# Table of Contents

What is MediaStore? .......................................................................................................................... 1  
Concepts and terminology .................................................................................................................... 1  
Related services ................................................................................................................................. 2  
Accessing MediaStore ......................................................................................................................... 2  
Pricing ................................................................................................................................................. 3  
Regions and endpoints .......................................................................................................................... 3  
Setting up ......................................................................................................................................... 4  
Signing up for AWS ............................................................................................................................... 4  
Creating an admin IAM user ................................................................................................................ 4  
Creating a non-admin IAM user ........................................................................................................... 5  
  Step 1: Create user groups .............................................................................................................. 5  
  Step 2: Create users ....................................................................................................................... 6  
Getting started ................................................................................................................................... 7  
  Step 1: Access AWS Elemental MediaStore ................................................................................ 7  
  Step 2: Create a container ............................................................................................................. 7  
  Step 3: Upload an object ............................................................................................................... 7  
  Step 4: Access an object .............................................................................................................. 8  
Containers ........................................................................................................................................... 9  
  Rules for container names ............................................................................................................ 9  
  Creating a container .................................................................................................................... 9  
  Viewing container details .......................................................................................................... 10  
  Viewing a list of containers ....................................................................................................... 11  
  Deleting a container ............................................................................................................... 11  
Policies ............................................................................................................................................. 13  
  Container policies ....................................................................................................................... 13  
    Viewing a container policy ........................................................................................................ 13  
    Editing a container policy ...................................................................................................... 14  
    Example container policies ................................................................................................. 15  
  CORS policies ............................................................................................................................ 19  
    Use-case scenarios .................................................................................................................. 20  
    Adding a CORS policy ........................................................................................................... 20  
    Viewing a CORS policy .......................................................................................................... 21  
    Editing a CORS policy .......................................................................................................... 22  
    Deleting a CORS policy ....................................................................................................... 23  
    Troubleshooting .................................................................................................................... 23  
    Example CORS policies ...................................................................................................... 24  
  Object lifecycle policies ............................................................................................................... 25  
    Components of an object lifecycle policy ............................................................................ 25  
    Adding an object lifecycle policy ......................................................................................... 29  
    Viewing an object lifecycle policy ....................................................................................... 30  
    Editing an object lifecycle policy ......................................................................................... 31  
    Deleting an object lifecycle policy ....................................................................................... 32  
    Example object lifecycle policies ..................................................................................... 32  
  Metric policies ........................................................................................................................... 35  
    Adding a metric policy ....................................................................................................... 36  
    Viewing a metric policy ...................................................................................................... 36  
    Editing a metric policy ....................................................................................................... 36  
    Example metric policies .................................................................................................... 39  
Folders ............................................................................................................................................... 42  
  Rules for folder names ........................................................................................................... 42  
  Creating a folder ................................................................................................................... 43  
  Deleting a folder ................................................................................................................... 43  
Objects ........................................................................................................................................... 44  
  Uploading an object ............................................................................................................. 44
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 (p. 1)
- Related services (p. 2)
- Accessing AWS Elemental MediaStore (p. 2)
- Pricing for AWS Elemental MediaStore (p. 3)
- Regions and endpoints for AWS Elemental MediaStore (p. 3)

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 (p. 4).

- **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:
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

Before you start using AWS Elemental MediaStore, you must sign up for AWS (if you don't already have an AWS account) and create IAM users and roles to allow access to MediaStore. This includes creating an IAM role for yourself.

**Topics**
- Signing up for AWS (p. 4)
- Creating an admin IAM user (p. 4)
- Creating a non-admin IAM user (p. 5)

**Signing up for AWS**

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

**To sign up for an AWS account**

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

**Creating an admin IAM user**

When you first create an AWS account, you begin with a single 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 do not use the root user for your everyday tasks, even the administrative ones. Instead, adhere to the best practice of using the root user only to create your first IAM user. Then securely lock away the root user credentials and use them to perform only a few account and service management tasks.

In this procedure, you use the AWS account root user to create your first IAM user. You add this IAM user to an Administrators group, to ensure that you have access to all services and their resources in your account. The next time that you access your AWS account, you should sign in with the credentials for this IAM user.

To create an IAM user with limited permissions, see the section called "Creating a non-admin IAM user" (p. 5).

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

1. Sign in to the IAM console as the account owner by choosing Root user and entering your AWS account email address. On the next page, enter your password.
Creating a non-admin IAM user

Users in the Administrators group for an account have access to all AWS services and resources in that account. This section describes how to create users with permissions that are limited to AWS Elemental MediaStore.

Topics
- Step 1: Create user groups (p. 5)
- Step 2: Create users (p. 6)

Step 1: Create user groups

You can create a user group for each AWS Elemental MediaStore policy and assign users to a group rather than attaching individual policies to each user. Using the following procedure, create
two user groups: one for the AWS Elemental MediaStoreFullAccess policy and one for the AWS Elemental MediaStoreReadOnly policy.

**Note**
AWS Elemental MediaStoreFullAccess and AWS Elemental MediaStoreReadOnly are AWS managed policies.

**To create user groups**

1. In the navigation pane of the IAM console, choose **Groups**, and then choose **Create New Group**.
2. On the **Groups** page, choose **Create New Group**, and then create an administrator group using the AWS Elemental MediaStoreFullAccess policy.
   a. On the **Set Group Name** page, enter a name for the group such as **MediaStoreAdmins**.
   b. Choose **Next Step**.
   c. On the **Attach Policy** page, for **Filter**, choose **AWS Managed**, and then enter **mediastore**.
   d. In the policy list, choose the **AWS Elemental MediaStoreFullAccess** policy.
   e. Choose **Next Step**.
   f. On the **Review** page, choose **Create Group**.
3. On the **Groups** page, choose **Create New Group**, and then create a read-only group using the AWS Elemental MediaStoreReadOnly policy.
   a. On the **Set Group Name** page, enter a name for the group such as **MediaStoreReaders**.
   b. Choose **Next Step**.
   c. On the **Attach Policy** page, for **Filter**, choose **AWS Managed**, and then enter **mediastore**.
   d. In the policy list, choose the **AWS Elemental MediaStoreReadOnly** policy.
   e. Choose **Next Step**.
   f. On the **Review** page, choose **Create Group**.

**Step 2: Create users**

Create IAM users for the individuals who require access to AWS Elemental MediaStore, and add each user to the appropriate user group to ensure that they have the right level of permissions.

