
    "MaxAgeSeconds": 3000
```

```
  }
]
```

2. In the AWS CLI, use the `put-cors-policy` command.

```
aws mediastore put-cors-policy --container-name ExampleContainer --cors-policy
 file://corsPolicy.json --region us-west-2
```

This command has no return value.

Viewing a CORS policy

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.

To view a CORS policy (console)

1. Open the MediaStore console at https://console.aws.amazon.com/mediastore/.
2. On the **Containers** page, choose the name of the container that you want to view the CORS policy for.

   The container details page appears, with the CORS policy in the **Container CORS policy** section.

To view a CORS policy (AWS CLI)

- In the AWS CLI, use the `get-cors-policy` command:

```
aws mediastore get-cors-policy --container-name ExampleContainer --region us-west-2
```

The following example shows the return value:

```
{
    "CorsPolicy": [
        {
            "AllowedMethods": [
                "GET",
                "HEAD"
            ],
```

```
            "MaxAgeSeconds": 3000,
            "AllowedOrigins": [
                "*"
            ],
            "AllowedHeaders": [
                "*"
            ]
        }
    ]
}
```

Editing a CORS policy

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.

To edit a CORS policy (console)

1. Open the MediaStore console at https://console.aws.amazon.com/mediastore/.
2. On the **Containers** page, choose the name of the container that you want to edit the CORS policy for.

   The container details page appears.
3. In the **Container CORS policy** section, choose **Edit CORS policy**.
4. Make your changes to the policy, and then choose **Save**.

To edit a CORS policy (AWS CLI)

1. Create a file that defines the updated CORS policy:

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

```
      "https://www.example.com"
    ],
    "MaxAgeSeconds": 3000
  }
]
```

2. In the AWS CLI, use the `put-cors-policy` command.

```
aws mediastore put-cors-policy --container-name ExampleContainer --cors-policy
 file://corsPolicy2.json --region us-west-2
```

This command has no return value.

Deleting a CORS policy

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. Deleting the CORS policy from a container removes permissions for cross-origin requests.

To delete a CORS policy (console)

1. Open the MediaStore console at https://console.aws.amazon.com/mediastore/.
2. On the **Containers** page, choose the name of the container that you want to delete the CORS policy for.

   The container details page appears.
3. In the **Container CORS policy** section, choose **Delete CORS policy**.
4. Choose **Continue** to confirm, and then choose **Save**.

To delete a CORS policy (AWS CLI)

- In the AWS CLI, use the `delete-cors-policy` command:

```
aws mediastore delete-cors-policy --container-name ExampleContainer --region us-
west-2
```

This command has no return value.

Troubleshooting CORS issues

If you encounter unexpected behavior when you access a container that has a CORS policy, follow these steps to troubleshoot the issue.

1. Verify that the CORS policy is attached to the container.

   For instructions, see the section called “Viewing a CORS policy”.

2. Capture the complete request and response using a tool of your choice (such as your browser's developer console). Verify that the CORS policy that is attached to the container includes at least one CORS rule that matches the data in your request, as follows:

   a. Verify that the request has an `Origin` header.

      If the header is missing, AWS Elemental MediaStore does not treat the request as a cross-origin request and does not send CORS response headers back in the response.

   b. Verify that the `Origin` header in your request matches at least one of the `AllowedOrigins` elements in the specific `CORSRule`.

      The scheme, the host, and the port values in the `Origin` request header must match the `AllowedOrigins` in the `CORSRule`. For example, if you set `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 do not match the allowed origin in your configuration.

   c. Verify that the method in your request (or the method specified in the `Access-Control-Request-Method` in case of a preflight request) is one of the `AllowedMethods` 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 `AllowedHeaders` entries for each value in the `Access-Control-Request-Headers` header.

Example CORS policies

The following examples show cross-origin resource sharing (CORS) policies.

Topics

Example CORS policy: Read access for any domain

The following policy allows a webpage from any domain to retrieve content from your AWS Elemental MediaStore container. The request includes all HTTP headers from the originating domain, and the service responds only to HTTP GET and HTTP HEAD requests from the originating domain. The results are cached for 3,000 seconds before a new set of results is delivered.

```
[
  {
    "AllowedHeaders": [
      "*"
    ],
    "AllowedMethods": [
      "GET",
      "HEAD"
    ],
    "AllowedOrigins": [
      "*"
    ],
    "MaxAgeSeconds": 3000
  }
]
```

Example CORS policy: Read access for a specific domain

The following policy allows a webpage from `https://www.example.com` to retrieve content from your AWS Elemental MediaStore container. The request includes all HTTP headers from `https://www.example.com`, and the service responds only to HTTP GET and HTTP HEAD requests from `https://www.example.com`. The results are cached for 3,000 seconds before a new set of results is delivered.

```
[
  {
    "AllowedHeaders": [
      "*"
    ],
    "AllowedMethods": [
      "GET",
```

```
      "HEAD"
    ],
    "AllowedOrigins": [
      "https://www.example.com"
    ],
    "MaxAgeSeconds": 3000
  }
]
```

Object lifecycle policies in AWS Elemental MediaStore

For each container, you can create an object lifecycle policy that governs how long objects should be stored in the container. When objects reach the maximum age that you specify, AWS Elemental MediaStore deletes the objects. You can delete objects after they are no longer needed to save on storage costs.

You can also specify that MediaStore should move objects to the infrequent access (IA) storage class after they reach a certain age. Objects that are stored in the IA storage class have different rates for storage and retrieval than objects that are stored in the standard storage class. For more information, see MediaStore Pricing.

An object lifecycle policy contains rules, which dictate the lifespan of objects by subfolder. (You can't assign an object lifecycle policy to individual objects). You can attach only one object lifecycle policy to a container, but you can add up to 10 rules to each object lifecycle policy. For more information, see Components of an object lifecycle policy.

Topics

Components of an object lifecycle policy

Object lifecycle policies govern how long objects remain in an AWS Elemental MediaStore container. Each object lifecycle policy consists of one or more rules, which dictate the lifespan of objects. A rule can apply to one folder, multiple folders, or the entire container.

You can attach one object lifecycle policy to a container, and each object lifecycle policy can contain up to 10 rules. You can't assign an object lifecycle policy to an individual object.

Rules in an object lifecycle policy

You can create three types of rules:

- Transient data
- Delete object
- Lifecycle transition

Transient data

A transient data rule sets objects to expire within seconds. This type of rule applies only to objects that are added to the container after the policy becomes effective. It takes up to 20 minutes for MediaStore to apply the new policy to the container.

An example of a rule for transient data looks like this:

```
        {
            "definition": {
                "path": [ {"wildcard": "Football/index*.m3u8"} ],
                "seconds_since_create": [
                    {"numeric": [">", 120]}
                ]
            },
            "action": "EXPIRE"
        },
```

Transient data rules have three parts:

- `path`: Always set to `wildcard`. You use this part to define which objects you want to delete. You can use one or more wildcards, represented by an asterisk (*). Each wildcard represents

any combination of zero or more characters. For example, `"path": [ {"wildcard": "Football/index*.m3u8"} ],` applies to all files in the `Football` folder that match the pattern of `index*.m3u8` (such as index.m3u8, index1.m3us8, and index123456.m3u8). You can include up to 10 paths in a single rule.

- `seconds_since_create`: Always set to `numeric`. You can specify a value from 1-300 seconds. You can also set the operator to greater than (>) or greater than or equal to (>=).
- `action`: Always set to `EXPIRE`.

For transient data rules (objects expire within seconds), there is no lag between the expiration of an object and the deletion of the object.

> **Note**
>
> Objects that are subject to a transient data rule are not included in a `list-items` response. In addition, objects that expire because of a transient data rule do not emit a CloudWatch event when they expire.

Delete object

A delete object rule sets objects to expire within days. This type of rule applies to all objects in the container, even if they were added to the container before the policy was created. It takes up to 20 minutes for MediaStore to apply the new policy, but it can take up to 24 hours for the objects to clear from the container.

An example of two rules for deleting objects looks like this:

```
        {
            "definition": {
                "path": [ { "prefix": "FolderName/" }  ],
                "days_since_create": [
                    {"numeric": [">" , 5]}
                ]
            },
            "action": "EXPIRE"
        },
        {
            "definition": {
                "path": [ { "wildcard": "Football/*.ts" }  ],
```

```
                "days_since_create": [
                    {"numeric": [">" , 5]}
                ]
            },
            "action": "EXPIRE"
        }
```

Delete object rules have three parts:

- `path`: Set to either `prefix` or `wildcard`. You can't mix `prefix` and `wildcard` in the same rule. If you want to use both, you must create one rule for `prefix` and a separate rule for `wildcard`, as shown in the example above.
  - `prefix` - You set the path to `prefix` if you want to delete all objects within a particular folder. If the parameter is empty (`"path": [ { "prefix": "" } ],`), the target is all objects that are stored anywhere within the current container. You can include up to 10 `prefix` paths in a single rule.
  - `wildcard` - You set the path to `wildcard` if you want to delete specific objects based on file name and/or file type. You can use one or more wildcards, represented by an asterisk (*). Each wildcard represents any combination of zero or more characters. For example, `"path": [ {"wildcard": "Football/*.ts"} ],` applies to all files in the `Football` folder that match the pattern of `*.ts` (such as filename.ts, filename1.ts, and filename123456.ts). You can include up to 10 `wildcard` paths in a single rule.
- `days_since_create`: Always set to `numeric`. You can specify a value from 1-36,500 days. You can also set the operator to greater than (>) or greater than or equal to (>=).
- `action`: Always set to `EXPIRE`.

For delete object rules (objects expire within days), there might be a slight lag between the expiration of an object and the deletion of the object. However, changes in billing happen as soon as the object expires. For example, if a lifecycle rule specifies 10 `days_since_create`, the account isn't billed for the object after the object is 10 days old, even if the object isn't deleted yet.

Lifecycle transition

A lifecycle transition rule sets objects to be moved to the infrequent access (IA) storage class after they reach a certain age, measured in days. Objects that are stored in the IA storage class have different rates for storage and retrieval than objects that are stored in the standard storage class. For more information, see MediaStore Pricing.

Once an object has moved to the IA storage class, you can't move it back to the standard storage class.

The lifecycle transition rule applies to all objects in the container, even if they were added to the container before the policy was created. It takes up to 20 minutes for MediaStore to apply the new policy, but it can take up to 24 hours for the objects to clear from the container.

An example of a lifecycle transition rule looks like this:

```
        {
            "definition": {
                "path": [
                    {"prefix": "AwardsShow/"}
                ],
                "days_since_create": [
                    {"numeric": [">=" , 30]}
                ]
            },
            "action": "ARCHIVE"
        }
```

Lifecycle transition rules have three parts:

- `path`: Set to either `prefix` or `wildcard`. You can't mix `prefix` and `wildcard` in the same rule. If you want to use both, you must create one rule for `prefix` and a separate rule for `wildcard`.
  - `prefix` - You set the path to `prefix` if you want to transition all objects within a particular folder to the IA storage class. If the parameter is empty (`"path": [ { "prefix": "" } ],`), the target is all objects that are saved anywhere within the current container. You can include up to 10 `prefix` paths in a single rule.
  - `wildcard` - You set the path to `wildcard` if you want to transition specific objects to the IA storage class based on file name and/or file type. You can use one or more wildcards, represented by an asterisk (*). Each wildcard represents any combination of zero or more characters. For example, `"path": [ {"wildcard": "Football/*.ts"} ],` applies to all files in the `Football` folder that match the pattern of `*.ts` (such as filename.ts, filename1.ts, and filename123456.ts). You can include up to 10 `wildcard` paths in a single rule.
- `days_since_create`: Always set to `"numeric": [">=" , 30]`.
- `action`: Always set to `ARCHIVE`.

Example

Suppose that a container named `LiveEvents` has four subfolders: `Football`, `Baseball`, `Basketball`, and `AwardsShow`. The object lifecycle policy assigned to the `LiveEvents` folder might look like this:

```
{
    "rules": [
         {
            "definition": {
                "path": [
                    {"prefix": "Football/"},
                    {"prefix": "Baseball/"}
                ],
                "days_since_create": [
                    {"numeric": [">" , 28]}
                ]
            },
            "action": "EXPIRE"
        },
        {
            "definition": {
                "path": [ { "prefix": "AwardsShow/" }  ],
                "days_since_create": [
                    {"numeric": [">=" , 15]}
                ]
            },
            "action": "EXPIRE"
        },
        {
            "definition": {
                "path": [ { "prefix": "" }  ],
                "days_since_create": [
                    {"numeric": [">" , 40]}
                ]
            },
            "action": "EXPIRE"
        },
        {
            "definition": {
                "path": [ { "wildcard": "Football/*.ts" }  ],
                "days_since_create": [
                    {"numeric": [">" , 20]}
```

```
                ]
            },
            "action": "EXPIRE"
        },
        {
            "definition": {
                "path": [
                    {"wildcard": "Football/index*.m3u8"}
                ],
                "seconds_since_create": [
                    {"numeric": [">" , 15]}
                ]
            },
            "action": "EXPIRE"
        },
        {
                "definition": {
                    "path": [
                        {"prefix": "Program/"}
                    ],
                    "days_since_create": [
                        {"numeric": [">=" , 30]}
                    ]
                },
                "action": "ARCHIVE"
            }
    ]
}
```

The preceding policy specifies the following:

- The first rule instructs AWS Elemental MediaStore to delete objects that are stored in the `LiveEvents/Football` folder and the `LiveEvents/Baseball` folder after they are older than 28 days.
- The second rule instructs the service to delete objects that are stored in the `LiveEvents/AwardsShow` folder when they are 15 days old or older.
- The third rule instructs the service to delete objects that are stored anywhere in the `LiveEvents` container after they are older than 40 days. This rule applies to objects stored directly in the `LiveEvents` container, as well as objects stored in any of the container's four subfolders.

- The fourth rule instructs the service to delete objects in the `Football` folder that match the pattern `*.ts` after they are older than 20 days.
- The fifth rule instructs the service to delete objects in the `Football` folder that match the pattern `index*.m3u8` after they are older than 15 seconds. MediaStore deletes these files 16 seconds after they are placed in the container.
- The sixth rule instructs the service to move objects in the `Program` folder to the IA storage class after they are 30 days old.

For more examples of object lifecycle policies, see Example object lifecycle policies.

Adding an object lifecycle policy to a container

An object lifecycle policy lets you specify how long to store your objects in a container. You set an expiration date, and after the expiration date AWS Elemental MediaStore deletes the objects. It takes up to 20 minutes for the service to apply the new policy to the container.

For information about how to construct a lifecycle policy, see Components of an object lifecycle policy.

> **Note**
>
> For delete object rules (objects expire within days), there might be a slight lag between the expiration of an object and the deletion of the object. However, changes in billing happen as soon as the object expires. For example, if a lifecycle rule specifies 10 `days_since_create`, the account isn't billed for the object after the object is 10 days old, even if the object isn't deleted yet.

To add an object lifecycle policy (console)

1. Open the MediaStore console at https://console.aws.amazon.com/mediastore/.
2. On the **Containers** page, choose the name of the container that you want to create an object lifecycle policy for.

   The container details page appears.
3. In the **Object lifecycle policy** section, choose **Create object lifecycle policy**.
4. Insert the policy in JSON format, and then choose **Save**.

To add an object lifecycle policy (AWS CLI)

1. Create a file that defines the object lifecycle policy:

```
{
    "rules": [
         {
            "definition": {
                "path": [
                    {"prefix": "Football/"},
                    {"prefix": "Baseball/"}
                ],
                "days_since_create": [
                    {"numeric": [">" , 28]}
                ]
            },
            "action": "EXPIRE"
        },
        {
            "definition": {
                "path": [
                    {"wildcard": "AwardsShow/index*.m3u8"}
                ],
                "seconds_since_create": [
                    {"numeric": [">" , 8]}
                ]
            },
            "action": "EXPIRE"
        }
    ]
}
```

2. In the AWS CLI, use the `put-lifecycle-policy` command:

```
aws mediastore put-lifecycle-policy --container-name LiveEvents --lifecycle-
policy file://LiveEventsLifecyclePolicy.json --region us-west-2
```

This command has no return value. The service attaches the specified policy to the container.

Viewing an object lifecycle policy

An object lifecycle policy specifies how long objects should be kept in a container.

To view an object lifecycle policy (console)

1. Open the MediaStore console at https://console.aws.amazon.com/mediastore/.
2. On the **Containers** page, choose the name of the container that you want to view the object lifecycle policy for.

   The container details page appears, with the object lifecycle policy in the **Object lifecycle policy** section.

To view an object lifecycle policy (AWS CLI)

- In the AWS CLI, use the `get-lifecycle-policy` command:

```
aws mediastore get-lifecycle-policy --container-name LiveEvents --region us-west-2
```

The following example shows the return value:

```
{
    "LifecyclePolicy": "{
        "rules": [
            {
                "definition": {
                    "path": [
                        {"prefix": "Football/"},
                        {"prefix": "Baseball/"}
                    ],
                    "days_since_create": [
                        {"numeric": [">" , 28]}
                    ]
                },
             "action": "EXPIRE"
            }
        ]
    }"
}
```

Editing an object lifecycle policy

You can't edit an existing object lifecycle policy. However, you can change an existing policy by uploading a replacement policy. It takes up to 20 minutes for the service to apply the updated policy to the container.

To edit an object lifecycle policy (console)

1. Open the MediaStore console at https://console.aws.amazon.com/mediastore/.
2. On the **Containers** page, choose the name of the container that you want to edit the object lifecycle policy for.

   The container details page appears.
3. In the **Object lifecycle policy** section, choose **Edit object lifecycle policy**.
4. Make your changes to the policy, and then choose **Save**.

To edit an object lifecycle policy (AWS CLI)

1. Create a file that defines the updated object lifecycle policy:

```
{
    "rules": [
         {
            "definition": {
                "path": [
                    {"prefix": "Football/"},
                    {"prefix": "Baseball/"}
                    {"prefix": "Basketball/"}
                ],
                "days_since_create": [
                    {"numeric": [">" , 28]}
                ]
            },
            "action": "EXPIRE"
        }
    ]
}
```

2. In the AWS CLI, use the `put-lifecycle-policy` command:

```
aws mediastore put-lifecycle-policy --container-name LiveEvents --lifecycle-
policy file://LiveEvents2LifecyclePolicy --region us-west-2
```

This command has no return value. The service attaches the specified policy to the container, replacing the previous policy.

Deleting an object lifecycle policy

When you delete an object lifecycle policy, it takes up to 20 minutes for the service to apply the change to the container.

To delete an object lifecycle policy (console)

1. Open the MediaStore console at https://console.aws.amazon.com/mediastore/.
2. On the **Containers** page, choose the name of the container that you want to delete the object lifecycle policy for.

   The container details page appears.
3. In the **Object lifecycle policy** section, choose **Delete lifecycle policy**.
4. Choose **Continue** to confirm, and then choose **Save**.

To delete an object lifecycle policy (AWS CLI)

- In the AWS CLI, use the `delete-lifecycle-policy` command:

```
aws mediastore delete-lifecycle-policy --container-name LiveEvents --region us-
west-2
```

This command has no return value.

Example object lifecycle policies

The following examples show object lifecycle policies.

Topics

- Example object lifecycle policy: Expire within seconds

Example object lifecycle policy: Expire within seconds

The following policy specifies that MediaStore deletes objects that match all of the following criteria:

- The object is added to the container after the policy becomes effective.
- The object is stored in the `Football` folder.
- The object has a file extension of `m3u8`.
- The object has been in the container for more than 20 seconds.

```
{
    "rules": [
        {
            "definition": {
                "path": [
                    {"wildcard": "Football/*.m3u8"}
                ],
                "seconds_since_create": [
                    {"numeric": [ ">", 20 ]}
                ]
            },
            "action": "EXPIRE"
        }
    ]
}
```

Example object lifecycle policy: Expire within days

The following policy specifies that MediaStore deletes objects that match all of the following criteria:

- The object is stored in the `Program` folder

- The object has a file extension of `ts`
- The object has been in the container for more than 5 days

```
{
    "rules": [
        {
            "definition": {
                "path": [
                    {"wildcard": "Program/*.ts"}
                ],
                "days_since_create": [
                    {"numeric": [ ">", 5 ]}
                ]
            },
            "action": "EXPIRE"
        }
    ]
}
```

Example object lifecycle policy: Transition to infrequent access storage class

The following policy specifies that MediaStore moves objects to the infrequent access (IA) storage class when they are 30 days old. Objects that are stored in the IA storage class have different rates for storage and retrieval than objects that are stored in the standard storage class.

The `days_since_create` field must be set to `"numeric": [">=" ,30]`.

```
{
    "rules": [
        {
            "definition": {
                "path": [
                    {"prefix": "Football/"},
                    {"prefix": "Baseball/"}
                ],
                "days_since_create": [
                    {"numeric": [">=" , 30]}
                ]
            },
            "action": "ARCHIVE"
        }
```

```
    ]
}
```

Example object lifecycle policy: Multiple rules

The following policy specifies that MediaStore does the following:

- Move objects that are stored in the `AwardsShow` folder to the infrequent access (IA) storage class after 30 days
- Delete objects that have a file extension of `m3u8` and are stored in the `Football` folder after 20 seconds
- Delete objects that are stored in the `April` folder after 10 days
- Delete objects that have a file extension of `ts` and are stored in the `Program` folder after 5 days

```
{
    "rules": [
        {
            "definition": {
                "path": [
                    {"prefix": "AwardsShow/"}
                ],
                "days_since_create": [
                    {"numeric": [ ">=" , 30 ]}
                ]
            },
            "action": "ARCHIVE"
        },
        {
            "definition": {
                "path": [
                    {"wildcard": "Football/*.m3u8"}
                ],
                "seconds_since_create": [
                    {"numeric": [ ">", 20 ]}
                ]
            },
            "action": "EXPIRE"
        },
        {
            "definition": {
                "path": [
```

```
                    {"prefix": "April"}
                ],
                "days_since_create": [
                    {"numeric": [ ">", 10 ]}
                ]
            },
            "action": "EXPIRE"
        },
        {
            "definition": {
                "path": [
                    {"wildcard": "Program/*.ts"}
                ],
                "days_since_create": [
                    {"numeric": [ ">", 5 ]}
                ]
            },
            "action": "EXPIRE"
        }
    ]
}
```

Example object lifecycle policy: Empty container

The following object lifecycle policy specifies that MediaStore deletes all objects in the container, including folders and subfolders, 1 day after they are added to the container. If the container holds any objects before this policy is applied, MediaStore deletes the objects 1 day after the policy becomes effective. It takes up to 20 minutes for the service to apply the new policy to the container.

```
{
    "rules": [
        {
            "definition": {
                "path": [
                    {"wildcard": "*"}
                ],
                "days_since_create": [
                    {"numeric": [ ">=", 1 ]}
                ]
            },
            "action": "EXPIRE"
        }
```

```
    ]
}
```

Metric policies in AWS Elemental MediaStore

For each container, you can add a metric policy to allow AWS Elemental MediaStore to send metrics to Amazon CloudWatch. It takes up to 20 minutes for the new policy to take effect. For a description of each MediaStore metric, see MediaStore metrics.

A metric policy contains the following:

- A setting to enable or disable metrics at the container level.
- Anywhere from zero to five rules that enable metrics at the object level. If the policy contains rules, each rule must include both of the following:
  - An object group that defines which objects to include in the group. The definition can be a path or a file name, but it can't have more than 900 characters. Valid characters are: a-z, A-Z, 0-9, _ (underscore), = (equal), : (colon), . (period), - (hyphen), ~ (tilde), / (forward slash), and * (asterisk). Wildcards (*) are acceptable.
  - An object group name that allows you to refer to the object group. The name can't have more than 30 characters. Valid characters are: a-z, A-Z, 0-9, and _ (underscore).

If an object matches multiple rules, CloudWatch displays a data point for each matching rule. For example, if an object matches two rules named `rule1` and `rule2`, CloudWatch displays two data points for these rules. The first has a dimension of `ObjectGroupName=rule1` and the second has a dimension of `ObjectGroupName=rule2`.

Topics

Adding a metric policy

A metric policy contains rules that dictate which metrics AWS Elemental MediaStore sends to Amazon CloudWatch. For examples of metric policies, see Example metric policies.

To add a metric policy (console)

1. Open the MediaStore console at https://console.aws.amazon.com/mediastore/.
2. On the **Containers** page, choose the name of the container that you want to add a metric policy to.

   The container details page appears.
3. In the **Metric policy** section, choose **Create metric policy**.
4. Insert the policy in JSON format, and then choose **Save**.

Viewing a metric policy

You can use the console or the AWS CLI to view the metric policy of a container.

To view a metric policy (console)

1. Open the MediaStore console at https://console.aws.amazon.com/mediastore/.
2. On the **Containers** page, choose the container name.

   The container details page appears. The policy is displayed in the **Metric policy** section.

Editing a metric policy

A metric policy contains rules that dictate which metrics AWS Elemental MediaStore sends to Amazon CloudWatch. When you edit an existing metric policy, it takes up to 20 minutes for the new policy to take effect. For examples of metric policies, see Example metric policies.

To edit a metric policy (console)

1. Open the MediaStore console at https://console.aws.amazon.com/mediastore/.
2. On the **Containers** page, choose the container name.
3. In the **Metric policy** section, choose **Edit metric policy**.
4. Make the appropriate changes, and then choose **Save**.

Example metric policies

The following examples show metric policies that are constructed for different use cases.

Topics

Example metric policy: Container-level metrics

This example policy indicates that AWS Elemental MediaStore should send metrics to Amazon CloudWatch at the container level. For example, this includes the `RequestCount` metric that counts the number of `Put` requests made to the container. Alternatively, you can set this to `DISABLED`.

Because there are no rules in this policy, MediaStore does not send metrics at the path level. For example, you can't see how many `Put` requests were made to a particular folder within this container.

```
{
   "ContainerLevelMetrics": "ENABLED"
}
```

Example metric policy: Path-level metrics

This example policy indicates that AWS Elemental MediaStore should not send metrics to Amazon CloudWatch at the container level. In addition, MediaStore should send metrics for objects in two specific folders: `baseball/saturday` and `football/saturday`. The metrics for MediaStore requests are as follows:

- Requests to the `baseball/saturday` folder have a CloudWatch dimension of `ObjectGroupName=baseballGroup`.
- Requests to the `football/saturday` folder have a dimension `ObjectGroupName=footballGroup`.

```
{
```

```
   "ContainerLevelMetrics": "DISABLED",
   "MetricPolicyRules": [
     {
       "ObjectGroup": "baseball/saturday",
       "ObjectGroupName": "baseballGroup"
     },
     {
       "ObjectGroup": "football/saturday",
       "ObjectGroupName": "footballGroup"
     }
   ]
}
```

Example metric policy: Container-level and path-level metrics

This example policy indicates that AWS Elemental MediaStore should send metrics to Amazon CloudWatch at the container level. In addition, MediaStore should send metrics for objects in two specific folders: `baseball/saturday` and `football/saturday`. The metrics for MediaStore requests are as follows:

- Requests to the `baseball/saturday` folder have a CloudWatch dimension of `ObjectGroupName=baseballGroup`.
- Requests to the `football/saturday` folder have a CloudWatch dimension `ObjectGroupName=footballGroup`.

```
{
   "ContainerLevelMetrics": "ENABLED",
   "MetricPolicyRules": [
     {
       "ObjectGroup": "baseball/saturday",
       "ObjectGroupName": "baseballGroup"
     },
     {
       "ObjectGroup": "football/saturday",
       "ObjectGroupName": "footballGroup"
     }
   ]
}
```

Example metric policy: Path-level metrics using wildcards

This example policy indicates that AWS Elemental MediaStore should send metrics to Amazon CloudWatch at the container level. In addition, MediaStore should also send metrics for objects based on their file name. A wildcard indicates that the objects can be stored anywhere in the container and they can have any file name, as long as it ends with a `.m3u8` extension.

```
{
   "ContainerLevelMetrics": "ENABLED",
   "MetricPolicyRules": [
     {
       "ObjectGroup": "*.m3u8",
       "ObjectGroupName": "index"
     }
   ]
 }
```

Example metric policy: Path-level metrics with overlapping rules

This example policy indicates that AWS Elemental MediaStore should send metrics to Amazon CloudWatch at the container level. In addition, MediaStore should send metrics for two folders: `sports/football/saturday` and `sports/football`.

The metrics for MediaStore requests to the `sports/football/saturday` folder have a CloudWatch dimension of `ObjectGroupName=footballGroup1`. Because objects that are stored in the `sports/football` folder match both rules, CloudWatch displays two data points for these objects: one with a dimension of `ObjectGroupName=footballGroup1` and the second with a dimension of `ObjectGroupName=footballGroup2`.

```
{
   "ContainerLevelMetrics": "ENABLED",
   "MetricPolicyRules": [
     {
       "ObjectGroup": "sports/football/saturday",
       "ObjectGroupName": "footballGroup1"
     },
     {
       "ObjectGroup": "sports/football",
       "ObjectGroupName": "footballGroup2"
     }
   ]
```

```
}
```

Folders in AWS Elemental MediaStore

Folders are divisions within a container. You use folders to subdivide your container in the same way that you create subfolders to divide a folder in a file system. You can create up to 10 levels of folders (not including the container itself).

Folders are optional; you can choose to upload your objects directly to a container instead of a folder. However, folders are an easy way to organize your objects.

To upload an object to a folder, you specify the path to the folder. If the folder already exists, AWS Elemental MediaStore stores the object in the folder. If the folder doesn't exist, the service creates it, and then stores the object in the folder.

For example, suppose you have a container named `movies`, and you upload a file named `mlaw.ts` with the path `premium/canada`. AWS Elemental MediaStore stores the object in the subfolder canada under the folder premium. If neither folder exists, the service creates both the `premium` folder and the `canada` subfolder, and then stores your object in the `canada` subfolder. If you specify only the container `movies` (with no path), the service stores the object directly in the container.

AWS Elemental MediaStore automatically deletes a folder when you delete the last object in that folder. The service also deletes any empty folders above that folder. For example, suppose that you have a folder named premium that doesn't contain any files but does contain one subfolder named `canada`. The `canada` subfolder contains one file named `mlaw.ts`. If you delete the file `mlaw.ts`, the service deletes both the `premium` and `canada` folders. This automatic deletion applies only to folders. The service does not delete empty containers.

Topics

Rules for folder names

When you choose a name for your folder, remember the following:

- The name can contain only the following characters: uppercase letters (A-Z), lowercase letters (a-z), numbers (0-9), periods (.), hyphens (-), tildes (~), underscores (_), equal signs (=), and colons (:).
- The name must be at least one character long. Empty folder names (such as `folder1//folder3/`) are not allowed.
- Names are case sensitive. For example, you can have a folder named `myFolder` and a folder named `myfolder` in the same container or folder because those names are unique.
- The name must be unique only within its parent container or folder. For example, you can create a folder named `myfolder` in two different containers: `movies/myfolder` and `sports/myfolder`.
- The name can have the same name as its parent container.
- The folder can't be renamed after it has been created.

Creating a folder

You can create folders when you upload objects. To upload an object to a folder, you specify the path to the folder. If the folder already exists, AWS Elemental MediaStore stores the object in the folder. If the folder doesn't exist, the service creates it, and then stores the object in the folder.

For more information, see the section called "Uploading an object".

Deleting a folder

You can delete folders only if the folder is empty; you can't delete folders that contain objects.

AWS Elemental MediaStore automatically deletes a folder when you delete the last object in that folder. The service also deletes any empty folders above that folder. For example, suppose that you have a folder named `premium` that doesn't contain any files but does contain one subfolder named `canada`. The `canada` subfolder contains one file named `mlaw.ts`. If you delete the file `mlaw.ts`, the service deletes both the `premium` and `canada` folders. This automatic deletion applies only to folders. The service does not delete empty containers.

For more information, see Deleting an object.

Objects in AWS Elemental MediaStore

AWS Elemental MediaStore assets are called *objects*. You can upload an object to a container or to a folder within the container.

In MediaStore, you can upload, download, and delete objects:

- **Upload** – Add an object to a container or folder. This is not the same as creating an object. You must create your objects locally before you can upload them to MediaStore.
- **Download** – Copy an object from MediaStore to another location. This does not remove the object from MediaStore.
- **Delete** – Remove an object from MediaStore completely. You can delete objects individually, or you can add an object lifecycle policy to automatically delete objects within a container after a specified duration.

MediaStore accepts all file types.

Topics

Uploading an object

You can upload objects to a container or to a folder within a container. To upload an object to a folder, you specify the path to the folder. If the folder already exists, AWS Elemental MediaStore stores the object in the folder. If the folder doesn't exist, the service creates it, and then stores the object in the folder. For more information about folders, see Folders in AWS Elemental MediaStore.

You can use the MediaStore console or the AWS CLI to upload objects.

MediaStore supports chunked transfer of objects, which reduces latency by making an object available for downloading while it is still being uploaded. To use this capability, set the object's

upload availability to `streaming`. You can set the value of this header when you upload the object using the API. If you don't specify this header in your request, MediaStore assigns the default value of `standard` for the object's upload availability.

Object sizes can't exceed 25 MB for standard upload availability and 10 MB for streaming upload availability.

> **Note**
>
> Object file names can contain only letters, numbers, periods (.), underscores (_), tildes (~), hyphens (-), equal signs (=), and colons (:).

To upload an object (console)

1. Open the MediaStore console at https://console.aws.amazon.com/mediastore/.
2. On the **Containers** page, choose the name of the container. The details panel for the container appears.
3. Choose **Upload object**.
4. For **Target path**, type a path for the folders. For example, `premium/canada`. If any of the folders in the path that you specify don't exist yet, the service creates them automatically.
5. In the **Object** section, choose **Browse**.
6. Navigate to the appropriate folder, and choose one object to upload.
7. Choose **Open**, and then choose **Upload**.

> **Note**
>
> If a file with the same name already exists in the selected folder, the service replaces the original file with the uploaded file.

To upload an object (AWS CLI)

- In the AWS CLI, use the `put-object` command. You can also include any of the following parameters: `content-type`, `cache-control` (to allow the caller to control the object's cache behavior), and `path` (to put the object in a folder within the container).

> **Note**
>
> After you upload the object, you can't edit the `content-type`, `cache-control`, or `path`.