**To create users**

1. In the navigation pane of the IAM console, choose **Users**, and then choose **Add user**.
2. For **User name**, enter the name that the user will use to sign in to MediaStore.
3. Select the check box next to **AWS Management Console access**, select **Custom password**, and then enter the new user's password in the box. You can optionally select **Require password reset** to force the user to create a password the next time the user signs in.
4. Choose **Next: Permissions**.
5. On the **Set permissions for user** page, choose **Add user to group**.
6. In the group list, choose the group with the appropriate attached policy. Remember that permissions levels are as follows:
   - The **MediaStoreAdmins** group has permissions that allow all actions on all resources in MediaStore.
   - The **MediaStoreReaders** group has permissions that allow read-only rights for all resources in MediaStore.
7. Choose **Next: Review** to see the list of group memberships that will be added to the new user.
8. When you are ready to proceed, choose **Create user**.
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 (p. 7)
• Step 2: Create a container (p. 7)
• Step 3: Upload an object (p. 7)
• Step 4: Access an object  (p. 8)

Step 1: Access AWS Elemental MediaStore

Once you have set up your AWS account and created IAM 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 (p. 4).

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 (p. 9).
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 (p. 9)
  • Creating a container (p. 9)
  • Viewing the details for a container (p. 10)
  • Viewing a list of containers (p. 11)
  • Deleting a container (p. 11)

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)

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 (p. 9).

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:

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

The following example shows the return value:

```json
code
{
    "Container": {
        "AccessLoggingEnabled": false,
        "CreationTime": 1563557265.0,
        "Name": "ExampleContainer",
        "Status": "CREATING",
    }
}
```

### 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)

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 (p. 13).

#### To view the details for a container (AWS CLI)

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

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

The following example shows the return value:

```json
code
{
    "Container": {
        "CreationTime": 1563558086.0,
        "AccessLoggingEnabled": false,
        "Status": "ACTIVE",
        "Name": "ExampleContainer",
        "Endpoint": "https://aaabbbcccddee.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.

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

The following example shows the return value:

```
{
   "Containers": [
   {
      "CreationTime": 1505317931.0, 
      "Endpoint": "https://aaabbbccccdde.data.mediastore.us-west-2.amazonaws.com",
      "Status": "ACTIVE",
      "AccessLoggingEnabled": false,
      "Name": "ExampleLiveDemo"
   },
   {
      "CreationTime": 1506528818.0, 
      "Endpoint": "https://fffgghhhiiijj.data.mediastore.us-west-2.amazonaws.com",
      "Status": "ACTIVE",
      "AccessLoggingEnabled": false,
      "Name": "ExampleContainer"
   }
   ]
}
```

Deleting a container

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

**To delete a container (console)**

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:

```bash
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 (p. 13)** - 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 (p. 19)** - 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 (p. 35)** - Allows MediaStore to send metrics to Amazon CloudWatch. MediaStore does not set a default metric policy.

- **Object lifecycle policy (p. 25)** - 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 (p. 25) 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 (p. 13)
- Editing a container policy (p. 14)
- Example container policies (p. 15)

**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)**

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:

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

The following example shows the return value:

```json
{
  "Policy": {
    "Version": "2012-10-17",
    "Statement": [
      {
        "Sid": "PublicReadOverHttps",
        "Effect": "Allow",
        "Principal": {
          "AWS": "arn:aws:iam::111122223333:root",
        },
        "Action": [
          "mediastore:GetObject",
          "mediastore:DescribeObject",
        ],
        "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)

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” (p. 15).
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:

```json
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "PublicReadOverHttps",
      "Effect": "Allow",
    }
  ]
}
```
Example container policies

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

Topics
- Example container policy: Default (p. 15)
- Example container policy: Public read access over HTTPS (p. 16)
- Example container policy: Public read access over HTTP or HTTPS (p. 16)
- Example container policy: Cross-account read access—HTTP enabled (p. 17)
- Example container policy: Cross-account read access over HTTPS (p. 17)
- Example container policy: Cross-account read access to a role (p. 17)
- Example container policy: Cross-account full access to a role (p. 18)
- Example container policy: Access restricted to specific IP addresses (p. 19)

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" }
      }
    }
  ]
}
```
Example container policies

The policy is built into the service, so you don’t have to create it. However, you can edit the policy (p. 14) 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:

```json
{
  "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):

```json
{
  "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 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>:

```json
{
  "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).

```json
{
  "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"
        }
      }
    }
  ]
}
```
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.

```json
{
    "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 (p. 20)
- Adding a CORS policy to a container (p. 20)
- Viewing a CORS policy (p. 21)
- Editing a CORS policy (p. 22)
- Deleting a CORS policy (p. 23)
- Troubleshooting CORS issues (p. 23)
- Example CORS policies (p. 24)

**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 (p. 13) (that govern access rights to the container) continue to apply.

**To add a CORS policy (console)**

2. On the **Containers** page, choose the name of the container that you want to create a CORS policy for.
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:

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

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

```bash
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)**

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:

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

The following example shows the return value:

```json
{
    "CorsPolicy": [
```
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)

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:

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

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

   ```bash
   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)

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" (p. 21).
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 (p. 24)
- Example CORS policy: Read access for a specific domain (p. 24)

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.

```json
{
    "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 the originating domain. The results are cached for 3,000 seconds before a new set of results is delivered.

```json
{
    "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 (p. 25).

Topics
• Components of an object lifecycle policy (p. 25)
• Adding an object lifecycle policy to a container (p. 29)
• Viewing an object lifecycle policy (p. 30)
• Editing an object lifecycle policy (p. 31)
• Deleting an object lifecycle policy (p. 32)
• Example object lifecycle policies (p. 32)

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 (p. 25)
• Delete object (p. 26)
• Lifecycle transition (p. 27)

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:

```json
{
    "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.
Components of an object lifecycle policy

- **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:

```json
{
   "rules": [
   {
      "definition": {
         "path": [ { "prefix": "Football/" }, { "prefix": "Baseball/" } ],
         "days_since_create": [ { "numeric": [">=", 28] } ],
         "action": "EXPIRE"
      },
      "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 (p. 32).

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 (p. 25).

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)

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:

   ```json
   {
   "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:

   ```bash
   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)

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:

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

  The following example shows the return value:

  ```json
  {
    "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)**

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:

   ```json
   {
     "rules": [
     {
       "definition": {
         "path": [
         ....
       }
     }
   }
   ```
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)

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 (p. 33)
- Example object lifecycle policy: Expire within days (p. 33)
- Example object lifecycle policy: Transition to infrequent access storage class (p. 34)
- Example object lifecycle policy: Multiple rules (p. 34)
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].

```json
{
  "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

```json
{
  "rules": [  
    {
      "definition": {  
        "path": [  
          {"prefix": "AwardsShow/"}  
        ],
        "days_since_create": [  
          {"numeric": [">" , 30]}  
        ]
      },
      "action": "ARCHIVE"
    },  
    {
      "definition": {  
        "path": [  
          {"wildcard": "Football/*.m3u8"}  
        ],
        "seconds_since_create": [  
          {"numeric": [">" , 20]}  
        ]
      }
    }
  ]
}
```
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 (p. 79).

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 (p. 36)
• Viewing a metric policy (p. 36)
• Editing a metric policy (p. 36)
• Example metric policies (p. 39)

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 (p. 39).

To add a metric policy (console)

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)

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

Topics

- Example metric policy: Container-level metrics (p. 39)
- Example metric policy: Path-level metrics (p. 39)
- Example metric policy: Container-level and path-level metrics (p. 40)
- Example metric policy: Path-level metrics using wildcards (p. 40)
- Example metric policy: Path-level metrics with overlapping rules (p. 41)

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.

```json
{
  "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`.

```json
{
  "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.

```json
{
    "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.

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

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

Topics

- Example metric policy: Container-level metrics (p. 39)
- Example metric policy: Path-level metrics (p. 39)
- Example metric policy: Container-level and path-level metrics (p. 40)
- Example metric policy: Path-level metrics using wildcards (p. 40)
- Example metric policy: Path-level metrics with overlapping rules (p. 41)

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

```json
{
    "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.

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” (p. 44).

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 (p. 49).
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 (p. 29) to automatically delete objects within a container after a specified duration.

MediaStore accepts all file types.

**Topics**

- Uploading an object (p. 44)
- Viewing a list of objects (p. 45)
- Viewing the details of an object (p. 47)
- Downloading an object (p. 47)
- Deleting objects (p. 48)

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 (p. 42).

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)

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.

   ```bash
   ``

   The following example shows the return value:

   ```json
   { "ContentSHA256": "74b5fd5b43ef423ed750ef214c44adfe2be36e37d861eafe9c842cbe1bf387a9d"", "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)

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)

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:

  ```bash
  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:

  ```json
  {
    "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:

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

  The following example shows the return value:

  ```json
  {
    "Items": [
      {
        "Type": "FOLDER",
        "Name": "folder_1"
      },
      {
        "LastModified": 1563571940.861,
        "ContentLength": 2307346,
      }
    ]
  }
  ```
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 (p. 84).

To view the details of an object (console)

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:

```bash
```

The following example shows the return value:

```json
{
  "ContentType": "image/jpeg",
  "LastModified": "Fri, 19 Jul 2019 21:32:20 GMT",
  "ContentLength": "2307346",
  "ETag": "2aa333bbcc8d8d22d777e999c88d4aa9e3e3e4dd89ff7f5555555555da6d3"
}
```

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)

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:

  ```
  ```

  The following example shows the return value:

  ```
  {
  "ContentLength": "2307346",
  "ContentType": "image/jpeg",
  "LastModified": "Fri, 19 Jul 2019 21:32:20 GMT",
  "ETag": "2aa33bbcc8d8d2d777e999c88d4aa9e0ee4dd89ff7f555555555555da6d3",
  "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": "2aa33bbcc8d8d2d777e999c88d4aa9e0ee4dd89ff7f555555555555da6d3"
  }
  ```

Deleting objects

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

- **Delete an individual object** (p. 49). No charges apply.
- **Empty a container** (p. 49) to delete all objects within a container at once. Because this process uses API calls, normal API charges apply.
- **Add an object lifecycle policy** (p. 29) 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 (p. 29) to automatically delete objects after they reach a certain age in a container, or you can empty a container (p. 49) 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)**