```
aws mediastore-data put-object --endpoint https://
aaabbbcccdddee.data.mediastore.us-west-2.amazonaws.com --body README.md --path /
folder_name/README.md --cache-control "max-age=6, public" --content-type binary/
octet-stream --region us-west-2
```

The following example shows the return value:

```
{
    "ContentSHA256":
 "74b5fdb517f423ed750ef214c44adfe2be36e37d861eafe9c842cbe1bf387a9d",
    "StorageClass": "TEMPORAL",
    "ETag": "af3e4731af032167a106015d1f2fe934e68b32ed1aa297a9e325f5c64979277b"
}
```

Viewing a list of objects

You can use the AWS Elemental MediaStore console to view items (objects and folders) stored in the top-most level of a container or in a folder. Items stored in a subfolder of the current container or folder will not be displayed. You can use the AWS CLI to view a list of objects and folders within a container, regardless of how many folders or subfolders are within the container.

To view a list of objects in a specific container (console)

1. Open the MediaStore console at https://console.aws.amazon.com/mediastore/.
2. On the **Containers** page, choose the name of the container that has the folder that you want to view.
3. Choose the name of the folder from the list.

   A details page appears, showing all folders and objects that are stored in the folder.

To view a list of objects in a specific folder (console)

1. Open the MediaStore console at https://console.aws.amazon.com/mediastore/.
2. On the **Containers** page, choose the name of the container that has the folder that you want to view.

   A details page appears, showing all folders and objects that are stored in the container.

To view a list of objects and folders in a specific container (AWS CLI)

- In the AWS CLI, use the `list-items` command:

```
aws mediastore-data list-items --endpoint https://
aaabbbcccdddee.data.mediastore.us-west-2.amazonaws.com --region us-west-2
```

The following example shows the return value:

```
{
    "Items": [
        {
            "ContentType": "image/jpeg",
            "LastModified": 1563571859.379,
            "Name": "filename.jpg",
            "Type": "OBJECT",
            "ETag":
 "543ab21abcd1a234ab123456a1a2b12345ab12abc12a1234abc1a2bc12345a12",
            "ContentLength": 3784
        },
        {
            "Type": "FOLDER",
            "Name": "ExampleLiveDemo"
        }
    ]
}
```

> **Note**
>
> Objects that are subject to a `seconds_since_create` rule are not included in a `list-items` response.

To view a list of objects and folders in a specific folder (AWS CLI)

- In the AWS CLI, use the `list-items` command, with the specified folder name at the end of the request:

```
aws mediastore-data list-items --endpoint https://
aaabbbcccdddee.data.mediastore.us-west-2.amazonaws.com --path /folder_name --
region us-west-2
```

The following example shows the return value:

```
{
    "Items": [
        {
            "Type": "FOLDER",
            "Name": "folder_1"
        },
        {
            "LastModified": 1563571940.861,
            "ContentLength": 2307346,
            "Name": "file1234.jpg",
            "ETag":
 "111a1a22222a1a1a222abc333a444444b55ab1111ab2222222222ab333333a2b",
            "ContentType": "image/jpeg",
            "Type": "OBJECT"
        }
    ]
}
```

> **Note**
>
> Objects that are subject to a `seconds_since_create` rule are not included in a `list-items` response.

Viewing the details of an object

After you upload an object, AWS Elemental MediaStore stores details such as the modification date, content length, ETag (entity tag), and content type. To learn how an object's metadata is used, see MediaStore's interaction with HTTP caches.

To view the details of an object (console)

1. Open the MediaStore console at https://console.aws.amazon.com/mediastore/.
2. On the **Containers** page, choose the name of container that has the object that you want to view.
3. If the object that you want to view is in a folder, continue choosing the folder names until you see the object.
4. Choose the name of the object.

   A details page appears, showing information about the object.

To view the details of an object (AWS CLI)

- In the AWS CLI, use the `describe-object` command:

  ```
  aws mediastore-data describe-object --endpoint https://
  aaabbbcccdddee.data.mediastore.us-west-2.amazonaws.com --path /folder_name/
  file1234.jpg --region us-west-2
  ```

  The following example shows the return value:

  ```
  {
      "ContentType": "image/jpeg",
      "LastModified": "Fri, 19 Jul 2019 21:32:20 GMT",
      "ContentLength": "2307346",
      "ETag": "2aa333bbcc8d8d22d777e999c88d4aa9eeeeee4dd89ff7f555555555555da6d3"
  }
  ```

Downloading an object

You can use the console to download an object. You can use the AWS CLI to download an object or only part of an object.

To download an object (console)

1. Open the MediaStore console at https://console.aws.amazon.com/mediastore/.
2. On the **Containers** page, choose the name of container that has the object that you want to download.

3. If the object that you want to download is in a folder, continue choosing the folder names until you see the object.
4. Choose the name of the object.
5. On the **Object** details page, choose **Download**.

To download an object (AWS CLI)

- In the AWS CLI, use the `get-object` command:

```
aws mediastore-data get-object --endpoint https://
aaabbbcccdddee.data.mediastore.us-west-2.amazonaws.com --path=/folder_name/
README.md README.md --region us-west-2
```

The following example shows the return value:

```
{
    "ContentLength": "2307346",
    "ContentType": "image/jpeg",
    "LastModified": "Fri, 19 Jul 2019 21:32:20 GMT",
    "ETag": "2aa333bbcc8d8d22d777e999c88d4aa9eeeeee4dd89ff7f555555555555da6d3",
    "StatusCode": 200
}
```

To download part of an object (AWS CLI)

- In the AWS CLI, use the `get-object` command, and specify a range.

```
aws mediastore-data get-object --endpoint https://
aaabbbcccdddee.data.mediastore.us-west-2.amazonaws.com --path /folder_name/
README.md --range="bytes=0-100" README2.md --region us-west-2
```

The following example shows the return value:

```
{
    "StatusCode": 206,
    "ContentRange": "bytes 0-100/2307346",
    "ContentLength": "101",
    "LastModified": "Fri, 19 Jul 2019 21:32:20 GMT",
```

```
    "ContentType": "image/jpeg",
    "ETag": "2aa333bbcc8d8d22d777e999c88d4aa9eeeeee4dd89ff7f555555555555da6d3"
}
```

Deleting objects

AWS Elemental MediaStore offers different options for deleting objects from containers:

- Delete an individual object. No charges apply.
- Empty a container to delete all objects within a container at once. Because this process uses API calls, normal API charges apply.
- Add an object lifecycle policy to delete objects when they reach a certain age. No charges apply.

Deleting an object

You can delete objects individually using the console or the AWS CLI. Alternatively, you can add an object lifecycle policy to automatically delete objects after they reach a certain age in a container, or you can empty a container to delete all objects within that container.

> **Note**
>
> When you delete the only object in a folder, AWS Elemental MediaStore automatically deletes the folder and any empty folders above that folder. For example, suppose that you have a folder named `premium` that doesn't contain any files but does contain one subfolder named `canada`. The `canada` subfolder contains one file named `mlaw.ts`. If you delete the file `mlaw.ts`, the service deletes both the `premium` and `canada` folders.

To delete an object (console)

1. Open the MediaStore console at https://console.aws.amazon.com/mediastore/.
2. On the **Containers** page, choose the name of the container that has the object that you want to delete.
3. If the object that you want to delete is in a folder, continue choosing the folder names until you see the object.
4. Choose the option to the left of the object name.

5. Choose **Delete**.

To delete an object (AWS CLI)

- In the AWS CLI, use the `delete-object` command.

  Example:

  ```
  aws mediastore-data --region us-west-2 delete-object --endpoint=https://
  aaabbbcccdddee.data.mediastore.us-west-2.amazonaws.com --path=/folder_name/
  README.md
  ```

  This command has no return value.

Emptying a container

You can empty a container to delete all objects that are stored within the container. Alternatively, you can add an object lifecycle policy to automatically delete objects after they reach a certain age in a container, or you can delete objects individually.

To empty a container (console)

1. Open the MediaStore console at https://console.aws.amazon.com/mediastore/.
2. On the **Containers** page, choose the option for the container that you want to empty.
3. Choose **Empty container**. A confirmation message appears.
4. Confirm that you want to empty the container by entering the container name into the text field, then choose **Empty**.

Security in AWS Elemental MediaStore

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 AWS Elemental MediaStore, 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 MediaStore. The following topics show you how to configure MediaStore to meet your security and compliance objectives. You also learn how to use other AWS services that help you to monitor and secure your MediaStore resources.

Topics

Data protection in AWS Elemental MediaStore

The AWS shared responsibility model applies to data protection in AWS Elemental MediaStore. As described in this model, AWS is responsible for protecting the global infrastructure that runs all of the AWS Cloud. You are responsible for maintaining control over your content that is hosted on this infrastructure. You are also responsible for the security configuration and management tasks for the AWS services that you use. For more information about data privacy, see the Data Privacy FAQ. For information about data protection in Europe, see the AWS Shared Responsibility Model and GDPR blog post on the *AWS Security Blog*.

For data protection purposes, we recommend that you protect AWS account credentials and set up individual users with AWS IAM Identity Center or AWS Identity and Access Management (IAM). That way, each user is given only the permissions necessary to fulfill their job duties. We also recommend that you secure your data in the following ways:

- Use multi-factor authentication (MFA) with each account.
- Use SSL/TLS to communicate with AWS resources. We require TLS 1.2 and recommend TLS 1.3.
- Set up API and user activity logging with AWS CloudTrail. For information about using CloudTrail trails to capture AWS activities, see Working with CloudTrail trails in the *AWS CloudTrail User Guide*.
- Use AWS encryption solutions, along with all default security controls within AWS services.
- Use advanced managed security services such as Amazon Macie, which assists in discovering and securing sensitive data that is stored in Amazon S3.
- If you require FIPS 140-3 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) 140-3.

We strongly recommend that you never put confidential or sensitive information, such as your customers' email addresses, into tags or free-form text fields such as a **Name** field. This includes when you work with MediaStore or other AWS services using the console, API, AWS CLI, or AWS SDKs. Any data that you enter into tags or free-form text fields used for names may be used for billing or diagnostic logs. If you provide a URL to an external server, we strongly recommend that you do not include credentials information in the URL to validate your request to that server.

Data encryption

MediaStore encrypts containers and objects at rest using the industry standard AES-256 algorithm. We recommend that you use MediaStore to secure your data in the following ways:

- Create a container policy to control access rights to all folders and objects in that container. For more information, see the section called “Container policies”.
- Create a cross-origin resource sharing (CORS) policy to allow cross-origin access selectively to your MediaStore resources. With CORS, you can allow client web applications that are loaded in one domain to interact with resources in a different domain. For more information, see the section called “CORS policies”.

Identity and Access Management for AWS Elemental MediaStore

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 MediaStore resources. IAM is an AWS service that you can use with no additional charge.

Topics

Audience

How you use AWS Identity and Access Management (IAM) differs, depending on the work that you do in MediaStore.

Service user – If you use the MediaStore service to do your job, then your administrator provides you with the credentials and permissions that you need. As you use more MediaStore features to do your work, you might need additional permissions. Understanding how access is managed can help you request the right permissions from your administrator. If you cannot access a feature in MediaStore, see Troubleshooting AWS Elemental MediaStore identity and access.

Service administrator – If you're in charge of MediaStore resources at your company, you probably have full access to MediaStore. It's your job to determine which MediaStore features and resources your service users should access. You must then submit requests to your IAM administrator to change the permissions of your service users. Review the information on this page to understand the basic concepts of IAM. To learn more about how your company can use IAM with MediaStore, see How AWS Elemental MediaStore works with IAM.

IAM administrator – If you're an IAM administrator, you might want to learn details about how you can write policies to manage access to MediaStore. To view example MediaStore identity-based policies that you can use in IAM, see Identity-based policy examples for AWS Elemental MediaStore.

Authenticating with identities

Authentication is how you sign in to AWS using your identity credentials. You must be *authenticated* (signed in to AWS) as the AWS account root user, as an IAM user, or by assuming an IAM role.

You can sign in to AWS as a federated identity by using credentials provided through an identity source. AWS IAM Identity Center (IAM Identity Center) users, your company's single sign-on authentication, and your Google or Facebook credentials are examples of federated identities. When you sign in as a federated identity, your administrator previously set up identity federation using IAM roles. When you access AWS by using federation, you are indirectly assuming a role.

Depending on the type of user you are, you can sign in to the AWS Management Console or the AWS access portal. For more information about signing in to AWS, see How to sign in to your AWS account in the *AWS Sign-In User Guide*.

If you access AWS programmatically, AWS provides a software development kit (SDK) and a command line interface (CLI) to cryptographically sign your requests by using your credentials. If you don't use AWS tools, you must sign requests yourself. For more information about using the recommended method to sign requests yourself, see AWS Signature Version 4 for API requests in the *IAM User Guide*.

Regardless of the authentication method that you use, you might be required to provide additional security information. For example, AWS recommends that you use multi-factor authentication (MFA) to increase the security of your account. To learn more, see Multi-factor authentication in the *AWS IAM Identity Center User Guide* and AWS Multi-factor authentication in IAM in the *IAM User Guide*.

AWS account root user

When you 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. We strongly recommend that you don't 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 *IAM User Guide*.

Federated identity

As a best practice, require human users, including users that require administrator access, to use federation with an identity provider to access AWS services by using temporary credentials.

A *federated identity* is a user from your enterprise user directory, a web identity provider, the AWS Directory Service, the Identity Center directory, or any user that accesses AWS services by using credentials provided through an identity source. When federated identities access AWS accounts, they assume roles, and the roles provide temporary credentials.

For centralized access management, we recommend that you use AWS IAM Identity Center. You can create users and groups in IAM Identity Center, or you can connect and synchronize to a set of users and groups in your own identity source for use across all your AWS accounts and applications. For information about IAM Identity Center, see What is IAM Identity Center? in the *AWS IAM Identity Center User Guide*.

IAM users and groups

An *IAM user* is an identity within your AWS account that has specific permissions for a single person or application. Where possible, we recommend relying on temporary credentials instead of creating IAM users who have long-term credentials such as passwords and access keys. However, if you have specific use cases that require long-term credentials with IAM users, we recommend that you rotate access keys. For more information, see Rotate access keys regularly for use cases that require long-term credentials in the *IAM User Guide*.

An *IAM group* is an identity that specifies a collection of IAM users. You can't sign in as a group. You can use groups to specify permissions for multiple users at a time. Groups make permissions easier to manage for large sets of users. For example, you could have a group named *IAMAdmins* and give that group permissions to administer IAM resources.

Users are different from roles. A user is uniquely associated with one person or application, but a role is intended to be assumable by anyone who needs it. Users have permanent long-term credentials, but roles provide temporary credentials. To learn more, see Use cases for IAM users in the *IAM User Guide*.

IAM roles

An *IAM role* is an identity within your AWS account that has specific permissions. It is similar to an IAM user, but is not associated with a specific person. To temporarily assume an IAM role in the AWS Management Console, you can switch from a user to an IAM role (console). You can assume a role by calling an AWS CLI or AWS API operation or by using a custom URL. For more information about methods for using roles, see Methods to assume a role in the *IAM User Guide*.

IAM roles with temporary credentials are useful in the following situations:

- **Federated user access** – To assign permissions to a federated identity, you create a role and define permissions for the role. When a federated identity authenticates, the identity is associated with the role and is granted the permissions that are defined by the role. For information about roles for federation, see Create a role for a third-party identity provider (federation) in the *IAM User Guide*. If you use IAM Identity Center, you configure a permission set. To control what your identities can access after they authenticate, IAM Identity Center correlates the permission set to a role in IAM. For information about permissions sets, see Permission sets in the *AWS IAM Identity Center User Guide*.
- **Temporary IAM user permissions** – An IAM user or role can assume an IAM role to temporarily take on different permissions for a specific task.
- **Cross-account access** – You can use an IAM role to allow someone (a trusted principal) in a different account to access resources in your account. Roles are the primary way to grant cross-account access. However, with some AWS services, you can attach a policy directly to a resource (instead of using a role as a proxy). To learn the difference between roles and resource-based policies for cross-account access, see Cross account resource access in IAM in the *IAM User Guide*.
- **Cross-service access** – Some AWS services use features in other AWS services. For example, when you make a call in a service, it's common for that service to run applications in Amazon EC2 or

store objects in Amazon S3. A service might do this using the calling principal's permissions, using a service role, or using a service-linked role.

- **Forward access sessions (FAS)** – When you use an IAM user or role to perform actions in AWS, you are considered a principal. When you use some services, you might perform an action that then initiates another action in a different service. FAS uses the permissions of the principal calling an AWS service, combined with the requesting AWS service to make requests to downstream services. FAS requests are only made when a service receives a request that requires interactions with other AWS services or resources to complete. In this case, you must have permissions to perform both actions. For policy details when making FAS requests, see Forward access sessions.
- **Service role** – A service role is an IAM role that a service assumes to perform actions on your behalf. An IAM administrator can create, modify, and delete a service role from within IAM. For more information, see Create a role to delegate permissions to an AWS service in the *IAM User Guide*.
- **Service-linked role** – A service-linked role is a type of service role that is linked to an AWS service. The service can assume the role to perform an action on your behalf. Service-linked roles appear in your AWS account and are owned by the service. An IAM administrator can view, but not edit the permissions for service-linked roles.

- **Applications running on Amazon EC2** – You can use an IAM role to manage temporary credentials for applications that are running on an EC2 instance and making AWS CLI or AWS API requests. This is preferable to storing access keys within the EC2 instance. To assign an AWS role to an EC2 instance and make it available to all of its applications, you create an instance profile that is attached to the instance. An instance profile contains the role and enables programs that are running on the EC2 instance to get temporary credentials. For more information, see Use an IAM role to grant permissions to applications running on Amazon EC2 instances in the *IAM User Guide*.

Managing access using policies

You control access in AWS by creating policies and attaching them to AWS identities or resources. A policy is an object in AWS that, when associated with an identity or resource, defines their permissions. AWS evaluates these policies when a principal (user, root user, or role session) makes a request. Permissions in the policies determine whether the request is allowed or denied. Most policies are stored in AWS as JSON documents. For more information about the structure and contents of JSON policy documents, see Overview of JSON policies in the *IAM User Guide*.

Administrators can use AWS JSON policies to specify who has access to what. That is, which **principal** can perform **actions** on what **resources**, and under what **conditions**.

By default, users and roles have no permissions. To grant users permission to perform actions on the resources that they need, an IAM administrator can create IAM policies. The administrator can then add the IAM policies to roles, and users can assume the roles.

IAM policies define permissions for an action regardless of the method that you use to perform the operation. For example, suppose that you have a policy that allows the `iam:GetRole` action. A user with that policy can get role information from the AWS Management Console, the AWS CLI, or the AWS API.

Identity-based policies

Identity-based policies are JSON permissions policy documents that you can attach to an identity, such as an IAM user, group of users, or role. These policies control what actions users and roles can perform, on which resources, and under what conditions. To learn how to create an identity-based policy, see Define custom IAM permissions with customer managed policies in the *IAM User Guide*.

Identity-based policies can be further categorized as *inline policies* or *managed policies*. Inline policies are embedded directly into a single user, group, or role. Managed policies are standalone policies that you can attach to multiple users, groups, and roles in your AWS account. Managed policies include AWS managed policies and customer managed policies. To learn how to choose between a managed policy or an inline policy, see Choose between managed policies and inline policies in the *IAM User Guide*.

Resource-based policies

Resource-based policies are JSON policy documents that you attach to a resource. Examples of resource-based policies are IAM *role trust policies* and Amazon S3 *bucket policies*. In services that support resource-based policies, service administrators can use them to control access to a specific resource. For the resource where the policy is attached, the policy defines what actions a specified principal can perform on that resource and under what conditions. You must specify a principal in a resource-based policy. Principals can include accounts, users, roles, federated users, or AWS services.

Resource-based policies are inline policies that are located in that service. You can't use AWS managed policies from IAM in a resource-based policy.

Access control lists (ACLs)

Access control lists (ACLs) control which principals (account members, users, or roles) have permissions to access a resource. ACLs are similar to resource-based policies, although they do not use the JSON policy document format.

Amazon S3, AWS WAF, and Amazon VPC are examples of services that support ACLs. To learn more about ACLs, see Access control list (ACL) overview in the *Amazon Simple Storage Service Developer Guide*.

Other policy types

AWS supports additional, less-common policy types. These policy types can set the maximum permissions granted to you by the more common policy types.

- **Permissions boundaries** – A permissions boundary is an advanced feature in which you set the maximum permissions that an identity-based policy can grant to an IAM entity (IAM user or role). You can set a permissions boundary for an entity. The resulting permissions are the intersection of an entity's identity-based policies and its permissions boundaries. Resource-based policies that specify the user or role in the `Principal` field are not limited by the permissions boundary. An explicit deny in any of these policies overrides the allow. For more information about permissions boundaries, see Permissions boundaries for IAM entities in the *IAM User Guide*.
- **Service control policies (SCPs)** – SCPs are JSON policies that specify the maximum permissions for an organization or organizational unit (OU) in AWS Organizations. AWS Organizations is a service for grouping and centrally managing multiple AWS accounts that your business owns. If you enable all features in an organization, then you can apply service control policies (SCPs) to any or all of your accounts. The SCP limits permissions for entities in member accounts, including each AWS account root user. For more information about Organizations and SCPs, see Service control policies in the *AWS Organizations User Guide*.
- **Resource control policies (RCPs)** – RCPs are JSON policies that you can use to set the maximum available permissions for resources in your accounts without updating the IAM policies attached to each resource that you own. The RCP limits permissions for resources in member accounts and can impact the effective permissions for identities, including the AWS account root user, regardless of whether they belong to your organization. For more information about Organizations and RCPs, including a list of AWS services that support RCPs, see Resource control policies (RCPs) in the *AWS Organizations User Guide*.
- **Session policies** – Session policies are advanced policies that you pass as a parameter when you programmatically create a temporary session for a role or federated user. The resulting session's

permissions are the intersection of the user or role's identity-based policies and the session policies. Permissions can also come from a resource-based policy. An explicit deny in any of these policies overrides the allow. For more information, see Session policies in the *IAM User Guide*.

Multiple policy types

When multiple types of policies apply to a request, the resulting permissions are more complicated to understand. To learn how AWS determines whether to allow a request when multiple policy types are involved, see Policy evaluation logic in the *IAM User Guide*.

How AWS Elemental MediaStore works with IAM

Before you use IAM to manage access to MediaStore, learn what IAM features are available to use with MediaStore.

IAM features you can use with AWS Elemental MediaStore

IAM feature	MediaStore support
Identity-based policies	Yes
Resource-based policies	Yes
Policy actions	Yes
Policy resources	Yes
Policy condition keys (service-specific)	Yes
ACLs	No
ABAC (tags in policies)	Partial
Temporary credentials	Yes
Principal permissions	Yes
Service roles	Yes
Service-linked roles	No

To get a high-level view of how MediaStore and other AWS services work with most IAM features, see AWS services that work with IAM in the *IAM User Guide*.

Identity-based policies for MediaStore

Supports identity-based policies: Yes

Identity-based policies are JSON permissions policy documents that you can attach to an identity, such as an IAM user, group of users, or role. These policies control what actions users and roles can perform, on which resources, and under what conditions. To learn how to create an identity-based policy, see Define custom IAM permissions with customer managed policies in the *IAM User Guide*.

With IAM identity-based policies, you can specify allowed or denied actions and resources as well as the conditions under which actions are allowed or denied. You can't specify the principal in an identity-based policy because it applies to the user or role to which it is attached. To learn about all of the elements that you can use in a JSON policy, see IAM JSON policy elements reference in the *IAM User Guide*.

Identity-based policy examples for MediaStore

To view examples of MediaStore identity-based policies, see Identity-based policy examples for AWS Elemental MediaStore.

Resource-based policies within MediaStore

Supports resource-based policies: Yes

Resource-based policies are JSON policy documents that you attach to a resource. Examples of resource-based policies are IAM *role trust policies* and Amazon S3 *bucket policies*. In services that support resource-based policies, service administrators can use them to control access to a specific resource. For the resource where the policy is attached, the policy defines what actions a specified principal can perform on that resource and under what conditions. You must specify a principal in a resource-based policy. Principals can include accounts, users, roles, federated users, or AWS services.

To enable cross-account access, you can specify an entire account or IAM entities in another account as the principal in a resource-based policy. Adding a cross-account principal to a resource-based policy is only half of establishing the trust relationship. When the principal and the resource are in different AWS accounts, an IAM administrator in the trusted account must also grant the principal entity (user or role) permission to access the resource. They grant permission by

attaching an identity-based policy to the entity. However, if a resource-based policy grants access to a principal in the same account, no additional identity-based policy is required. For more information, see Cross account resource access in IAM in the *IAM User Guide*.

> **Note**
>
> MediaStore also supports container policies that define which principal entities (accounts, users, roles, and federated users) can perform actions on the container. For more information, see Container policies.

Policy actions for MediaStore

Supports policy actions: Yes

Administrators can use AWS JSON policies to specify who has access to what. That is, which **principal** can perform **actions** on what **resources**, and under what **conditions**.

The `Action` element of a JSON policy describes the actions that you can use to allow or deny access in a policy. Policy actions usually have the same name as the associated AWS API operation. There are some exceptions, such as *permission-only actions* that don't have a matching API operation. There are also some operations that require multiple actions in a policy. These additional actions are called *dependent actions*.

Include actions in a policy to grant permissions to perform the associated operation.

To see a list of MediaStore actions, see Actions defined by AWS Elemental MediaStore in the *Service Authorization Reference*.

Policy actions in MediaStore use the following prefix before the action:

```
mediastore
```

To specify multiple actions in a single statement, separate them with commas.

```
"Action": [
      "mediastore:action1",
      "mediastore:action2"
         ]
```

To view examples of MediaStore identity-based policies, see Identity-based policy examples for AWS Elemental MediaStore.

Policy resources for MediaStore

Supports policy resources: Yes

Administrators can use AWS JSON policies to specify who has access to what. That is, which **principal** can perform **actions** on what **resources**, and under what **conditions**.

The `Resource` JSON policy element specifies the object or objects to which the action applies. Statements must include either a `Resource` or a `NotResource` element. As a best practice, specify a resource using its Amazon Resource Name (ARN). You can do this for actions that support a specific resource type, known as *resource-level permissions*.

For actions that don't support resource-level permissions, such as listing operations, use a wildcard (*) to indicate that the statement applies to all resources.

```
"Resource": "*"
```

To see a list of MediaStore resource types and their ARNs, see Resources defined by AWS Elemental MediaStore in the *Service Authorization Reference*. To learn with which actions you can specify the ARN of each resource, see Actions defined by AWS Elemental MediaStore.

The MediaStore container resource has the following ARN:

```
arn:${Partition}:mediastore:${Region}:${Account}:container/${containerName}
```

For more information about the format of ARNs, see Amazon Resource Names (ARNs) and AWS Service Namespaces.

For example, to specify the `AwardsShow` container in your statement, use the following ARN:

```
"Resource": "arn:aws:mediastore:us-east-1:111122223333:container/AwardsShow"
```

Policy condition keys for MediaStore

Supports service-specific policy condition keys: Yes

Administrators can use AWS JSON policies to specify who has access to what. That is, which **principal** can perform **actions** on what **resources**, and under what **conditions**.

The `Condition` element (or `Condition` *block*) lets you specify conditions in which a statement is in effect. The `Condition` element is optional. You can create conditional expressions that use condition operators, such as equals or less than, to match the condition in the policy with values in the request.

If you specify multiple `Condition` elements in a statement, or multiple keys in a single `Condition` element, AWS evaluates them using a logical `AND` operation. If you specify multiple values for a single condition key, AWS evaluates the condition using a logical `OR` operation. All of the conditions must be met before the statement's permissions are granted.

You can also use placeholder variables when you specify conditions. For example, you can grant an IAM user permission to access a resource only if it is tagged with their IAM user name. For more information, see IAM policy elements: variables and tags in the *IAM User Guide*.

AWS supports global condition keys and service-specific condition keys. To see all AWS global condition keys, see AWS global condition context keys in the *IAM User Guide*.

To see a list of MediaStore condition keys, see Condition keys for AWS Elemental MediaStore in the *Service Authorization Reference*. To learn with which actions and resources you can use a condition key, see Actions defined by AWS Elemental MediaStore.

To view examples of MediaStore identity-based policies, see Identity-based policy examples for AWS Elemental MediaStore.

ACLs in MediaStore

Supports ACLs: No

Access control lists (ACLs) control which principals (account members, users, or roles) have permissions to access a resource. ACLs are similar to resource-based policies, although they do not use the JSON policy document format.

ABAC with MediaStore

Supports ABAC (tags in policies): Partial

Attribute-based access control (ABAC) is an authorization strategy that defines permissions based on attributes. In AWS, these attributes are called *tags*. You can attach tags to IAM entities (users or roles) and to many AWS resources. Tagging entities and resources is the first step of ABAC. Then

you design ABAC policies to allow operations when the principal's tag matches the tag on the resource that they are trying to access.

ABAC is helpful in environments that are growing rapidly and helps with situations where policy management becomes cumbersome.

To control access based on tags, you provide tag information in the condition element of a policy using the `aws:ResourceTag/key-name`, `aws:RequestTag/key-name`, or `aws:TagKeys` condition keys.

If a service supports all three condition keys for every resource type, then the value is **Yes** for the service. If a service supports all three condition keys for only some resource types, then the value is **Partial**.

For more information about ABAC, see Define permissions with ABAC authorization in the *IAM User Guide*. To view a tutorial with steps for setting up ABAC, see Use attribute-based access control (ABAC) in the *IAM User Guide*.

Using temporary credentials with MediaStore

Supports temporary credentials: Yes

Some AWS services don't work when you sign in using temporary credentials. For additional information, including which AWS services work with temporary credentials, see AWS services that work with IAM in the *IAM User Guide*.

You are using temporary credentials if you sign in to the AWS Management Console using any method except a user name and password. For example, when you access AWS using your company's single sign-on (SSO) link, that process automatically creates temporary credentials. You also automatically create temporary credentials when you sign in to the console as a user and then switch roles. For more information about switching roles, see Switch from a user to an IAM role (console) in the *IAM User Guide*.

You can manually create temporary credentials using the AWS CLI or AWS API. You can then use those temporary credentials to access AWS. AWS recommends that you dynamically generate temporary credentials instead of using long-term access keys. For more information, see Temporary security credentials in IAM.

Cross-service principal permissions for MediaStore

Supports forward access sessions (FAS): Yes

When you use an IAM user or role to perform actions in AWS, you are considered a principal. When you use some services, you might perform an action that then initiates another action in a different service. FAS uses the permissions of the principal calling an AWS service, combined with the requesting AWS service to make requests to downstream services. FAS requests are only made when a service receives a request that requires interactions with other AWS services or resources to complete. In this case, you must have permissions to perform both actions. For policy details when making FAS requests, see Forward access sessions.

Service roles for MediaStore

Supports service roles: Yes

A service role is an IAM role that a service assumes to perform actions on your behalf. An IAM administrator can create, modify, and delete a service role from within IAM. For more information, see Create a role to delegate permissions to an AWS service in the *IAM User Guide*.

> **Warning**
>
> Changing the permissions for a service role might break MediaStore functionality. Edit service roles only when MediaStore provides guidance to do so.

Service-linked roles for MediaStore

Supports service-linked roles: No

A service-linked role is a type of service role that is linked to an AWS service. The service can assume the role to perform an action on your behalf. Service-linked roles appear in your AWS account and are owned by the service. An IAM administrator can view, but not edit the permissions for service-linked roles.

For details about creating or managing service-linked roles, see AWS services that work with IAM. Find a service in the table that includes a `Yes` in the **Service-linked role** column. Choose the **Yes** link to view the service-linked role documentation for that service.

Identity-based policy examples for AWS Elemental MediaStore

By default, users and roles don't have permission to create or modify MediaStore resources. They also can't perform tasks by using the AWS Management Console, AWS Command Line Interface (AWS CLI), or AWS API. To grant users permission to perform actions on the resources that they

need, an IAM administrator can create IAM policies. The administrator can then add the IAM policies to roles, and users can assume the roles.

To learn how to create an IAM identity-based policy by using these example JSON policy documents, see Create IAM policies (console) in the *IAM User Guide*.

For details about actions and resource types defined by MediaStore, including the format of the ARNs for each of the resource types, see Actions, resources, and condition keys for AWS Elemental MediaStore in the *Service Authorization Reference*.

Topics

Policy best practices

Identity-based policies determine whether someone can create, access, or delete MediaStore resources in your account. These actions can incur costs for your AWS account. When you create or edit identity-based policies, follow these guidelines and recommendations:

- **Get started with AWS managed policies and move toward least-privilege permissions** – To get started granting permissions to your users and workloads, use the *AWS managed policies* that grant permissions for many common use cases. They are available in your AWS account. We recommend that you reduce permissions further by defining AWS customer managed policies that are specific to your use cases. For more information, see AWS managed policies or AWS managed policies for job functions in the *IAM User Guide*.
- **Apply least-privilege permissions** – When you set permissions with IAM policies, grant only the permissions required to perform a task. You do this by defining the actions that can be taken on specific resources under specific conditions, also known as *least-privilege permissions*. For more information about using IAM to apply permissions, see Policies and permissions in IAM in the *IAM User Guide*.
- **Use conditions in IAM policies to further restrict access** – You can add a condition to your policies to limit access to actions and resources. For example, you can write a policy condition to specify that all requests must be sent using SSL. You can also use conditions to grant access to service actions if they are used through a specific AWS service, such as AWS CloudFormation. For more information, see IAM JSON policy elements: Condition in the *IAM User Guide*.

- **Use IAM Access Analyzer to validate your IAM policies to ensure secure and functional permissions** – IAM Access Analyzer validates new and existing policies so that the policies adhere to the IAM policy language (JSON) and IAM best practices. IAM Access Analyzer provides more than 100 policy checks and actionable recommendations to help you author secure and functional policies. For more information, see Validate policies with IAM Access Analyzer in the *IAM User Guide*.
- **Require multi-factor authentication (MFA)** – If you have a scenario that requires IAM users or a root user in your AWS account, turn on MFA for additional security. To require MFA when API operations are called, add MFA conditions to your policies. For more information, see Secure API access with MFA in the *IAM User Guide*.

For more information about best practices in IAM, see Security best practices in IAM in the *IAM User Guide*.

Using the MediaStore console

To access the AWS Elemental MediaStore console, you must have a minimum set of permissions. These permissions must allow you to list and view details about the MediaStore resources in your AWS account. If you create an identity-based policy that is more restrictive than the minimum required permissions, the console won't function as intended for entities (users or roles) with that policy.

You don't need to allow minimum console permissions for users that are making calls only to the AWS CLI or the AWS API. Instead, allow access to only the actions that match the API operation that they're trying to perform.

To ensure that users and roles can still use the MediaStore console, also attach the MediaStore `ConsoleAccess` or `ReadOnly` AWS managed policy to the entities. For more information, see Adding permissions to a user in the *IAM User Guide*.

Allow users to view their own permissions

This example shows how you might create a policy that allows IAM users to view the inline and managed policies that are attached to their user identity. This policy includes permissions to complete this action on the console or programmatically using the AWS CLI or AWS API.

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

```
    "Statement": [
        {
            "Sid": "ViewOwnUserInfo",
            "Effect": "Allow",
            "Action": [
                "iam:GetUserPolicy",
                "iam:ListGroupsForUser",
                "iam:ListAttachedUserPolicies",
                "iam:ListUserPolicies",
                "iam:GetUser"
            ],
            "Resource": ["arn:aws:iam::*:user/${aws:username}"]
        },
        {
            "Sid": "NavigateInConsole",
            "Effect": "Allow",
            "Action": [
                "iam:GetGroupPolicy",
                "iam:GetPolicyVersion",
                "iam:GetPolicy",
                "iam:ListAttachedGroupPolicies",
                "iam:ListGroupPolicies",
                "iam:ListPolicyVersions",
                "iam:ListPolicies",
                "iam:ListUsers"
            ],
            "Resource": "*"
        }
    ]
}
```

Troubleshooting AWS Elemental MediaStore identity and access

Use the following information to help you diagnose and fix common issues that you might encounter when working with MediaStore and IAM.

Topics

- I am not authorized to perform an action in MediaStore
- I am not authorized to perform iam:PassRole

- I want to allow people outside of my AWS account to access my MediaStore resources

I am not authorized to perform an action in MediaStore

If you receive an error that you're not authorized to perform an action, your policies must be updated to allow you to perform the action.

The following example error occurs when the `mateojackson` IAM user tries to use the console to view details about a fictional *`my-example-widget`* resource but doesn't have the fictional `mediastore:`*`GetWidget`* permissions.

```
User: arn:aws:iam::123456789012:user/mateojackson is not authorized to perform:
 mediastore:GetWidget on resource: my-example-widget
```

In this case, the policy for the `mateojackson` user must be updated to allow access to the *`my-example-widget`* resource by using the `mediastore:`*`GetWidget`* action.

If you need help, contact your AWS administrator. Your administrator is the person who provided you with your sign-in credentials.

I am not authorized to perform iam:PassRole

If you receive an error that you're not authorized to perform the `iam:PassRole` action, your policies must be updated to allow you to pass a role to MediaStore.

Some AWS services allow you to pass an existing role to that service instead of creating a new service role or service-linked role. To do this, you must have permissions to pass the role to the service.

The following example error occurs when an IAM user named `marymajor` tries to use the console to perform an action in MediaStore. However, the action requires the service to have permissions that are granted by a service role. Mary does not have permissions to pass the role to the service.