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:**

  ```bash
  aws mediastore-data --region us-west-2 delete-object --endpoint=https://aaabbbccccdde.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 (p. 35) to automatically delete objects after they reach a certain age in a container, or you can delete objects individually (p. 49).

**To empty a container (console)**

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 a data center and network architecture that is 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 (p. 50)
- Identity and access management in AWS Elemental MediaStore (p. 51)
- Logging and monitoring in AWS Elemental MediaStore (p. 64)
- Compliance validation for AWS Elemental MediaStore (p. 65)
- Resilience in AWS Elemental MediaStore (p. 65)
- Infrastructure security in AWS Elemental MediaStore (p. 66)

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. This content includes 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 user accounts with 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 recommend TLS 1.2 or later.
• Set up API and user activity logging with AWS CloudTrail.
• 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 personal data that is stored in Amazon S3.
• If you require FIPS 140-2 validated cryptographic modules when accessing AWS through a command line interface or an API, use a FIPS endpoint. For more information about the available FIPS endpoints, see Federal Information Processing Standard (FIPS) 140-2.

We strongly recommend that you never put confidential or sensitive information, such as your customers' email addresses, into tags or free-form 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 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.

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” (p. 13).
• 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” (p. 19).

Identity and access management in 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.

This section provides background and additional information about the setup procedures that you follow to use MediaStore. See Setting up (p. 4).

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 (p. 63).

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 employees 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 (p. 55).

**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 AWS Elemental MediaStore identity-based policy examples (p. 58).

## Authenticating with identities

Authentication is how you sign in to AWS using your identity credentials. For more information about signing in using the AWS Management Console, see Signing in to the AWS Management Console as an IAM user or root user in the IAM User Guide.

You must be authenticated (signed in to AWS) as the AWS account root user, an IAM user, or by assuming an IAM role. You can also use your company's single sign-on authentication or even sign in using Google or Facebook. In these cases, your administrator previously set up identity federation using IAM roles. When you access AWS using credentials from another company, you are assuming a role indirectly.

To sign in directly to the AWS Management Console, use your password with your root user email address or your IAM user name. You can access AWS programmatically using your root user or IAM users access keys. AWS provides SDK and command line tools to cryptographically sign your request using your credentials. If you don't use AWS tools, you must sign the request yourself. Do this using Signature Version 4, a protocol for authenticating inbound API requests. For more information about authenticating requests, see Signature Version 4 signing process in the AWS General Reference.

Regardless of the authentication method that you use, you might also 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 Using multi-factor authentication (MFA) in AWS in the IAM User Guide.

### AWS account root user

When you first create an AWS account, you begin with a single 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 do not use the root user for your everyday tasks, even the administrative ones. Instead, adhere to the best practice of using the root user only to create your first IAM user. Then securely lock away the root user credentials and use them to perform only a few account and service management tasks.

### IAM users and groups

An IAM user is an identity within your AWS account that has specific permissions for a single person or application. An IAM user can have long-term credentials such as a user name and password or a set of access keys. To learn how to generate access keys, see Managing access keys for IAM users in the IAM User Guide. When you generate access keys for an IAM user, make sure you view and securely save the key pair. You cannot recover the secret access key in the future. Instead, you must generate a new access key pair.

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 When to create an IAM user (instead of a role) 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. You can temporarily assume an IAM role in the AWS Management Console by switching roles. 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 Using IAM roles in the IAM User Guide.

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

- **Temporary IAM user permissions** – An IAM user can assume an IAM role to temporarily take on different permissions for a specific task.

- **Federated user access** – Instead of creating an IAM user, you can use existing identities from AWS Directory Service, your enterprise user directory, or a web identity provider. These are known as federated users. AWS assigns a role to a federated user when access is requested through an identity provider. For more information about federated users, see Federated users and roles in the IAM User Guide.

- **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 How IAM roles differ from resource-based policies 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.

- **Principal permissions** – When you use an IAM user or role to perform actions in AWS, you are considered a principal. Policies grant permissions to a principal. When you use some services, you might perform an action that then triggers another action in a different service. In this case, you must have permissions to perform both actions. To see whether an action requires additional dependent actions in a policy, see Actions, Resources, and Condition Keys for AWS Elemental MediaStore in the Service Authorization Reference.

- **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 Creating 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 IAM 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 Using an IAM role to grant permissions to applications running on Amazon EC2 instances in the IAM User Guide.

To learn whether to use IAM roles or IAM users, see When to create an IAM role (instead of a user) in the IAM User Guide.

Managing access using policies

You control access in AWS by creating policies and attaching them to IAM identities or AWS resources. A policy is an object in AWS that, when associated with an identity or resource, defines their permissions.
You can sign in as the root user or an IAM user, or you can assume an IAM role. When you then make a request, AWS evaluates the related identity-based or resource-based policies. 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.

Every IAM entity (user or role) starts with no permissions. In other words, by default, users can do nothing, not even change their own password. To give a user permission to do something, an administrator must attach a permissions policy to a user. Or the administrator can add the user to a group that has the intended permissions. When an administrator gives permissions to a group, all users in that group are granted those permissions.

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 Creating IAM 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 Choosing 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 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 How SCPs work 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*.

### Learn more

For more information about identity and access management for MediaStore, continue to the following pages:

- How MediaStore works with IAM (p. 55)
- Identity-based policy examples (p. 58)
- Resource-based policy examples (p. 61)
- Troubleshooting (p. 63)

### How AWS Elemental MediaStore works with IAM

Before you use IAM to manage access to MediaStore, you should understand what IAM features are available to use with MediaStore. To get a high-level view of how MediaStore and other AWS services work with IAM, see AWS Services That Work with IAM in the *IAM User Guide*.

#### MediaStore identity-based policies

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. MediaStore supports specific actions, resources, and condition keys. To learn about all of the elements that you use in a JSON policy, see IAM JSON Policy Elements Reference in the *IAM User Guide*.

#### Actions

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.

Policy actions in MediaStore use the following prefix before the action: `mediastore:`. For example, to grant someone permission to create a new container with the MediaStore `CreateContainer` API operation, you include the `mediastore:CreateContainer` action in their policy. Policy statements must include either an `Action` or `NotAction` element. MediaStore defines its own set of actions that describe tasks that you can perform with this service.

To specify multiple actions in a single statement, separate them with commas as follows:

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

You can specify multiple actions using wildcards (*). For example, to specify all actions that begin with the word `Describe`, include the following action:

```
"Action": "mediastore:Describe*"
```

To see a list of MediaStore actions, see Actions Defined by AWS Elemental MediaStore in the IAM User Guide.

**Resources**

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`.

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"
```

To specify all instances that belong to a specific account, use the wildcard (*). This is required to call ListContainers:

```
```
To see a list of MediaStore resource types and their ARNs, see Resources Defined by AWS Elemental MediaStore in the IAM User Guide. To learn with which actions you can specify the ARN of each resource, see Actions Defined by AWS Elemental MediaStore.

Condition keys

MediaStore does not provide any service-specific condition keys, but it does support using some global condition keys. To see all AWS global condition keys, see AWS Global Condition Context Keys in the IAM User Guide.

Examples

For examples of MediaStore identity-based policies, see AWS Elemental MediaStore identity-based policy examples (p. 58).

MediaStore resource-based policies

Resource-based policies are JSON policy documents that specify what actions a specified principal can perform on the MediaStore resource and under what conditions. MediaStore supports resource-based permissions policies for MediaStore containers. Resource-based policies let you grant usage permission to other accounts on a per-resource basis. You can also use a resource-based policy to allow an AWS service to access your MediaStore containers.

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, you must also grant the principal entity permission to access the resource. 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 How IAM Roles Differ from Resource-based Policies 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 (p. 13).

Examples

For examples of MediaStore resource-based policies, see the section called “Resource-based policy examples” (p. 61).

Authorization based on MediaStore tags

You can attach tags to MediaStore resources or pass tags in a request to MediaStore. To control access based on tags, you provide tag information in the condition element of a policy using the mediatstore:ResourceTag/key-name, aws:RequestTag/key-name, or aws:TagKeys condition keys. You can only tag MediaStore resources using the API. For more information about how to assign a tag to a resource, see TagResource in the AWS Elemental MediaStore API Reference.

To view an example identity-based policy for limiting access to a resource based on the tags on that resource, see Allow or deny actions based on tags on MediaStore resources (p. 62).

MediaStore IAM roles

An IAM role is an entity within your AWS account that has specific permissions.
Using temporary credentials with MediaStore

You can use temporary credentials to sign in with federation, assume an IAM role, or to assume a cross-account role. You obtain temporary security credentials by calling AWS STS API operations such as AssumeRole or GetFederationToken.

MediaStore supports using temporary credentials.

Service-linked roles

Service-linked roles allow AWS services to access resources in other services to complete an action on your behalf. Service-linked roles appear in your IAM account and are owned by the service. An IAM administrator can view but not edit the permissions for service-linked roles.