```
User: arn:aws:iam::123456789012:user/marymajor is not authorized to perform:
 iam:PassRole
```

In this case, Mary's policies must be updated to allow her to perform the `iam:PassRole` action.

If you need help, contact your AWS administrator. Your administrator is the person who provided you with your sign-in credentials.

I want to allow people outside of my AWS account to access my MediaStore resources

You can create a role that users in other accounts or people outside of your organization can use to access your resources. You can specify who is trusted to assume the role. For services that support resource-based policies or access control lists (ACLs), you can use those policies to grant people access to your resources.

To learn more, consult the following:

- To learn whether MediaStore supports these features, see How AWS Elemental MediaStore works with IAM.
- To learn how to provide access to your resources across AWS accounts that you own, see Providing access to an IAM user in another AWS account that you own in the *IAM User Guide*.
- To learn how to provide access to your resources to third-party AWS accounts, see Providing access to AWS accounts owned by third parties in the *IAM User Guide*.
- To learn how to provide access through identity federation, see Providing access to externally authenticated users (identity federation) in the *IAM User Guide*.
- To learn the difference between using roles and resource-based policies for cross-account access, see Cross account resource access in IAM in the *IAM User Guide*.

Logging and monitoring in AWS Elemental MediaStore

This section provides an overview of the options for logging and monitoring in AWS Elemental MediaStore for security purposes. For more information about logging and monitoring in MediaStore, see Monitoring and tagging in AWS Elemental MediaStore.

Monitoring is an important part of maintaining the reliability, availability, and performance of AWS Elemental MediaStore and your AWS solutions. You should collect monitoring data from all 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 MediaStore resources and responding to potential incidents.

Amazon CloudWatch alarms

Using 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 don't 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 with CloudWatch.

AWS CloudTrail logs

CloudTrail provides a record of actions taken by a user, role, or an AWS service in AWS Elemental MediaStore. Using the information collected by CloudTrail, you can determine the request that was made to MediaStore, 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 API calls with CloudTrail.

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.

For more information, see AWS Trusted Advisor.

Compliance validation for AWS Elemental MediaStore

To learn whether an AWS service is within the scope of specific compliance programs, see AWS services in Scope by Compliance Program and choose the compliance program that you are interested in. For general information, see AWS Compliance Programs.

You can download third-party audit reports using AWS Artifact. For more information, see Downloading Reports in AWS Artifact.

Your compliance responsibility when using AWS services is determined by the sensitivity of your data, your company's compliance objectives, and applicable laws and regulations. AWS provides the following resources to help with compliance:

- Security Compliance & Governance – These solution implementation guides discuss architectural considerations and provide steps for deploying security and compliance features.
- HIPAA Eligible Services Reference – Lists HIPAA eligible services. Not all AWS services are HIPAA eligible.

- AWS Compliance Resources – This collection of workbooks and guides might apply to your industry and location.
- AWS Customer Compliance Guides – Understand the shared responsibility model through the lens of compliance. The guides summarize the best practices for securing AWS services and map the guidance to security controls across multiple frameworks (including National Institute of Standards and Technology (NIST), Payment Card Industry Security Standards Council (PCI), and International Organization for Standardization (ISO)).
- Evaluating Resources with Rules in the *AWS Config Developer Guide* – The AWS Config service assesses how well your resource configurations comply with internal practices, industry guidelines, and regulations.
- AWS Security Hub – This AWS service provides a comprehensive view of your security state within AWS. Security Hub uses security controls to evaluate your AWS resources and to check your compliance against security industry standards and best practices. For a list of supported services and controls, see Security Hub controls reference.
- Amazon GuardDuty – This AWS service detects potential threats to your AWS accounts, workloads, containers, and data by monitoring your environment for suspicious and malicious activities. GuardDuty can help you address various compliance requirements, like PCI DSS, by meeting intrusion detection requirements mandated by certain compliance frameworks.
- AWS Audit Manager – This AWS service helps you continuously audit your AWS usage to simplify how you manage risk and compliance with regulations and industry standards.

Resilience in AWS Elemental MediaStore

The AWS global infrastructure is built around AWS Regions and Availability Zones. AWS Regions provide multiple physically separated and isolated Availability Zones, which are connected with low-latency, high-throughput, and highly redundant networking. With Availability Zones, you can design and operate applications and databases that automatically fail over between zones without interruption. Availability Zones are more highly available, fault tolerant, and scalable than traditional single or multiple data center infrastructures.

For more information about AWS Regions and Availability Zones, see AWS Global Infrastructure.

In addition to the AWS global infrastructure, MediaStore offers several features to help support your data resiliency and backup needs.

Infrastructure Security in AWS Elemental MediaStore

As a managed service, AWS Elemental MediaStore is protected by AWS global network security. For information about AWS security services and how AWS protects infrastructure, see AWS Cloud Security. To design your AWS environment using the best practices for infrastructure security, see Infrastructure Protection in *Security Pillar AWS Well-Architected Framework*.

You use AWS published API calls to access MediaStore through the network. Clients must support the following:

- Transport Layer Security (TLS). We require TLS 1.2 and recommend TLS 1.3.
- Cipher suites with perfect forward secrecy (PFS) such as DHE (Ephemeral Diffie-Hellman) or ECDHE (Elliptic Curve Ephemeral Diffie-Hellman). Most modern systems such as Java 7 and later support these modes.

Additionally, requests must be signed by using an access key ID and a secret access key that is associated with an IAM principal. Or you can use the AWS Security Token Service (AWS STS) to generate temporary security credentials to sign requests.

Cross-service confused deputy prevention

The confused deputy problem is a security issue where an entity that doesn't have permission to perform an action can coerce a more-privileged entity to perform the action. In AWS, cross-service impersonation can result in the confused deputy problem. Cross-service impersonation can occur when one service (the *calling service*) calls another service (the *called service*). The calling service can be manipulated to use its permissions to act on another customer's resources in a way it should not otherwise have permission to access. To prevent this, AWS provides tools that help you protect your data for all services with service principals that have been given access to resources in your account.

We recommend using the `aws:SourceArn` and `aws:SourceAccount` global condition context keys in resource policies to limit the permissions that AWS Elemental MediaStore gives another service to the resource. Use `aws:SourceArn` if you want only one resource to be associated with the cross-service access. Use `aws:SourceAccount` if you want to allow any resource in that account to be associated with the cross-service use.

The most effective way to protect against the confused deputy problem is to use the `aws:SourceArn` global condition context key with the full ARN of the resource. If you don't know

the full ARN of the resource or if you are specifying multiple resources, use the `aws:SourceArn` global context condition key with wildcard characters (*) for the unknown portions of the ARN. For example, `arn:aws:`*`servicename`*`:*:`*`123456789012`*`:*`.

If the `aws:SourceArn` value does not contain the account ID, such as an Amazon S3 bucket ARN, you must use both global condition context keys to limit permissions.

The value of `aws:SourceArn` must be the configuration that MediaStore publishes CloudWatch logs for in your Region and account.

The following example shows how you can use the `aws:SourceArn` and `aws:SourceAccount` global condition context keys in MediaStore to prevent the confused deputy problem.

```
{
  "Version": "2012-10-17",
  "Statement": {
    "Sid": "ConfusedDeputyPreventionExamplePolicy",
    "Effect": "Allow",
    "Principal": {
      "Service": "servicename.amazonaws.com"
    },
    "Action": "servicename:ActionName",
    "Resource": [
      "arn:aws:servicename:::ResourceName/*"
    ],
    "Condition": {
      "ArnLike": {
        "aws:SourceArn": "arn:aws:servicename:*:123456789012:*"
      },
      "StringEquals": {
        "aws:SourceAccount": "123456789012"
      }
    }
  }
}
```

Monitoring and tagging in AWS Elemental MediaStore

Monitoring is an important part of maintaining the reliability, availability, and performance of AWS Elemental MediaStore and your other AWS solutions. AWS provides the following monitoring tools to watch MediaStore, report when something is wrong, and take automatic actions when appropriate:

- *AWS CloudTrail* captures API calls and related events made by or on behalf of your AWS account and delivers the log files to an Amazon S3 bucket that you specify. You can identify which users and accounts called AWS, the source IP address from which the calls were made, and when the calls occurred. For more information, see the AWS CloudTrail User Guide.
- *Amazon CloudWatch* monitors your AWS resources and the applications that you run on AWS in real time. You can collect and track metrics, create customized dashboards, and set alarms that notify you or take actions when a specified metric reaches a threshold that you specify. For example, you can have CloudWatch track CPU usage or other metrics of your Amazon EC2 instances and automatically launch new instances when needed. For more information, see the Amazon CloudWatch User Guide.
- *Amazon CloudWatch Events* delivers a stream of system events that describe changes in AWS resources. Typically, AWS services deliver event notifications to CloudWatch Events in seconds but can sometimes take a minute or longer. CloudWatch Events enables automated event-driven computing, as you can write rules that watch for certain events and trigger automated actions in other AWS services when these events happen. For more information, see the Amazon CloudWatch Events User Guide.
- *Amazon CloudWatch Logs* enables you to monitor, store, and access your log files from Amazon EC2 instances, CloudTrail, and other sources. CloudWatch Logs can monitor information in the log files and notify you when certain thresholds are met. You can also archive your log data in highly durable storage. For more information, see the Amazon CloudWatch Logs User Guide.

You can also assign metadata to your MediaStore containers in the form of tags. Each tag is a label that consists of a key and value that you define. Tags can make it easier to manage, search for, and filter resources. You can use tags to organize your AWS resources in the AWS Management Console, create usage and billing reports across all of your AWS resources, and filter resources during infrastructure automation activities.

Topics

Logging AWS Elemental MediaStore API calls with AWS CloudTrail

AWS Elemental MediaStore is integrated with AWS CloudTrail, a service that provides a record of actions taken by a user, role, or an AWS service in MediaStore. CloudTrail captures a subset of API calls for MediaStore as events, including calls from the MediaStore console and from code calls to the MediaStore API. If you create a trail, you can enable continuous delivery of CloudTrail events to an Amazon S3 bucket, including events for MediaStore. 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 MediaStore, the IP address from which the request was made, who made the request, when it was made, and more.

To learn more about CloudTrail, including how to configure and enable it, see the AWS CloudTrail User Guide.

Topics

AWS Elemental MediaStore information in CloudTrail

CloudTrail is enabled on your AWS account when you create the account. When supported event activity occurs in AWS Elemental MediaStore, 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 MediaStore, 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 AWS 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 topics:

- Overview for Creating a Trail
- CloudTrail Supported Services and Integrations
- Configuring Amazon SNS Notifications for CloudTrail
- Receiving CloudTrail Log Files from Multiple Regions and Receiving CloudTrail Log Files from Multiple Accounts

AWS Elemental MediaStore supports logging the following operations as events in CloudTrail log files:

- `CreateContainer`
- `DeleteContainer`
- `DeleteContainerPolicy`
- `DeleteCorsPolicy`
- `DescribeContainer`
- `GetContainerPolicy`
- `GetCorsPolicy`
- `ListContainers`
- `PutContainerPolicy`
- `PutCorsPolicy`

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 user or 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.

Example: AWS Elemental MediaStore log file entries

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 and 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.

The following example shows a CloudTrail log entry that demonstrates the `CreateContainer` operation:


```
{
    "eventVersion": "1.05",
    "userIdentity": {
        "type": "IAMUser",
        "principalId": "ABCDEFGHIJKL123456789",
        "arn": "arn:aws:iam::111122223333:user/testUser",
        "accountId": "111122223333",
        "accessKeyId": "AKIAIOSFODNN7EXAMPLE",
        "userName": "testUser",
        "sessionContext": {
            "attributes": {
                "mfaAuthenticated": "false",
                "creationDate": "2018-07-09T12:55:42Z"
            }
        },
        "invokedBy": "signin.amazonaws.com"
    },
    "eventTime": "2018-07-09T12:56:54Z",
    "eventSource": "mediastore.amazonaws.com",
    "eventName": "CreateContainer",
    "awsRegion": "ap-northeast-1",
    "sourceIPAddress": "54.239.119.16",
    "userAgent": "signin.amazonaws.com",
    "requestParameters": {
        "containerName": "TestContainer"
    },
    "responseElements": {
        "container": {
            "status": "CREATING",
            "creationTime": "Jul 9, 2018 12:56:54 PM",
            "name": " TestContainer ",
```

```
            "aRN": "arn:aws:mediastore:ap-northeast-1:111122223333:container/
TestContainer"
        }
    },
    "requestID":
 "MNCTGH4HRQJ27GRMBVDPIVHEP4LO2BN6MUVHBCPSHOAWNSOKSXCO24B2UEOBBND5DONRXTMFK3TOJ4G7AHWMESI",
    "eventID": "7085b140-fb2c-409b-a329-f567912d704c",
    "eventType": "AwsApiCall",
    "recipientAccountId": "111122223333"
}
```

Monitoring AWS Elemental MediaStore with Amazon CloudWatch

You can monitor AWS Elemental MediaStore using CloudWatch, which collects raw data and processes it into readable metrics. CloudWatch keeps statistics for 15 months so that you can access historical information and gain a better perspective on how your web application or service is performing. You can also set alarms that watch for certain thresholds, and send notifications or take actions when those thresholds are met. For more information, see the Amazon CloudWatch User Guide.

AWS provides the following monitoring tools to watch MediaStore, report when something is wrong, and take automatic actions when appropriate:

- Amazon CloudWatch Logs allows you to monitor, store, and access your log files from AWS services such as AWS Elemental MediaStore. You can use CloudWatch Logs to monitor applications and systems using log data. For example, CloudWatch Logs can track the number of errors that occur in your application logs and send you a notification whenever the rate of errors exceeds a threshold that you specify. CloudWatch Logs uses your log data for monitoring, so no code changes are required. For example, you can monitor application logs for specific literal terms (such as "ValidationException") or count the number of `PutObject` requests that were made during a certain time period. When the term that you are searching for is found, CloudWatch Logs reports the data to a CloudWatch metric that you specify. Log data is encrypted while in transit and while it is at rest.
- Amazon CloudWatch Events delivers system events that describe changes in AWS resources, such as MediaStore objects. Typically, AWS services deliver event notifications to CloudWatch Events in seconds but can sometimes take a minute or longer. You can set up rules to match events (such as a `DeleteObject` request) and route them to one or more target functions or

streams. CloudWatch Events becomes aware of operational changes as they occur. In addition, CloudWatch Events responds to these operational changes and takes corrective action as necessary, by sending messages to respond to the environment, activating functions, making changes, and capturing state information.

CloudWatch Logs

Access logging provides detailed records for the requests that are made to objects in a container. Access logs are useful for many applications, such as security and access audits. They can also help you learn about your customer base and understand your MediaStore bill. CloudWatch Logs are categorized as follows:

- A log stream is a sequence of log events that share the same source.
- A log group is a group of log streams that share the same retention, monitoring, and access control settings. When you enable access logging on a container, MediaStore creates a log group with a name such as `/aws/mediastore/MyContainerName`. You can define log groups and specify which streams to put into each group. There is no quota on the number of log streams that can belong to one log group.

By default, logs are kept indefinitely and never expire. You can adjust the retention policy for each log group, keeping the indefinite retention, or choosing a retention period from one day to 10 years.

Setting up permissions for Amazon CloudWatch

Use AWS Identity and Access Management (IAM) to create a role that gives AWS Elemental MediaStore access to Amazon CloudWatch. You must perform these steps for CloudWatch Logs to be published for your account. CloudWatch automatically publishes metrics for your account.

To allow MediaStore access to CloudWatch

1. Open the IAM console at https://console.aws.amazon.com/iam/.
2. In the navigation pane of the IAM console, choose **Policies**, and then choose **Create policy**.
3. Choose the **JSON** tab and paste the following policy:

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

```
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "logs:DescribeLogGroups",
                "logs:CreateLogGroup"
            ],
            "Resource": "*"
        },
        {
            "Effect": "Allow",
            "Action": [
                "logs:CreateLogStream",
                "logs:DescribeLogStreams",
                "logs:PutLogEvents"
            ],
            "Resource": "arn:aws:logs:*:*:log-group:/aws/mediastore/*"
        }
    ]
}
```

This policy allows MediaStore to create log groups and log streams for any containers in any Region within your AWS account.

4. Choose **Review policy**.
5. On the **Review policy** page, for **Name**, enter **`MediaStoreAccessLogsPolicy`**, and then choose **Create policy**.
6. In the navigation pane of the IAM console, choose **Roles**, and then choose **Create role**.
7. Choose the **Another AWS account** role type.
8. For **Account ID**, enter your AWS account ID.
9. Choose **Next: Permissions**.
10. In the search box, enter **`MediaStoreAccessLogsPolicy`**.
11. Select the check box next to your new policy, and then choose **Next: Tags**.
12. Choose **Next: Review** to preview your new user.
13. For **Role name**, enter **`MediaStoreAccessLogs`**, and then choose **Create role**.
14. In the confirmation message, choose the name of the role that you just created (**`MediaStoreAccessLogs`**).
15. On the role's **Summary** page, choose the **Trust relationships** tab.

16. Choose **Edit trust relationship**.
17. In the policy document, change the principal to the MediaStore service. It should look like this:

```
"Principal": {
   "Service": "mediastore.amazonaws.com"
},
```

The entire policy should read as follows:

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

18. Choose **Update Trust Policy**.

Enabling access logging for a container

By default, AWS Elemental MediaStore doesn't collect access logs. When you enable access logging on a container, MediaStore delivers access logs for objects stored in that container to Amazon CloudWatch. The access logs provide detailed records for requests that are made to any object stored in the container. 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.

> **Important**
>
> There is no extra charge for enabling access logging on an MediaStore container. However, any log files that the service delivers to you accrues the usual charges for storage. (You can delete the log files at any time.) AWS doesn't assess data transfer charges for log file delivery, but does charge the normal data transfer rate for accessing the log files.

To enable access logging (AWS CLI)

- In the AWS CLI, use the `start-access-logging` command:

```
aws mediastore start-access-logging --container-name LiveEvents --region us-west-2
```

  This command has no return value.

Disabling access logging for a container

When you disable access logging on a container, AWS Elemental MediaStore stops sending access logs to Amazon CloudWatch. These access logs are not saved and are not retrievable.

To disable access logging (AWS CLI)

- In the AWS CLI, use the `stop-access-logging` command:

```
aws mediastore stop-access-logging --container-name LiveEvents --region us-west-2
```

  This command has no return value.

Troubleshooting access logging in AWS Elemental MediaStore

When AWS Elemental MediaStore access logs do not appear in Amazon CloudWatch, refer to the following table for potential causes and resolutions.

> **Note**
>
> Be sure to enable AWS CloudTrail Logs to assist with the troubleshooting process.

Symptom	The Problem Might Be...	Try This...
You don't see any CloudTrail events, even though CloudTrail logs are enabled.	The IAM role either does not exist or it has the incorrect	Create a role with the correct name, permissions, and trust policy. See the section called

Symptom	The Problem Might Be...	Try This...
	name, permissions, or trust policy.	"Setting up permissions for CloudWatch".
You submitted a `DescribeContainer` API request, but the response shows that the `AccessLoggingEnabled` parameter has a value of `False`. In addition, you don't see any CloudTrail events for the `MediaStoreAccessLogs` role making a successful `DescribeLogGroup`, `CreateLogGroup`, `DescribeLogStream`, or `CreateLogStream` call.	The IAM role either does not exist or it has the incorrect name, permissions, or trust policy.	Create a role with the correct name, permissions, and trust policy. See the section called "Setting up permissions for CloudWatch".
	Access logging is not enabled on the container.	Enable access logs for the container. See the section called "Enabling access logging".

Symptom	The Problem Might Be...	Try This...

On the CloudTrail console, you see an event with an access denied error related to the `MediaStoreAccessLogs` role. The CloudTrail event might include lines such as the following:
 	
`"eventSource": "logs.amazonaws.com",`
 	
`"errorCode": "AccessDenied",`
 	
`"errorMessage": "User: arn:aws:sts::111122223333:assumed-role/MediaStoreAccessLogs/MediaStoreAccessLogsSession is not authorized to perform: logs:DescribeLogGroups on resource: arn:aws:logs:us-west-2:111122223333:log-group::log-stream:",`
	The IAM role doesn't have the correct permissions for AWS Elemental MediaStore.	Update the IAM role to have the correct permissions and trust policy. See the section called "Setting up permissions for CloudWatch".

You don't see any logs for an entire container or containers.	Your account might have exceeded the CloudWatch quota for log groups per account per Region. See the quotas for log groups in the Amazon CloudWatch Logs User Guide.	On the CloudWatch console, determine if your account has met the CloudWatch quota for log groups. If necessary, request a quota increase.

Symptom	The Problem Might Be...	Try This...
You see some logs in CloudWatch, but not all logs that you expect to see.	Your account might have exceeded the CloudWatch quota for transactions per second per account per Region. See the quotas for `PutLogEvents` in the Amazon CloudWatch Logs User Guide.	Request a quota increase for CloudWatch transactions per second per account per Region.

Access log format

The access log files consist of a sequence of JSON-formatted log records, where each log record represents one request. The order of the fields within the log can vary. The following is an example log that consists of two log records:

```
{
  "Path": "/FootballMatch/West",
  "Requester": "arn:aws:iam::111122223333:user/maria-garcia",
  "AWSAccountId": "111122223333",
  "RequestID":
 "aaaAAA111bbbBBB222cccCCC333dddDDD444eeeEEE555fffFFF666gggGGG777hhhHHH888iiiIII999jjjJJJ",
  "ContainerName": "LiveEvents",
  "TotalTime": 147,
  "BytesReceived": 1572864,
  "BytesSent": 184,
  "ReceivedTime": "2018-12-13T12:22:06.245Z",
  "Operation": "PutObject",
  "ErrorCode": null,
  "Source": "192.0.2.3",
  "HTTPStatus": 200,
  "TurnAroundTime': 7,
  "ExpiresAt": "2018-12-13T12:22:36Z"
}
```

```
 {
  "Path": "/FootballMatch/West",
  "Requester": "arn:aws:iam::111122223333:user/maria-garcia",
  "AWSAccountId": "111122223333",
  "RequestID":
 "dddDDD444eeeEEE555fffFFF666gggGGG777hhhHHH888iiiIII999jjjJJJ000cccCCC333bbbBBB222aaaAAA",
  "ContainerName": "LiveEvents",
  "TotalTime": 3,
  "BytesReceived": 641354,
  "BytesSent": 163,
  "ReceivedTime": "2018-12-13T12:22:51.779Z",
  "Operation": "PutObject",
  "ErrorCode": "ValidationException",
  "Source": "198.51.100.15",
  "HTTPStatus": 400,
  "TurnAroundTime": 1,
  "ExpiresAt": null
}
```

The following list describes the log record fields:

AWSAccountId

The AWS account ID of the account that was used to make the request.

BytesReceived

The number of bytes in the request body that the MediaStore server receives.

BytesSent

The number of bytes in the response body that the MediaStore server sends. This value often is the same as the value of the `Content-Length` header included with server responses.

ContainerName

The name of the container that received the request.

ErrorCode

The MediaStore error code (such as `InternalServerError`). If no error occurred, the `-` character appears. An error code might appear even if the status code is 200 (indicating a closed connection or an error after the server started streaming the response).

ExpiresAt

The object's expiration date and time. This value is based on the expiration age set by a transient data rule in the lifecycle policy that is applied to the container. The value is `ISO-8601` date time and is based on the system clock of the host that served the request. If the lifecycle policy doesn't have a transient data rule that applies to the object, or if there is no lifecylce policy applied to the container, the value of this field is `null`. This field applies only to the following operations: `PutObject`, `GetObject`, `DescribeObject`, and `DeleteObject`.

HTTPStatus

The numeric HTTP status code of the response.

Operation

The operation that was performed, such as `PutObject` or `ListItems`.

Path

The path within the container where the object is stored. If the operation does not take a path parameter, the `-` character appears.

ReceivedTime

The time of day when the request was received. The value is `ISO-8601` date time and is based on the system clock of the host that served the request.

Requester

The user Amazon Resource Name (ARN) of the account that was used to make the request. For unauthenticated requests, this value is `anonymous`. If the request fails before authentication is complete, this field might be missing from the log. For such requests, the `ErrorCode` might identify the authorization issue.

RequestID

A string that is generated by AWS Elemental MediaStore to uniquely identify each request.

Source

The apparent internet address of the requester or the service principal of the AWS service making the call. If intermediate proxies and firewalls obscure the address of the machine making the request, the value is set to null.

TotalTime

The number of milliseconds (ms) that the request was in flight from the server's perspective. This value is measured beginning with the time that your request is received by the service and ending with the time that the last byte of the response is sent. This value is measured from the server's perspective because measurements made from the client's perspective are affected by network latency.

TurnAroundTime

The number of milliseconds that MediaStore 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.

The order of the fields in the log can vary.

Logging status changes take effect over time

Changes to the logging status of a container take time to actually affect the delivery of log files. For example, if you enable logging for container A, some requests made in the following hour might be logged, while others might not. If you disable logging for container B, some logs for the next hour might continue to be delivered to, while others might not. In all cases, the new settings eventually take effect without any further action on your part.

Best effort server log delivery

Access log records are delivered on a best effort basis. Most requests for a container 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 access 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 access logs is to give you an idea of the nature of traffic against your container. It is rare to lose log records, but access logging is not meant to be a complete accounting of all requests.

It follows from the best-effort nature of the access 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 access log.

Programming considerations for access log format

From time to time, we might extend the access log format by adding new fields. Code that parses access logs must be written to handle additional fields that it does not understand.

CloudWatch Events

Amazon CloudWatch Events enables you to automate your AWS services and respond automatically to system events such as application availability issues or resource changes. You can write simple rules to indicate which events are of interest to you, and what automated actions to take when an event matches a rule.

> **Important**
>
> Typically, AWS services deliver event notifications to CloudWatch Events in seconds but can sometimes take a minute or longer.

When a file is uploaded to a container or removed from a container, two events are fired in succession in the CloudWatch service:

1. the section called “Object state change event”
2. the section called “Container state change event”

For information about subscribing to these events, see Amazon CloudWatch.

The actions that can be automatically triggered include the following:

- Invoking an AWS Lambda function
- Invoking Amazon EC2 Run Command
- Relaying the event to Amazon Kinesis Data Streams
- Activating an AWS Step Functions state machine
- Notifying an Amazon SNS topic or an AWS SMS queue

Some examples of using CloudWatch Events with AWS Elemental MediaStore include the following:

- Activating a Lambda function whenever a container is created

- Notifying an Amazon SNS topic when an object is deleted

For more information, see the Amazon CloudWatch Events User Guide.

Topics

AWS Elemental MediaStore object state change event

This event is published when an object's state has changed (when the object has been uploaded or deleted).

> **Note**
>
> Objects that expire because of a transient data rule do not emit a CloudWatch event when they expire.

For information about subscribing to this event, see Amazon CloudWatch.

Object updated

```
{
  "version": "1",
  "id": "6a7e8feb-b491-4cf7-a9f1-bf3703467718",
  "detail-type": "MediaStore Object State Change",
  "source": "aws.mediastore",
  "account": "111122223333",
  "time": "2017-02-22T18:43:48Z",
  "region": "us-east-1",
  "resources": [
    "arn:aws:mediastore:us-east-1:111122223333:MondayMornings/Episode1/
Introduction.avi"
  ],
  "detail": {
    "ContainerName": "Movies",
    "Operation": "UPDATE",
    "Path":"TVShow/Episode1/Pilot.avi",
    "ObjectSize":123456,
```

```
    "URL": "https://a832p1qeaznlp9.files.mediastore-us-west-2.com/Movies/
MondayMornings/Episode1/Introduction.avi"
  }
}
```

Object removed

```
{
  "version": "1",
  "id": "6a7e8feb-b491-4cf7-a9f1-bf3703467718",
  "detail-type": "MediaStore Object State Change",
  "source": "aws.mediastore",
  "account": "111122223333",
  "time": "2017-02-22T18:43:48Z",
  "region": "us-east-1",
  "resources": [
    "arn:aws:mediastore:us-east-1:111122223333:Movies/MondayMornings/Episode1/
Introduction.avi"
  ],
  "detail": {
    "ContainerName": "Movies",
    "Operation": "REMOVE",
    "Path":"Movies/MondayMornings/Episode1/Introduction.avi",
    "URL": "https://a832p1qeaznlp9.files.mediastore-us-west-2.com/Movies/
MondayMornings/Episode1/Introduction.avi"
  }
}
```

AWS Elemental MediaStore container state change event

This event is published when a container's state has changed (when a container has been added or deleted). For information about subscribing to this event, see Amazon CloudWatch.

Container created

```
{
  "version": "1",
  "id": "6a7e8feb-b491-4cf7-a9f1-bf3703467718",
  "detail-type": "MediaStore Container State Change",
  "source": "aws.mediastore",
  "account": "111122223333",
  "time": "2017-02-22T18:43:48Z",
  "region": "us-east-1",
```

```
  "resources": [
    "arn:aws:mediastore:us-east-1:111122223333:container/Movies"
  ],
  "detail": {
    "ContainerName": "Movies",
    "Operation": "CREATE"
    "Endpoint": "https://a832p1qeaznlp9.mediastore-us-west-2.amazonaws.com"
  }
}
```

Container removed

```
{
  "version": "1",
  "id": "6a7e8feb-b491-4cf7-a9f1-bf3703467718",
  "detail-type": "MediaStore Container State Change",
  "source": "aws.mediastore",
  "account": "111122223333",
  "time": "2017-02-22T18:43:48Z",
  "region": "us-east-1",
  "resources": [
    "arn:aws:mediastore:us-east-1:111122223333:container/Movies"
  ],
  "detail": {
    "ContainerName": "Movies",
    "Operation": "REMOVE"
  }
}
```

Monitoring AWS Elemental MediaStore with Amazon CloudWatch metrics

You can monitor AWS Elemental MediaStore using CloudWatch, which collects raw data and processes it into readable metrics. CloudWatch keeps statistics are kept for 15 months so that you can access historical information and gain a better perspective on how your web application or service is performing. You can also set alarms that watch for certain thresholds, and send notifications or take actions when those thresholds are met. For more information, see the Amazon CloudWatch User Guide.

For AWS Elemental MediaStore, you might want to watch `BytesDownloaded` and send an email to yourself when that metric reaches a certain threshold.

To view metrics using the CloudWatch console

Metrics are grouped first by the service namespace, and then by the various dimension combinations within each namespace.

1. Sign in to the AWS Management Console and open the CloudWatch console at https://console.aws.amazon.com/cloudwatch/.
2. In the navigation pane, choose **Metrics**.
3. Under **All metrics**, choose the **AWS/MediaStore** namespace.
4. Choose the metric dimension to view the metrics. For example, choose `Request metrics by container` to view metrics for the different types of requests that have been sent to the container.

To view metrics using the AWS CLI

- At a command prompt, use the following command:

```
aws cloudwatch list-metrics --namespace "AWS/MediaStore"
```

AWS Elemental MediaStore metrics

The following table lists metrics that AWS Elemental MediaStore sends to CloudWatch.

> **Note**
>
> To view metrics, you must add a metric policy to the container to allow MediaStore to send metrics to Amazon CloudWatch.

Metric	Description
`RequestCount`	The total number of HTTP requests made to a MediaStore container, separated by operation type (`Put`, `Get`, `Delete`, `Describe`, `List`). Units: Count Valid dimensions:

Metric	Description
	• Container name • Object group name • Request type Valid statistics: Sum
4xxErrorCount	The number of HTTP requests made to MediaStore that resulted in a 4xx error. Units: Count Valid dimensions: • Container name • Object group name • Request type Valid statistics: Sum
5xxErrorCount	The number of HTTP requests made to MediaStore that resulted in a 5xx error. Units: Count Valid dimensions: • Container name • Object group name • Request type Valid statistics: Sum

Metric	Description
BytesUploaded	The number of bytes uploaded for requests made to a MediaStore container, where the request includes a body. Units: Bytes Valid dimensions: • Container name • Object group name Valid statistics: Average (bytes per request), Sum (bytes per period), Sample Count, Min (same as P0.0), Max (same as p100), any percentile between p0.0 and p99.9
BytesDownloaded	The number of bytes downloaded for requests made to a MediaStore container, where the response includes a body. Units: Bytes Valid dimensions: • Container name • Object group name Valid statistics: Average (bytes per request), Sum (bytes per period), Sample Count, Min (same as P0.0), Max (same as p100), any percentile between p0.0 and p99.9

Metric	Description
TotalTime	The number of milliseconds that the request was in flight from the server's perspective. This value is measured from the time that MediaStore receives your request, to the time that it sends the last byte of the response. This value is measured from the server's perspective because measurements made from the client's perspective are affected by network latency. Units: Milliseconds Valid dimensions: • Container name • Object group name • Request type Valid statistics: Average, Min (same as P0.0), Max (same as p100), any percentile between p0.0 and p100
TurnaroundTime	The number of milliseconds that MediaStore spent processing your request. This value is measured from the time that MediaStore receives the last byte of your request, to the time that it sends the first byte of the response. Units: Milliseconds Valid dimensions: • Container name • Object group name • Request type Valid statistics: Average, Min (same as P0.0), Max (same as p100), any percentile between p0.0 and p100

Metric	Description
`ThrottleCount`	The number of HTTP requests made to MediaStore that were throttled. Units: Count Valid dimensions: • Container name • Object group name • Request type Valid statistics: Sum

Tagging AWS Elemental MediaStore resources

A *tag* is a custom attribute label that you assign or that AWS assigns to an AWS resource. Each tag has two parts:

- A *tag key* (for example, `CostCenter`, `Environment`, or `Project`). Tag keys are case sensitive.
- An optional field known as a *tag value* (for example, `111122223333` or `Production`). Omitting the tag value is the same as using an empty string. Like tag keys, tag values are case sensitive.

Tags help you do the following:

- Identify and organize your AWS resources. Many AWS services support tagging, so you can assign the same tag to resources from different services to indicate that the resources are related. For example, you could assign the same tag to an AWS Elemental MediaStore *`container`* that you assign to an AWS Elemental MediaLive input.
- Track your AWS costs. You activate these tags on the AWS Billing and Cost Management dashboard. AWS uses the tags to categorize your costs and deliver a monthly cost allocation report to you. For more information, see Use Cost Allocation Tags in the AWS Billing User Guide.

The following sections provide more information about tags for AWS Elemental MediaStore.

Supported resources in AWS Elemental MediaStore

The following resources in AWS Elemental MediaStore support tagging:

- *container*

For information about adding and managing tags, see Managing tags.

AWS Elemental MediaStore doesn't support the tag-based access control feature of AWS Identity and Access Management (IAM).

Tag naming and usage conventions

The following basic naming and usage conventions apply to using tags with AWS Elemental MediaStore resources:

- Each resource can have a maximum of 50 tags.
- For each resource, each tag key must be unique, and each tag key can have only one value.
- The maximum tag key length is 128 Unicode characters in UTF-8.
- The maximum tag value length is 256 Unicode characters in UTF-8.
- Allowed characters are letters, numbers, spaces representable in UTF-8, and the following characters: **. : + = @ _ / -** (hyphen). Amazon EC2 resources allow any characters.
- Tag keys and values are case sensitive. As a best practice, decide on a strategy for capitalizing tags, and consistently implement that strategy across all resource types. For example, decide whether to use `Costcenter`, `costcenter`, or `CostCenter`, and use the same convention for all tags. Avoid using similar tags with inconsistent case treatment.
- The `aws:` prefix is prohibited for tags; it's reserved for AWS use. You can't edit or delete tag keys or values with this prefix. Tags with this prefix do not count against your tags per resource quota.

Managing tags

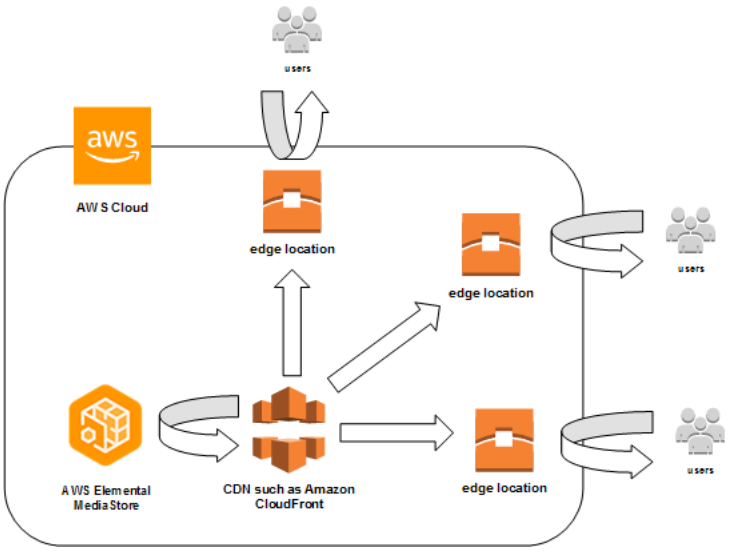
Tags are made up of the `Key` and `Value` properties on a resource. You can use the AWS CLI or the MediaStore API to add, edit, or delete the values for these properties. For information about working with tags, see the following sections in the *AWS Elemental MediaStore API Reference*:

- CreateContainer

- ListTagsForResource
- Resources
- TagResource
- UntagResource

Working with content delivery networks (CDNs)

You can use a content delivery network (CDN) such as Amazon CloudFront to serve the content that you store in AWS Elemental MediaStore. A CDN is a globally distributed set of servers that caches content such as videos. When a user requests your content, the CDN routes the request to the edge location that provides the lowest latency. If your content is already cached in that edge location, the CDN delivers it immediately. If your content is not currently in that edge location, the CDN retrieves it from your origin (such as your MediaStore container) and distributes it to the user.



Topics

- Allowing Amazon CloudFront to access your AWS Elemental MediaStore container
- AWS Elemental MediaStore's interaction with HTTP caches

Allowing Amazon CloudFront to access your AWS Elemental MediaStore container

You can use Amazon CloudFront to serve the content that you store in a container in AWS Elemental MediaStore. You can do so in one of the following ways:

- Using Origin Access Control (OAC) - (Recommended) Use this option if your AWS Region supports the OAC feature of CloudFront.
- Using Shared Secrets - Use this option if your AWS Region does not support the OAC feature of CloudFront.

Using Origin Access Control (OAC)

You can use the Origin Access Control (OAC) feature of Amazon CloudFront to secure AWS Elemental MediaStore origins with improved security. You can enable AWS Signature Version 4 (SigV4) on CloudFront requests for MediaStore origins and set when and if CloudFront should sign the requests. You can access the OAC feature of CloudFront through the console, APIs, SDK, or CLI, and there are no additional fees for its use.

For more information about using the OAC feature with MediaStore, see Restricting access to a MediaStore origin in the Amazon CloudFront Developer Guide.

Using Shared Secrets

If your AWS Region does not support the OAC feature of Amazon CloudFront, you can attach a policy to your AWS Elemental MediaStore container that grants read access or greater to CloudFront.

> **Note**
>
> We recommend using the OAC feature if your AWS Region supports it. The following procedures require you to configure MediaStore and CloudFront with shared secrets in order to restrict access to MediaStore containers. To follow best security practices, this manual configuration requires periodic rotation of secrets. With OAC on MediaStore origins, you can instruct CloudFront to sign requests using SigV4 and forward them to MediaStore for signature matching, eliminating the need to use and rotate secrets. This ensures that requests are automatically verified before media content is served, making the delivery of media content through MediaStore and CloudFront simpler and more secure.

To allow CloudFront to access your container (console)

1. Open the MediaStore console at https://console.aws.amazon.com/mediastore/.
2. On the **Containers** page, choose the container name.

   The container details page appears.
3. In the **Container policy** section, attach a policy that grants read access or greater to Amazon CloudFront.

Example

The following example policy, which is similar to the example policy for Public Read Access over HTTPS, matches these requirements because it allows `GetObject` and `DescribeObject` commands from anyone who submits requests to your domain through HTTPS. Furthermore, the following example policy better secures your workflow because it allows CloudFront access to MediaStore objects only when the request occurs over an HTTPS connection and contains the correct Referer header.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "CloudFrontRead",
      "Effect": "Allow",
      "Principal": "*",
      "Action": [
        "mediastore:GetObject",
        "mediastore:DescribeObject"
      ],
      "Resource": "arn:aws:mediastore:<region>:<owner acct
 number>:container/<container name>/*",
      "Condition": {
        "StringEquals": {
          "aws:Referer": "<secretValue>"
        },
        "Bool": {
          "aws:SecureTransport": "true"
        }
      }
    }
 ]}
```

4. In the **Container CORS policy** section, assign a policy that allows the appropriate access level.

> **Note**
>
> A CORS policy is necessary only if you want to provide access to a browser-based player.

5. Make note of the following details:

- The data endpoint that is assigned to your container. You can find this information in the **Info** section of the **Containers** page. In CloudFront, the data endpoint is referred to as the *origin domain name*.
- The folder structure in the container where the objects are stored. In CloudFront, this is referred to as the *origin path*. Note that this setting is optional. For more information about origin paths, see the Amazon CloudFront Developer Guide.

6. In CloudFront, create a distribution that is configured to serve content from AWS Elemental MediaStore. You will need the information that you collected in the preceding step.

After attaching the policy to your MediaStore containers, you must configure CloudFront to use only HTTPS connections for origin requests, and also add a custom header with the correct secret value.

To configure CloudFront to access your container via an HTTPS connection with a secret value for the Referer header (console)

1. Open the CloudFront console.
2. On the **Origins** page, choose your MediaStore origin.
3. Choose **Edit**.
4. Choose **HTTPS only** for the protocol.
5. In the **Add custom header** section, choose **Add header**.
6. For the **Name**, choose **Referer**. For the **value**, use the same `<secretValue>` string that you used in your container policy.
7. Choose **Save** and let the changes deploy.

AWS Elemental MediaStore's interaction with HTTP caches

AWS Elemental MediaStore stores objects so that they can be cached correctly and efficiently by content delivery networks (CDNs) like Amazon CloudFront. When an end user or CDN retrieves an object from MediaStore, the service returns HTTP headers that affect the caching behavior of the object. (The standards for HTTP 1.1 caching behavior are found in RFC2616 section 13.) These headers are:

- **ETag (not customizable)** – The entity tag header is a unique identifier for the response that MediaStore sends. Standards-compliant CDNs and web browsers use this tag as a key to cache the object with. MediaStore automatically generates an `ETag` for each object when it is uploaded. You can view an object's details to determine its ETag value.
- **`Last-Modified` (not customizable)** – The value of this header indicates the date and time that the object was modified. MediaStore automatically generates this value when the object is uploaded.
- **`Cache-Control` (customizable)** – The value of this header controls how long an object should be cached before the CDN checks to see if it has been modified. You can set this header to any value when you upload an object to a MediaStore container using the CLI or API. The complete set of valid values is described in HTTP/1.1 documentation. If you don't set this value when you upload an object, MediaStore won't return this header when the object is retrieved.

  A common use case for the Cache-Control header is to specify a duration to cache the object. For example, suppose that you have a video manifest file that is being frequently overwritten by an encoder. You could set the `max-age` to 10 to indicate that the object should be cached for only 10 seconds. Or suppose that you have a stored video segment that will never be overwritten. You could set the `max-age` for this object to 31536000 to cache for approximately 1 year.

Conditional requests

Conditional requests to MediaStore

MediaStore responds identically to conditional requests (using request headers such as `If-Modified-Since` and `If-None-Match`, as described in RFC7232) and unconditional requests. This means that when MediaStore receives a valid `GetObject` request, the service always returns the object even if the client already has the object.

Conditional requests to CDNs

CDNs that serve content on behalf of MediaStore can process conditional requests by returning `304 Not Modified`, as described in RFC7232 section 4.1. This indicates that there is no need to transfer the complete object contents, because the requester already has an object that matches the conditional request.

CDNs (and other caches that are compliant with HTTP/1.1) base these decisions on the `ETag` and `Cache-Control` headers that are forwarded by the origin servers. To control how often

CDNs query MediaStore origin servers for updates to repeatedly retrieved objects, set the `Cache-Control` headers for those objects when you upload them to MediaStore.

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 CLI	AWS CLI 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 Tools for PowerShell	AWS Tools for PowerShell 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 SAP ABAP	AWS SDK for SAP ABAP code examples
AWS SDK for Swift	AWS SDK for Swift code examples

For examples specific to this service, see Code examples for MediaStore using AWS SDKs.

Example availability

Can't find what you need? Request a code example by using the **Provide feedback** link at the bottom of this page.

Code examples for MediaStore using AWS SDKs

The following code examples show how to use MediaStore with an AWS software development kit (SDK).

Actions are code excerpts from larger programs and must be run in context. While actions show you how to call individual service functions, you can see actions in context in their related scenarios.

For a complete list of AWS SDK developer guides and code examples, see Using this service with an AWS SDK. This topic also includes information about getting started and details about previous SDK versions.

Code examples

- Basic examples for MediaStore using AWS SDKs
  - Actions for MediaStore using AWS SDKs
    - Use CreateContainer with an AWS SDK or CLI
    - Use DeleteContainer with an AWS SDK or CLI
    - Use DeleteObject with an AWS SDK
    - Use DescribeContainer with an AWS SDK or CLI
    - Use GetObject with an AWS SDK or CLI
    - Use ListContainers with an AWS SDK or CLI
    - Use PutObject with an AWS SDK or CLI

Basic examples for MediaStore using AWS SDKs

The following code examples show how to use the basics of AWS Elemental MediaStore with AWS SDKs.

Examples

- Actions for MediaStore using AWS SDKs
  - Use CreateContainer with an AWS SDK or CLI
  - Use DeleteContainer with an AWS SDK or CLI
  - Use DeleteObject with an AWS SDK
  - Use DescribeContainer with an AWS SDK or CLI

- Use GetObject with an AWS SDK or CLI
- Use ListContainers with an AWS SDK or CLI
- Use PutObject with an AWS SDK or CLI

Actions for MediaStore using AWS SDKs

The following code examples demonstrate how to perform individual MediaStore actions with AWS SDKs. Each example includes a link to GitHub, where you can find instructions for setting up and running the code.

The following examples include only the most commonly used actions. For a complete list, see the AWS Elemental MediaStore API Reference.

Examples

- Use CreateContainer with an AWS SDK or CLI
- Use DeleteContainer with an AWS SDK or CLI
- Use DeleteObject with an AWS SDK
- Use DescribeContainer with an AWS SDK or CLI
- Use GetObject with an AWS SDK or CLI
- Use ListContainers with an AWS SDK or CLI
- Use PutObject with an AWS SDK or CLI

Use `CreateContainer` with an AWS SDK or CLI

The following code examples show how to use `CreateContainer`.

CLI

AWS CLI

To create a container

The following `create-container` example creates a new, empty container.

```
aws mediastore create-container --container-name ExampleContainer
```

Output:

```
{
    "Container": {
        "AccessLoggingEnabled": false,
        "CreationTime": 1563557265,
        "Name": "ExampleContainer",
        "Status": "CREATING",
        "ARN": "arn:aws:mediastore:us-west-2:111122223333:container/
ExampleContainer"
    }
}
```

For more information, see Creating a Container in the *AWS Elemental MediaStore User Guide*.

- For API details, see CreateContainer in *AWS CLI Command Reference*.

Java

SDK for Java 2.x

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
import software.amazon.awssdk.services.mediastore.MediaStoreClient;
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.mediastore.model.CreateContainerRequest;
import software.amazon.awssdk.services.mediastore.model.CreateContainerResponse;
import software.amazon.awssdk.services.mediastore.model.MediaStoreException;

/**
 * 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 CreateContainer {
```

```
   public static long sleepTime = 10;

   public static void main(String[] args) {
       final String usage = """

               Usage:    <containerName>

               Where:
                  containerName - The name of the container to create.
               """;

       if (args.length != 1) {
           System.out.println(usage);
           System.exit(1);
       }

       String containerName = args[0];
       Region region = Region.US_EAST_1;
       MediaStoreClient mediaStoreClient = MediaStoreClient.builder()
               .region(region)
               .build();

       createMediaContainer(mediaStoreClient, containerName);
       mediaStoreClient.close();
   }


   public static void createMediaContainer(MediaStoreClient mediaStoreClient,
String containerName) {
       try {
           CreateContainerRequest containerRequest =
CreateContainerRequest.builder()
                   .containerName(containerName)
                   .build();

           CreateContainerResponse containerResponse =
mediaStoreClient.createContainer(containerRequest);
           String status = containerResponse.container().status().toString();
           while (!status.equalsIgnoreCase("Active")) {
               status = DescribeContainer.checkContainer(mediaStoreClient,
containerName);
               System.out.println("Status - " + status);
               Thread.sleep(sleepTime * 1000);
           }
```

```
            System.out.println("The container ARN value is " +
 containerResponse.container().arn());
            System.out.println("Finished ");

        } catch (MediaStoreException | InterruptedException e) {
            System.err.println(e.getMessage());
            System.exit(1);
        }
    }
}
```

- For API details, see CreateContainer 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. This topic also includes information about getting started and details about previous SDK versions.

Use `DeleteContainer` with an AWS SDK or CLI

The following code examples show how to use `DeleteContainer`.

CLI

AWS CLI

To delete a container

The following `delete-container` example deletes the specified container. You can delete a container only if it has no objects.

```
aws mediastore delete-container \
    --container-name=ExampleLiveDemo
```

This command produces no output.

For more information, see Deleting a Container in the *AWS Elemental MediaStore User Guide*.

- For API details, see DeleteContainer in *AWS CLI Command Reference*.

Java

SDK for Java 2.x

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
import software.amazon.awssdk.services.mediastore.MediaStoreClient;
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.mediastore.model.CreateContainerRequest;
import software.amazon.awssdk.services.mediastore.model.CreateContainerResponse;
import software.amazon.awssdk.services.mediastore.model.MediaStoreException;

/**
 * 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 CreateContainer {
    public static long sleepTime = 10;

    public static void main(String[] args) {
        final String usage = """

                Usage:    <containerName>

                Where:
                   containerName - The name of the container to create.
                """;

        if (args.length != 1) {
            System.out.println(usage);
            System.exit(1);
        }
```

```
        String containerName = args[0];
        Region region = Region.US_EAST_1;
        MediaStoreClient mediaStoreClient = MediaStoreClient.builder()
                .region(region)
                .build();

        createMediaContainer(mediaStoreClient, containerName);
        mediaStoreClient.close();
    }


    public static void createMediaContainer(MediaStoreClient mediaStoreClient,
 String containerName) {
        try {
            CreateContainerRequest containerRequest =
 CreateContainerRequest.builder()
                    .containerName(containerName)
                    .build();

            CreateContainerResponse containerResponse =
 mediaStoreClient.createContainer(containerRequest);
            String status = containerResponse.container().status().toString();
            while (!status.equalsIgnoreCase("Active")) {
                status = DescribeContainer.checkContainer(mediaStoreClient,
 containerName);
                System.out.println("Status - " + status);
                Thread.sleep(sleepTime * 1000);
            }

            System.out.println("The container ARN value is " +
 containerResponse.container().arn());
            System.out.println("Finished ");

        } catch (MediaStoreException | InterruptedException e) {
            System.err.println(e.getMessage());
            System.exit(1);
        }
    }
}
```

- For API details, see DeleteContainer 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. This topic also includes information about getting started and details about previous SDK versions.

Use `DeleteObject` with an AWS SDK

The following code example shows how to use `DeleteObject`.

Java

SDK for Java 2.x

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.mediastore.MediaStoreClient;
import software.amazon.awssdk.services.mediastore.model.DescribeContainerRequest;
import
 software.amazon.awssdk.services.mediastore.model.DescribeContainerResponse;
import software.amazon.awssdk.services.mediastoredata.MediaStoreDataClient;
import software.amazon.awssdk.services.mediastoredata.model.DeleteObjectRequest;
import
 software.amazon.awssdk.services.mediastoredata.model.MediaStoreDataException;
import java.net.URI;
import java.net.URISyntaxException;

/**
 * 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 DeleteObject {
    public static void main(String[] args) throws URISyntaxException {
        final String usage = """
```

```
                Usage:    <completePath> <containerName>

                Where:
                   completePath - The path (including the container) of the item
to delete.
                   containerName - The name of the container.
                """;

        if (args.length != 2) {
            System.out.println(usage);
            System.exit(1);
        }

        String completePath = args[0];
        String containerName = args[1];
        Region region = Region.US_EAST_1;
        URI uri = new URI(getEndpoint(containerName));

        MediaStoreDataClient mediaStoreData = MediaStoreDataClient.builder()
                .endpointOverride(uri)
                .region(region)
                .build();

        deleteMediaObject(mediaStoreData, completePath);
        mediaStoreData.close();
    }

    public static void deleteMediaObject(MediaStoreDataClient mediaStoreData,
String completePath) {
        try {
            DeleteObjectRequest deleteObjectRequest =
DeleteObjectRequest.builder()
                    .path(completePath)
                    .build();

            mediaStoreData.deleteObject(deleteObjectRequest);

        } catch (MediaStoreDataException e) {
            System.err.println(e.awsErrorDetails().errorMessage());
            System.exit(1);
        }
    }
```

```
    private static String getEndpoint(String containerName) {
        Region region = Region.US_EAST_1;
        MediaStoreClient mediaStoreClient = MediaStoreClient.builder()
                .region(region)
                .build();

        DescribeContainerRequest containerRequest =
 DescribeContainerRequest.builder()
                .containerName(containerName)
                .build();

        DescribeContainerResponse response =
 mediaStoreClient.describeContainer(containerRequest);
        mediaStoreClient.close();
        return response.container().endpoint();
    }
}
```

- For API details, see DeleteObject 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. This topic also includes information about getting started and details about previous SDK versions.

Use `DescribeContainer` with an AWS SDK or CLI

The following code examples show how to use `DescribeContainer`.

CLI

AWS CLI

To view the details of a container

The following `describe-container` example displays the details of the specified container.

```
aws mediastore describe-container \
    --container-name ExampleContainer
```

Output:

```
{
    "Container": {
        "CreationTime": 1563558086,
        "AccessLoggingEnabled": false,
        "ARN": "arn:aws:mediastore:us-west-2:111122223333:container/
ExampleContainer",
        "Status": "ACTIVE",
        "Name": "ExampleContainer",
        "Endpoint": "https://aaabbbcccdddee.data.mediastore.us-
west-2.amazonaws.com"
    }
}
```

For more information, see Viewing the Details for a Container in the *AWS Elemental MediaStore User Guide*.

- For API details, see DescribeContainer in *AWS CLI Command Reference*.

Java

SDK for Java 2.x

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.mediastore.MediaStoreClient;
import software.amazon.awssdk.services.mediastore.model.DescribeContainerRequest;
import
 software.amazon.awssdk.services.mediastore.model.DescribeContainerResponse;
import software.amazon.awssdk.services.mediastore.model.MediaStoreException;

/**
 * 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 DescribeContainer {

    public static void main(String[] args) {
        final String usage = """

                Usage:    <containerName>

                Where:
                   containerName - The name of the container to describe.
                """;

        if (args.length != 1) {
            System.out.println(usage);
            System.exit(1);
        }

        String containerName = args[0];
        Region region = Region.US_EAST_1;
        MediaStoreClient mediaStoreClient = MediaStoreClient.builder()
                .region(region)
                .build();

        System.out.println("Status is " + checkContainer(mediaStoreClient,
 containerName));
        mediaStoreClient.close();
    }

    public static String checkContainer(MediaStoreClient mediaStoreClient, String
 containerName) {
        try {
            DescribeContainerRequest describeContainerRequest =
 DescribeContainerRequest.builder()
                    .containerName(containerName)
                    .build();

            DescribeContainerResponse containerResponse =
 mediaStoreClient.describeContainer(describeContainerRequest);
            System.out.println("The container name is " +
 containerResponse.container().name());
            System.out.println("The container ARN is " +
 containerResponse.container().arn());
```

```
            return containerResponse.container().status().toString();

        } catch (MediaStoreException e) {
            System.err.println(e.awsErrorDetails().errorMessage());
            System.exit(1);
        }
        return "";
    }
}
```

- For API details, see DescribeContainer 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. This topic also includes information about getting started and details about previous SDK versions.