MediaStore does not support service-linked roles.

Service roles

This feature allows a service to assume a service role on your behalf. This role allows the service to access resources in other services to complete an action on your behalf. Service roles appear in your IAM account and are owned by the account. This means that an IAM administrator can change the permissions for this role. However, doing so might break the functionality of the service.

MediaStore supports service roles.

AWS Elemental MediaStore identity-based policy examples

By default, IAM users and roles don’t have permission to create or modify MediaStore resources. They also can’t perform tasks using the AWS Management Console, AWS CLI, or an AWS API. An IAM administrator must create IAM policies that grant users and roles permission to perform specific API operations on the specified resources that they need. The administrator must then attach those policies to IAM users or groups that require those permissions.

To learn how to create an IAM identity-based policy using these example JSON policy documents, see Creating Policies on the JSON Tab in the IAM User Guide.

Policy best practices

Identity-based policies are very powerful. They 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 using AWS managed policies** – To start using MediaStore quickly, use AWS managed policies to give your employees the permissions they need. These policies are already available in your account and are maintained and updated by AWS. For more information, see Get started using permissions with AWS managed policies in the IAM User Guide.

- **Grant least privilege** – When you create custom policies, grant only the permissions required to perform a task. Start with a minimum set of permissions and grant additional permissions as necessary. Doing so is more secure than starting with permissions that are too lenient and then trying to tighten them later. For more information, see Grant least privilege in the IAM User Guide.

- **Enable MFA for sensitive operations** – For extra security, require IAM users to use multi-factor authentication (MFA) to access sensitive resources or API operations. For more information, see Using multi-factor authentication (MFA) in AWS in the IAM User Guide.

- **Use policy conditions for extra security** – To the extent that it’s practical, define the conditions under which your identity-based policies allow access to a resource. For example, you can write conditions to
specify a range of allowable IP addresses that a request must come from. You can also write conditions to allow requests only within a specified date or time range, or to require the use of SSL or MFA. For more information, see IAM JSON policy elements: Condition 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 (IAM users or roles) with that policy.

To ensure that those entities can still use the MediaStore console, also attach the following AWS managed policy to the entities. For more information, see Adding Permissions to a User in the IAM User Guide:

```json
{
"Version": "2012-10-17",
"Statement": [
  {
    "Action": [
      "mediastore:*"
    ],
    "Effect": "Allow",
    "Resource": "*",
    "Condition": {
      "Bool": {
        "aws:SecureTransport": "true"
      }
    }
  }
]
}
```

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

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.

```json
{
"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": "*"
}
}

## Accessing a MediaStore container

In this example, you want to grant an IAM user in your AWS account access to one of your MediaStore containers, *AwardsShow*. You also want to allow the user to manage and view objects.

The policy has three statements:

- **ListContainersInConsole** grants permission to view a list of all containers in this account.
- **ReadContainerMetadata** grants permission to view metadata associated with the *AwardsShow* container. This includes policies assigned to the container that govern access to the container as well as the lifecycle of objects stored within the container.
- **ManageContainerContents** grants permission to manage and view objects stored in the *AwardsShow* container.

```json
{
  "Version":"2012-10-17",
  "Statement": [
    { "Sid":"ListContainersInConsole",
      "Effect":"Allow",
      "Action": [ "mediastore:ListContainers" ],
      "Resource": "*" },
    { "Sid":"ReadContainerMetadata",
      "Effect":"Allow",
      "Action": [ "mediastore:DescribeContainer",
                   "mediastore:GetContainerPolicy",
                   "mediastore:GetCorsPolicy",
                   "mediastore:GetLifecyclePolicy" ],
      "Resource": "arn:aws:mediastore:*:111122223333:container/AwardsShow" },
    { "Sid":"ManageContainerContents",
      "Effect":"Allow",
      "Action": [ "mediastore:ListItems",
                   "mediastore:GetObject",
                   "mediastore:PutObject",
                   "mediastore:DescribeObject" ],
      "Resource": "*" }
  ]
}
```
AWS Elemental MediaStore resource-based policy examples

To access the AWS Elemental MediaStore console, you must have a minimum set of permissions that allows you to list and view details about the MediaStore resources in your AWS account. The IAM policies in this section show examples of policies that allow specific actions on resources in AWS Elemental MediaStore.

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 (p. 13).

Allow read access to all MediaStore resources

To access the AWS Elemental MediaStore console, you must have a policy that defines which actions you are allowed to take on MediaStore resources in your AWS account. The IAM policy below provides the following permissions. The section for the mediastore:List* and mediastore:Describe* actions allows read-only access to all resources that you create in MediaStore.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Action": [
        "mediastore:List*",
        "mediastore:Describe*
      ],
      "Effect": "Allow",
      "Resource": "arn:aws:mediastore:::*:*
    }
  ]
}
```

Allow all actions on all MediaStore resources

Every user of MediaStore must have a policy that defines permissions on MediaStore resources. The IAM policy below provides the following permissions:

- The section for the mediastore:* action allows all actions on all resources that you create in MediaStore.
- The section for the iam:PassRole action grants permission to pass the MediaStoreAccessLogs IAM role to MediaStore. This allows the service to assume the role later and publish CloudWatch Logs into your account. This portion of the policy is required to enable access logging.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Action": [
        "mediastore:*",
        "iam:PassRole"
      ],
      "Effect": "Allow",
      "Resource": "arn:aws:mediastore:::*:*
    }
  ]
}
```
Resource-based policy examples

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Action": [
        "mediastore:*"
      ],
      "Effect": "Allow",
      "Resource": "arn:aws:mediastore:*:*:*"
    },
    {
      "Action": [
        "iam:PassRole"
      ],
      "Effect": "Allow",
      "Resource": "arn:aws:iam:*:*:role/MediaStoreAccessLogs"
    }
  ]
}
```

Allow or deny actions based on tags on MediaStore resources

To allow access based on a resource’s tags, use a tag-based access policy.

**Note**
Tag-based access control works with only the data types listed in the AWS Elemental MediaStore section of the AWS Elemental MediaStore API Reference. Tag-based access control doesn’t apply for MediaStore data plane APIs.

The IAM policy below provides the following permissions:

- Allow `DeleteContainer` actions on all resources that are tagged with the key `company` and value `ITW`
- Allow `DeleteContainer` actions on all resources that are tagged with the key `environment` and value `ITW-prod`

```
{
  "Version": "2012-10-17",
  "Statement": {
    "Sid": "AllowDeleteForITW",
    "Effect": "Allow",
    "Action": [
      "mediastore:DeleteContainer"
    ],
    "Resource": "arn:aws:mediastore:*:*:container/*",
    "Condition": {
      "StringEquals": {
        "aws:ResourceTag/company": "ITW",
        "aws:ResourceTag/environment": "ITW-prod"
      }
    }
  }
}
```

To create a policy that denies access in these circumstances, change the permission line in the policy to the following:

```
"Effect": "Deny",
```
**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.

**I am not authorized to perform an action in MediaStore**

If the AWS Management Console tells you that you're not authorized to perform an action, contact your administrator for assistance. Your administrator is the person who provided you with your user name and password.

The following example error occurs when the mateojackson IAM user tries to use the console to view details about a container but doesn't have mediastore: **GetContainer** permissions:

```
User: arn:aws:iam::123456789012:user/mateojackson is not authorized to perform: mediastore:GetContainer on resource: exampleContainer
```

In this case, Mateo asks his administrator to update his policies to allow him to access the **exampleContainer** resource using the **mediastore:GetContainer** action.

**I am not authorized to perform iam:PassRole**

If you receive an error that you're not authorized to perform the **iam:PassRole** action, then you must contact your administrator for assistance. Your administrator is the person that provided you with your user name and password. Ask that person to update your policies 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 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 asks her administrator to update her policies to allow her to perform the **iam:PassRole** action.

**I want to view my access keys**

After you create your IAM user access keys, you can view your access key ID at any time. However, you can't view your secret access key again. If you lose your secret key, you must create a new access key pair.

Access keys consist of two parts: an access key ID (for example, AKIAILOSPODDNN7EXAMPLE) and a secret access key (for example, wJalrXUtEnFEMI/K7MDENG/bF7xRfIcYEXAMPLEKEY). Like a user name and password, you must use both the access key ID and secret access key together to authenticate your requests. Manage your access keys as securely as you do your user name and password.

**Important**

Do not provide your access keys to a third party, even to help find your canonical user ID. By doing this, you might give someone permanent access to your account.

When you create an access key pair, you are prompted to save the access key ID and secret access key in a secure location. The secret access key is available only at the time you create it. If you lose your secret
access key, you must add new access keys to your IAM user. You can have a maximum of two access keys. If you already have two, you must delete one key pair before creating a new one. To view instructions, see Managing access keys in the IAM User Guide.

I'm an administrator and want to allow others to access MediaStore

To allow others to access MediaStore, you must create an IAM entity (user or role) for the person or application that needs access. They will use the credentials for that entity to access AWS. You must then attach a policy to the entity that grants them the correct permissions in MediaStore.

To get started right away, see Creating your first IAM delegated user and group in the IAM User Guide.

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 (p. 55).
- 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 How IAM roles differ from resource-based policies 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 (p. 67).

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 (p. 69).

**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 (p. 67).

**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**

AWS Elemental MediaStore is not in scope of any AWS compliance programs.

For a list of AWS services in scope of specific compliance programs, see AWS Services in Scope by Compliance Program. 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 MediaStore is determined by the sensitivity of your data, your company's compliance objectives, and applicable laws and regulations. AWS provides resources to help:

- **Security and Compliance Quick Start Guides** – These deployment guides discuss architectural considerations and provide steps for deploying security- and compliance-focused baseline environments on AWS.
- **Architecting for HIPAA Security and Compliance Whitepaper** – This whitepaper describes how companies can use AWS to create HIPAA-compliant applications.
- **AWS Compliance Resources** – This collection of workbooks and guides might apply to your industry and location.
- **AWS Config** – This AWS 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 that helps you check your compliance with security industry standards and best practices.

**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 Availability 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.

Infrastructure security in AWS Elemental MediaStore

As a managed service, AWS Elemental MediaStore is protected by the AWS global network security procedures that are described in the Amazon Web Services: Overview of Security Processes whitepaper.

You use AWS published API calls to access MediaStore through the network. Clients must support Transport Layer Security (TLS) 1.0 or later. We recommend TLS 1.2 or later. Clients must also support cipher suites with perfect forward secrecy (PFS) such as Ephemeral Diffie-Hellman (DHE) or Elliptic Curve Ephemeral Diffie-Hellman (ECDHE). 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.
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 near real-time stream of system events that describe changes in AWS resources. 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 (p. 67)
- Monitoring AWS Elemental MediaStore with Amazon CloudWatch (p. 69)
- Tagging AWS Elemental MediaStore resources (p. 81)

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 (p. 68)
- Example: AWS Elemental MediaStore log file entries (p. 69)

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 or IAM user credentials
- Whether the request was made with temporary security credentials for a role or federated user
- Whether the request was made by another AWS service

For more information, see the CloudTrail userIdentity Element.
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:

```json
{
  "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": "2018-07-09 12:56:54 PM",
      "name": "TestContainer",
      "aRN": "arn:aws:mediastore:ap-northeast-1:111122223333:container/TestContainer"
    }
  },
  "requestID": "MNCTGH4HRQJ27GRMBVDP1VHEF4L02BN6MVUHCP6OHVSAWNSOKXCO24B2UEOBBDN5DONRXTMFK3TQJ4G7AHWRESI",
  "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, near real-time metrics. These 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.

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. 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:

```json
{
    "Version": "2012-10-17",
    "Statement": [
```
This policy allows MediaStore to create log groups and log streams for any containers in any Region within your AWS account.

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.
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.
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 a 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:

```bash
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:

```bash
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.

<table>
<thead>
<tr>
<th>Symptom</th>
<th>The Problem Might Be...</th>
<th>Try This...</th>
</tr>
</thead>
<tbody>
<tr>
<td>You don't see any CloudTrail events, even though CloudTrail logs are enabled.</td>
<td>The IAM role either does not exist or it has the incorrect name,</td>
<td>Create a role with the correct name, permissions, and trust policy. See the section called</td>
</tr>
</tbody>
</table>
### CloudWatch Logs

<table>
<thead>
<tr>
<th>Symptom</th>
<th>The Problem Might Be...</th>
<th>Try This...</th>
</tr>
</thead>
<tbody>
<tr>
<td>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.</td>
<td>The IAM role either does not exist or it has the incorrect name, permissions, or trust policy.</td>
<td>Create a role with the correct name, permissions, and trust policy. See the section called “Setting up permissions for CloudWatch” (p. 70).</td>
</tr>
<tr>
<td>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:</td>
<td>Access logging is not enabled on the container.</td>
<td>Enable access logs for the container. See the section called “Enabling access logging” (p. 72).</td>
</tr>
<tr>
<td>You don't see any logs for an entire container or containers.</td>
<td>The IAM role doesn't have the correct permissions for AWS Elemental MediaStore.</td>
<td>Update the IAM role to have the correct permissions and trust policy. See the section called “Setting up permissions for CloudWatch” (p. 70).</td>
</tr>
<tr>
<td>You see some logs in CloudWatch, but not all logs that you expect to see.</td>
<td>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.</td>
<td>Request a quota increase for CloudWatch transactions per second per account per Region.</td>
</tr>
</tbody>
</table>
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": "aaaAAA111bbbBBB222cccCCC333ddddd444eeeee555fffFFFF666gggGGG777hhhhHHHH888iiiiIII999jjjjJJJJ",
  "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": "ddddDDDD444eeeee555fffFFFF666gggGGG777hhhhHHHH888iiiiIII999jjjjJJJJ000cccCCC333bbbbbBBB222aaaAAA",
  "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 lifecycle 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. Events from AWS services are delivered to CloudWatch Events in near real time. 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.

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” (p. 77)
2. the section called “Container state change event” (p. 78)

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 (p. 77)
- AWS Elemental MediaStore container state change event (p. 78)

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

```json
{
    "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": [
    ],
    "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

```json
{
    "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": [
    ],
    "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**

```json
{
    "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"
    }
}
```

**Container removed**

```json
{
    "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, near real-time metrics. These 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 (p. ) to the container to allow MediaStore to send metrics to Amazon CloudWatch.

<table>
<thead>
<tr>
<th>Metric</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>RequestCount</td>
<td>The total number of HTTP requests made to a MediaStore container, separated by operation type (Put, Get, Delete, Describe, List).</td>
</tr>
<tr>
<td>Units: Count</td>
<td>Valid dimensions:</td>
</tr>
<tr>
<td>Valid dimensions:</td>
<td>• Container name</td>
</tr>
<tr>
<td></td>
<td>• Object group name</td>
</tr>
<tr>
<td></td>
<td>• Request type</td>
</tr>
<tr>
<td>Valid statistics:</td>
<td>Sum</td>
</tr>
</tbody>
</table>

<p>| 4xxErrorCount       | The number of HTTP requests made to MediaStore that resulted in a 4xx error. |
| Units: Count        | Valid dimensions:                                                           |
| Valid dimensions:   | • Container name                                                             |
|                     | • Object group name                                                          |
|                     | • Request type                                                               |
| Valid statistics:   | Sum                                                                          |</p>
<table>
<thead>
<tr>
<th>Metric</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>5xxErrorCount</td>
<td>The number of HTTP requests made to MediaStore that resulted in a 5xx error.</td>
</tr>
<tr>
<td></td>
<td>Units: Count</td>
</tr>
<tr>
<td></td>
<td>Valid dimensions:</td>
</tr>
<tr>
<td></td>
<td>• Container name</td>
</tr>
<tr>
<td></td>
<td>• Object group name</td>
</tr>
<tr>
<td></td>
<td>• Request type</td>
</tr>
<tr>
<td></td>
<td>Valid statistics: Sum</td>
</tr>
<tr>
<td>BytesUploaded</td>
<td>The number of bytes uploaded for requests made to a MediaStore container,</td>
</tr>
<tr>
<td></td>
<td>where the request includes a body.</td>
</tr>
<tr>
<td></td>
<td>Units: Bytes</td>
</tr>
<tr>
<td></td>
<td>Valid dimensions:</td>
</tr>
<tr>
<td></td>
<td>• Container name</td>
</tr>
<tr>
<td></td>
<td>• Object group name</td>
</tr>
<tr>
<td></td>
<td>Valid statistics: Average (bytes per request), Sum (bytes per period),</td>
</tr>
<tr>
<td></td>
<td>Sample Count, Min (same as P0.0), Max (same as p100), any percentile between</td>
</tr>
<tr>
<td></td>
<td>p0.0 and p99.9</td>
</tr>
<tr>
<td>BytesDownloaded</td>
<td>The number of bytes downloaded for requests made to a MediaStore container,</td>
</tr>
<tr>
<td></td>
<td>where the response includes a body.</td>
</tr>
<tr>
<td></td>
<td>Units: Bytes</td>
</tr>
<tr>
<td></td>
<td>Valid dimensions:</td>
</tr>
<tr>
<td></td>
<td>• Container name</td>
</tr>
<tr>
<td></td>
<td>• Object group name</td>
</tr>
<tr>
<td></td>
<td>Valid statistics: Average (bytes per request), Sum (bytes per period),</td>
</tr>
<tr>
<td></td>
<td>Sample Count, Min (same as P0.0), Max (same as p100), any percentile between</td>
</tr>
<tr>
<td></td>
<td>p0.0 and p99.9</td>
</tr>
</tbody>
</table>
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 and Cost Management User Guide.

### Tagging AWS Elemental MediaStore resources

<table>
<thead>
<tr>
<th>Metric</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>TotalTime</strong></td>
<td>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</td>
</tr>
<tr>
<td><strong>TurnaroundTime</strong></td>
<td>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</td>
</tr>
</tbody>
</table>
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 (p. 82).

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 (p. 83)
- AWS Elemental MediaStore's interaction with HTTP caches (p. 84)

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. To get started, you attach a policy to your container that grants read access or greater to CloudFront.

To allow CloudFront to access your container (console)

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.
Note
The example policy for Public Read Access over HTTPS (p. 16) matches these requirements because it allows GetObject and DescribeObject commands from anyone who submits requests to your domain through HTTPS.

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

Note
A CORS policy (p. 19) 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.

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 (p. 47) 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 (p. 44) 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.
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.