Use `GetObject` with an AWS SDK or CLI

The following code examples show how to use `GetObject`.

CLI

AWS CLI

To download an object

The following `get-object` example download an object to the specified endpoint.

```
aws mediastore-data get-object \
    --endpoint https://aaabbbcccdddee.data.mediastore.us-west-2.amazonaws.com \
    --path=/folder_name/README.md README.md
```

Output:

```
{
    "ContentLength": "2307346",
    "ContentType": "image/jpeg",
    "LastModified": "Fri, 19 Jul 2019 21:32:20 GMT",
    "ETag": "2aa333bbcc8d8d22d777e999c88d4aa9eeeeee4dd89ff7f555555555555da6d3",
    "StatusCode": 200
}
```

To download part of an object

The following `get-object` example downloads a portion an object to the specified endpoint.

```
aws mediastore-data get-object \
    --endpoint https://aaabbbcccdddee.data.mediastore.us-west-2.amazonaws.com \
    --path /folder_name/README.md \
    --range="bytes=0-100" README2.md
```

Output:

```
{
    "StatusCode": 206,
    "ContentRange": "bytes 0-100/2307346",
    "ContentLength": "101",
    "LastModified": "Fri, 19 Jul 2019 21:32:20 GMT",
    "ContentType": "image/jpeg",
    "ETag": "2aa333bbcc8d8d22d777e999c88d4aa9eeeeee4dd89ff7f555555555555da6d3"
}
```

For more information, see Downloading an Object in the *AWS Elemental MediaStore User Guide*.

- For API details, see GetObject in *AWS CLI Command Reference*.

Java

SDK for Java 2.x

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
import software.amazon.awssdk.core.ResponseInputStream;
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.mediastore.MediaStoreClient;
import software.amazon.awssdk.services.mediastore.model.DescribeContainerRequest;
```

```
import
 software.amazon.awssdk.services.mediastore.model.DescribeContainerResponse;
import software.amazon.awssdk.services.mediastoredata.MediaStoreDataClient;
import software.amazon.awssdk.services.mediastoredata.model.GetObjectRequest;
import software.amazon.awssdk.services.mediastoredata.model.GetObjectResponse;
import
 software.amazon.awssdk.services.mediastoredata.model.MediaStoreDataException;
import java.io.File;
import java.io.FileOutputStream;
import java.io.IOException;
import java.io.OutputStream;
import java.net.URI;
import java.net.URISyntaxException;

/**
 * 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 GetObject {
    public static void main(String[] args) throws URISyntaxException {
        final String usage = """

                Usage:    <completePath> <containerName> <savePath>

                Where:
                   completePath - The path of the object in the container (for
 example, Videos5/sampleVideo.mp4).
                   containerName - The name of the container.
                   savePath - The path on the local drive where the file is
 saved, including the file name (for example, C:/AWS/myvid.mp4).
                """;

        if (args.length != 3) {
            System.out.println(usage);
            System.exit(1);
        }

        String completePath = args[0];
        String containerName = args[1];
```

```
        String savePath = args[2];

        Region region = Region.US_EAST_1;
        URI uri = new URI(getEndpoint(containerName));
        MediaStoreDataClient mediaStoreData = MediaStoreDataClient.builder()
                .endpointOverride(uri)
                .region(region)
                .build();

        getMediaObject(mediaStoreData, completePath, savePath);
        mediaStoreData.close();
    }

    public static void getMediaObject(MediaStoreDataClient mediaStoreData, String
completePath, String savePath) {

        try {
            GetObjectRequest objectRequest = GetObjectRequest.builder()
                    .path(completePath)
                    .build();

            // Write out the data to a file.
            ResponseInputStream<GetObjectResponse> data =
mediaStoreData.getObject(objectRequest);
            byte[] buffer = new byte[data.available()];
            data.read(buffer);

            File targetFile = new File(savePath);
            OutputStream outStream = new FileOutputStream(targetFile);
            outStream.write(buffer);
            System.out.println("The data was written to " + savePath);

        } catch (MediaStoreDataException | IOException e) {
            System.err.println(e.getMessage());
            System.exit(1);
        }
    }

    private static String getEndpoint(String containerName) {
        Region region = Region.US_EAST_1;
        MediaStoreClient mediaStoreClient = MediaStoreClient.builder()
                .region(region)
                .build();
```

```
        DescribeContainerRequest containerRequest =
 DescribeContainerRequest.builder()
                .containerName(containerName)
                .build();

        DescribeContainerResponse response =
 mediaStoreClient.describeContainer(containerRequest);
        return response.container().endpoint();
    }
}
```

- For API details, see GetObject 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. This topic also includes information about getting started and details about previous SDK versions.

Use `ListContainers` with an AWS SDK or CLI

The following code examples show how to use `ListContainers`.

CLI

AWS CLI

To view a list of containers

The following `list-containers` example displays a list of all containers that are associated with your account.

```
aws mediastore list-containers
```

Output:

```
{
    "Containers": [
        {
            "CreationTime": 1505317931,
            "Endpoint": "https://aaabbbcccdddee.data.mediastore.us-
west-2.amazonaws.com",
```

```
            "Status": "ACTIVE",
            "ARN": "arn:aws:mediastore:us-west-2:111122223333:container/
ExampleLiveDemo",
            "AccessLoggingEnabled": false,
            "Name": "ExampleLiveDemo"
        },
        {
            "CreationTime": 1506528818,
            "Endpoint": "https://fffgggghhhiiijj.data.mediastore.us-
west-2.amazonaws.com",
            "Status": "ACTIVE",
            "ARN": "arn:aws:mediastore:us-west-2:111122223333:container/
ExampleContainer",
            "AccessLoggingEnabled": false,
            "Name": "ExampleContainer"
        }
    ]
}
```

For more information, see Viewing a List of Containers in the *AWS Elemental MediaStore User Guide*.

- For API details, see ListContainers in *AWS CLI Command Reference*.

Java

SDK for Java 2.x

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
import software.amazon.awssdk.auth.credentials.ProfileCredentialsProvider;
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.mediastore.MediaStoreClient;
import software.amazon.awssdk.services.mediastore.model.Container;
import software.amazon.awssdk.services.mediastore.model.ListContainersResponse;
import software.amazon.awssdk.services.mediastore.model.MediaStoreException;
import java.util.List;
```

```
/**
 * 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 ListContainers {

    public static void main(String[] args) {

        Region region = Region.US_EAST_1;
        MediaStoreClient mediaStoreClient = MediaStoreClient.builder()
                .region(region)
                .build();

        listAllContainers(mediaStoreClient);
        mediaStoreClient.close();
    }

    public static void listAllContainers(MediaStoreClient mediaStoreClient) {
        try {
            ListContainersResponse containersResponse =
 mediaStoreClient.listContainers();
            List<Container> containers = containersResponse.containers();
            for (Container container : containers) {
                System.out.println("Container name is " + container.name());
            }

        } catch (MediaStoreException e) {
            System.err.println(e.awsErrorDetails().errorMessage());
            System.exit(1);
        }
    }
}
```

- For API details, see ListContainers 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. This topic also includes information about getting started and details about previous SDK versions.

Use `PutObject` with an AWS SDK or CLI

The following code examples show how to use `PutObject`.

CLI

AWS CLI

To upload an object

The following `put-object` example uploads an object to the specified container. You can specify a folder path where the object will be saved within the container. If the folder already exists, AWS Elemental MediaStore stores the object in the folder. If the folder doesn't exist, the service creates it, and then stores the object in the folder.

```
aws mediastore-data put-object \
    --endpoint https://aaabbbcccdddee.data.mediastore.us-west-2.amazonaws.com \
    --body README.md \
    --path /folder_name/README.md \
    --cache-control "max-age=6, public" \
    --content-type binary/octet-stream
```

Output:

```
{
    "ContentSHA256":
 "74b5fdb517f423ed750ef214c44adfe2be36e37d861eafe9c842cbe1bf387a9d",
    "StorageClass": "TEMPORAL",
    "ETag": "af3e4731af032167a106015d1f2fe934e68b32ed1aa297a9e325f5c64979277b"
}
```

For more information, see Uploading an Object in the *AWS Elemental MediaStore User Guide*.

- For API details, see PutObject in *AWS CLI Command Reference*.

Java

SDK for Java 2.x

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.mediastore.MediaStoreClient;
import software.amazon.awssdk.services.mediastoredata.MediaStoreDataClient;
import software.amazon.awssdk.core.sync.RequestBody;
import software.amazon.awssdk.services.mediastoredata.model.PutObjectRequest;
import
 software.amazon.awssdk.services.mediastoredata.model.MediaStoreDataException;
import software.amazon.awssdk.services.mediastoredata.model.PutObjectResponse;
import software.amazon.awssdk.services.mediastore.model.DescribeContainerRequest;
import
 software.amazon.awssdk.services.mediastore.model.DescribeContainerResponse;
import java.io.File;
import java.net.URI;
import java.net.URISyntaxException;

/**
 * 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 PutObject {
    public static void main(String[] args) throws URISyntaxException {
        final String USAGE = """

                To run this example, supply the name of a container, a file
 location to use, and path in the container\s

                Ex: <containerName> <filePath> <completePath>
```

```
                """;

        if (args.length < 3) {
            System.out.println(USAGE);
            System.exit(1);
        }

        String containerName = args[0];
        String filePath = args[1];
        String completePath = args[2];

        Region region = Region.US_EAST_1;
        URI uri = new URI(getEndpoint(containerName));
        MediaStoreDataClient mediaStoreData = MediaStoreDataClient.builder()
                .endpointOverride(uri)
                .region(region)
                .build();

        putMediaObject(mediaStoreData, filePath, completePath);
        mediaStoreData.close();
    }

    public static void putMediaObject(MediaStoreDataClient mediaStoreData, String
filePath, String completePath) {
        try {
            File myFile = new File(filePath);
            RequestBody requestBody = RequestBody.fromFile(myFile);

            PutObjectRequest objectRequest = PutObjectRequest.builder()
                    .path(completePath)
                    .contentType("video/mp4")
                    .build();

            PutObjectResponse response = mediaStoreData.putObject(objectRequest,
requestBody);
            System.out.println("The saved object is " +
response.storageClass().toString());

        } catch (MediaStoreDataException e) {
            System.err.println(e.awsErrorDetails().errorMessage());
            System.exit(1);
        }
    }
```

```
    public static String getEndpoint(String containerName) {

        Region region = Region.US_EAST_1;
        MediaStoreClient mediaStoreClient = MediaStoreClient.builder()
                .region(region)
                .build();

        DescribeContainerRequest containerRequest =
 DescribeContainerRequest.builder()
                .containerName(containerName)
                .build();

        DescribeContainerResponse response =
 mediaStoreClient.describeContainer(containerRequest);
        return response.container().endpoint();
    }
}
```

- For API details, see PutObject 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. This topic also includes information about getting started and details about previous SDK versions.

Quotas in AWS Elemental MediaStore

The Service Quotas console provides information about AWS Elemental MediaStore quotas. Along with viewing the default quotas, you can use the Service Quotas console to request quota increases for adjustable quotas.

The following table describes quotas, formerly referred to as *limits*, in AWS Elemental MediaStore. Quotas are the maximum number of service resources or operations for your AWS account.

> **Note**
>
> To assign quotas to individual containers within your account, contact AWS Support or your account manager. This option can help you divide up the account-level limits among your containers, to prevent one container from using up your entire quota.

Resource or Operation	Default Quota	Comments
Containers	100	The maximum number of containers that you can create in this account.
Folder Levels	10	The maximum number of folder levels that you can create in a container. You can create as many folders as you want, as long as they are not nested more than 10 levels within a container.
Folders	Unlimited	You can create as many folders as you want, as long as they are not nested more than 10 levels within a container.
Object Size	25 MB	The maximum file size of a single object.
Objects	Unlimited	You can upload as many objects as you want to a folder or container in your account.

Resource or Operation	Default Quota	Comments
Rate of DeleteObject API requests	100	The maximum number of operation requests that you can make per second. Additional requests are throttled. You can request a quota increase.
Rate of DescribeObject API requests	1,000	The maximum number of operation requests that you can make per second. Additional requests are throttled. You can request a quota increase.
Rate of GetObject API requests for standard upload availability	1,000	The maximum number of operation requests that you can make per second. Additional requests are throttled. You can request a quota increase.
Rate of GetObject API requests for streaming upload availability	25	The maximum number of operation requests that you can make per second. Additional requests are throttled. You can request a quota increase.
Rate of ListItems API requests	5	The maximum number of operation requests that you can make per second. Additional requests are throttled. You can request a quota increase.

Resource or Operation	Default Quota	Comments
Rate of PutObject API requests for chunked transfer encoding (also known as streaming upload availability)	10	The maximum number of operation requests that you can make per second. Additional requests are throttled. You can request a quota increase. In the request, specify the requested TPS and average object size.
Rate of PutObject API requests for standard upload availability	100	The maximum number of operation requests that you can make per second. Additional requests are throttled. You can request a quota increase. In the request, specify the requested TPS and average object size.
Rules in a Metric Policy	10	The maximum number of rules that you can include in a metric policy.
Rules in an Object Lifecycle Policy	10	The maximum number of rules that you can include in an object lifecycle policy.

AWS Elemental MediaStore related information

The following table lists related resources that you'll find useful as you work with AWS Elemental MediaStore.

- Classes & Workshops – Links to role-based and specialty courses, in addition to self-paced labs to help sharpen your AWS skills and gain practical experience.
- AWS Developer Center – Explore tutorials, download tools, and learn about AWS developer events.
- AWS Developer Tools – Links to developer tools, SDKs, IDE toolkits, and command line tools for developing and managing AWS applications.
- Getting Started Resource Center – Learn how to set up your AWS account, join the AWS community, and launch your first application.
- Hands-On Tutorials – Follow step-by-step tutorials to launch your first application on AWS.
- AWS Whitepapers – Links to a comprehensive list of technical AWS whitepapers, covering topics such as architecture, security, and economics and authored by AWS Solutions Architects or other technical experts.
- AWS Support Center – The hub for creating and managing your AWS Support cases. Also includes links to other helpful resources, such as forums, technical FAQs, service health status, and AWS Trusted Advisor.
- Support – The primary webpage for information about Support, a one-on-one, fast-response support channel to help you build and run applications in the cloud.
- Contact Us – A central contact point for inquiries concerning AWS billing, account, events, abuse, and other issues.
- AWS Site Terms – Detailed information about our copyright and trademark; your account, license, and site access; and other topics.

Document history for user guide

The following table describes the documentation for this release of AWS Elemental MediaStore. For notification about updates to this documentation, you can subscribe to an RSS feed.

Change	Description	Date
End of support notice	End of support notice: On November 13, 2025, AWS will discontinue support for AWS Elemental MediaStore. After November 13, 2025, you will no longer be able to access the MediaStore console or MediaStore resources. For more information, visit this blog post.	November 12, 2024
Origin Access Control (OAC) Improvement	Added information about how to use OAC with AWS Elemental MediaStore.	April 17, 2023
Quotas updates	Corrected quota value and description for `Rules in a Metric Policy`.	October 25, 2022
ExpiresAt field	Access logs now include an `ExpiresAt` field that indicates the object's expiration date and time based on transient data rules in the container's lifecycle policy.	July 16, 2020
Lifecycle transition rules	You can now add a lifecycle transition rule to your object lifecycle policy that sets objects to be moved to the	April 20, 2020

	infrequent access (IA) storage class after they reach a certain age.	
Empty container	You can now delete all objects within a container at once.	April 7, 2020
Support for Amazon CloudWatch metrics	You can set a metric policy to dictate which metrics MediaStore sends to CloudWatch.	March 30, 2020
Wildcards in delete object rules	In an object lifecycle policy, you can now use a wildcard in a delete object rule. This allows you to specify files based on their filename or extension that you want the service to delete after a certain number of days.	December 20, 2019
Object lifecycle policies	You can now add a rule to your object lifecycle policy that indicates an expiration by age in seconds.	September 13, 2019
AWS CloudFormation support	You can now use an AWS CloudFormation template to create a container automatic ally. The AWS CloudForm ation template manages data for five API actions: creating a container, setting access logging, updating the default container policy, adding a cross-origin resource sharing (CORS) policy, and adding an object lifecycle policy.	May 17, 2019

Quotas for streaming upload availability	For objects with streaming upload availability (chunked transfer of objects), the `PutObject` operation can't exceed 10 TPS and the `GetObject` operation can't exceed 25 TPS.	April 8, 2019
Chunked transfer of objects	Added support for chunked transfer of objects. This capability allows you to specify that an object is available for downloading before the object is uploaded completely.	April 5, 2019
Access logging	AWS Elemental MediaStore now supports access logging, which provides detailed records for the requests that are made to objects in a container.	February 25, 2019
Object lifecycle policies	Added support for object lifecycle policies, which govern the expiration date of objects within the current container.	December 12, 2018
Increased object size quota	The quota for an object's size is now 25 MB.	October 10, 2018
Increased object size quota	The quota for an object's size is now 20 MB.	September 6, 2018

AWS CloudTrail integration	The CloudTrail integration content has been updated to align with recent changes to the CloudTrail service.	July 12, 2018
CDN collaboration	Added information about how to use AWS Elemental MediaStore with a content delivery network (CDN) such as Amazon CloudFront.	April 14, 2018
CORS configurations	AWS Elemental MediaStor e now supports cross-ori gin resource sharing (CORS), which allows client web applications that are loaded in one domain to interact with resources in a different domain.	February 7, 2018
New service and guide	This is the initial release of the video origination and storage service, AWS Elemental MediaStore, and the *AWS Elemental MediaStor e User Guide*.	November 27, 2017

Note

- The AWS Media Services are not designed or intended for use with applications or in situations requiring fail-safe performance, such as life safety operations, navigation or communication systems, air traffic control, or life support machines in which the unavailability, interruption or failure of the services could lead to death, personal injury, property damage or environmental damage.

AWS Glossary

For the latest AWS terminology, see the AWS glossary in the *AWS Glossary Reference*.