<table>
<thead>
<tr>
<th>Resource or Operation</th>
<th>Default Quota</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>Containers</td>
<td>100</td>
<td>The maximum number of containers that you can create in your account.</td>
</tr>
<tr>
<td>DeleteObject</td>
<td>100 transactions per second (TPS)</td>
<td>The maximum number of operation requests that you can make per second. Additional requests are throttled. You can request a quota increase.</td>
</tr>
<tr>
<td>DescribeObject</td>
<td>1,000 TPS</td>
<td>The maximum number of operation requests that you can make per second. Additional requests are throttled. You can request a quota increase.</td>
</tr>
<tr>
<td>Folder Levels</td>
<td>10</td>
<td>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.</td>
</tr>
<tr>
<td>Folders</td>
<td>Unlimited</td>
<td>You can create as many folders as you want, as long as they are not nested more than 10 levels within a container.</td>
</tr>
<tr>
<td>GetObject – Standard Upload Availability</td>
<td>1,000 TPS</td>
<td>The maximum number of operation requests that you can make per second. Additional requests are throttled. You can request a quota increase.</td>
</tr>
<tr>
<td>GetObject – Streaming Upload Availability</td>
<td>25 TPS</td>
<td>The maximum number of operation requests that you can make per second. Additional requests are throttled. You can request a quota increase.</td>
</tr>
<tr>
<td>ListItems</td>
<td>5 TPS</td>
<td>The maximum number of operation requests that you can make per second. Additional requests are throttled. You can request a quota increase.</td>
</tr>
<tr>
<td>Object Size</td>
<td>25 MB</td>
<td>The maximum file size of a single object.</td>
</tr>
<tr>
<td>Resource or Operation</td>
<td>Default Quota</td>
<td>Comments</td>
</tr>
<tr>
<td>-----------------------</td>
<td>---------------</td>
<td>----------</td>
</tr>
<tr>
<td>Objects</td>
<td>Unlimited</td>
<td>You can upload as many objects as you want to a folder or container in your account.</td>
</tr>
<tr>
<td><strong>PutObject</strong> – Standard Upload Availability</td>
<td>100 TPS</td>
<td>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.</td>
</tr>
<tr>
<td><strong>PutObject</strong> – Streaming Upload Availability</td>
<td>10 TPS</td>
<td>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.</td>
</tr>
<tr>
<td>Rules in an Object Lifecycle Policy</td>
<td>10</td>
<td>The maximum number of rules that you can include in an object lifecycle policy.</td>
</tr>
<tr>
<td>Rules in a Metric Policy</td>
<td>5</td>
<td>The maximum number of rules that you can include in a metric policy. You can request a quota increase.</td>
</tr>
</tbody>
</table>
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 Tools** – Links to developer tools, SDKs, IDE toolkits, and command line tools for developing and managing AWS applications.
- **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.
- **AWS Support** – The primary webpage for information about AWS 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.

<table>
<thead>
<tr>
<th>update-history-change</th>
<th>update-history-description</th>
<th>update-history-date</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tag-based access control (p. 57)</td>
<td>You can now set access permissions for a resource based tags that you've assigned to the resource.</td>
<td>November 18, 2020</td>
</tr>
<tr>
<td>ExpiresAt field (p. 74)</td>
<td>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.</td>
<td>July 16, 2020</td>
</tr>
<tr>
<td>Lifecycle transition rules (p. 25)</td>
<td>You can now add a lifecycle transition rule to your object lifecycle policy that sets objects to be moved to the infrequent access (IA) storage class after they reach a certain age.</td>
<td>April 20, 2020</td>
</tr>
<tr>
<td>Empty container (p. 49)</td>
<td>You can now delete all objects within a container at once.</td>
<td>April 7, 2020</td>
</tr>
<tr>
<td>Support for Amazon CloudWatch metrics (p. 35)</td>
<td>You can set a metric policy to dictate which metrics MediaStore sends to CloudWatch.</td>
<td>March 30, 2020</td>
</tr>
<tr>
<td>Wildcards in delete object rules (p. 25)</td>
<td>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.</td>
<td>December 20, 2019</td>
</tr>
<tr>
<td>Object lifecycle policies (p. 25)</td>
<td>You can now add a rule to your object lifecycle policy that indicates an expiration by age in seconds.</td>
<td>September 13, 2019</td>
</tr>
<tr>
<td>AWS CloudFormation support (p. 9)</td>
<td>You can now use an AWS CloudFormation template to create a container automatically. 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</td>
<td>May 17, 2019</td>
</tr>
<tr>
<td>Feature</td>
<td>Description</td>
<td>Date</td>
</tr>
<tr>
<td>--------------------------------------------</td>
<td>-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
<td>--------------------</td>
</tr>
<tr>
<td><strong>Quotas for streaming upload availability</strong></td>
<td>For objects with streaming upload availability (chunked transfer of objects), the <code>PutObject</code> operation can't exceed 10 TPS and the <code>GetObject</code> operation can't exceed 25 TPS.</td>
<td>April 8, 2019</td>
</tr>
<tr>
<td><strong>Chunked transfer of objects (p. 44)</strong></td>
<td>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.</td>
<td>April 5, 2019</td>
</tr>
<tr>
<td><strong>Access logging (p. 70)</strong></td>
<td>AWS Elemental MediaStore now supports access logging, which provides detailed records for the requests that are made to objects in a container.</td>
<td>February 25, 2019</td>
</tr>
<tr>
<td><strong>Object lifecycle policies (p. 25)</strong></td>
<td>Added support for object lifecycle policies, which govern the expiration date of objects within the current container.</td>
<td>December 12, 2018</td>
</tr>
<tr>
<td><strong>Increased object size quota (p. 86)</strong></td>
<td>The quota for an object's size is now 25 MB.</td>
<td>October 10, 2018</td>
</tr>
<tr>
<td><strong>Increased object size quota (p. 86)</strong></td>
<td>The quota for an object's size is now 20 MB.</td>
<td>September 6, 2018</td>
</tr>
<tr>
<td><strong>AWS CloudTrail integration (p. 67)</strong></td>
<td>The CloudTrail integration content has been updated to align with recent changes to the CloudTrail service.</td>
<td>July 12, 2018</td>
</tr>
<tr>
<td><strong>CDN collaboration (p. 83)</strong></td>
<td>Added information about how to use AWS Elemental MediaStore with a content delivery network (CDN) such as Amazon CloudFront.</td>
<td>April 14, 2018</td>
</tr>
<tr>
<td><strong>CORS configurations (p. 19)</strong></td>
<td>AWS Elemental MediaStore now supports cross-origin resource sharing (CORS), which allows client web applications that are loaded in one domain to interact with resources in a different domain.</td>
<td>February 7, 2018</td>
</tr>
</tbody>
</table>
New service and guide (p. 1)  This is the initial release of the video origination and storage service, AWS Elemental MediaStore, and the AWS Elemental MediaStore 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 General Reference.