AWS Elemental MediaStore

User Guide

AWS Elemental MediaStore: User Guide

Table of Contents

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 AWS Elemental MediaStore.

AWS Elemental 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 are not delivering live streaming videos, consider using Amazon Simple Storage Service (Amazon S3).

To learn more about video and AWS Media Services, register for any of our free 30-minute online training courses:

- Introduction to AWS Media Services
- AWS Elemental Foundations - Video Compression Basics
- AWS Elemental Foundations - Video Delivery Basics

Topics

AWS Elemental MediaStore Concepts and Terminology

ARN

An Amazon Resource Name.

Body

The data to be uploaded into an object.

(Byte) range

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

Container

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

Endpoint

An entry point to the AWS Elemental MediaStore service, given as an HTTP(S) 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 AWS Elemental MediaStore. The service accepts all file types.

Origination service

AWS Elemental 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.

AWS Elemental MediaStore verbs, Create

Creates an object, often implemented with HTTP POST.

Delete

Deletes an object, often implemented with HTTP DELETE.

Describe

Returns metadata about an object, often implemented with HTTP HEAD.

Get

Retrieves an object.

List

Retrieves a list of items, which can be objectsor folders.

Put

Updates an object, often implemented with HTTP PUT.

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 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 AWS Elemental 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 AWS Elemental 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 AWS Elemental MediaStore.
- **AWS SDKs** – If you're using a programming language that AWS provides an SDK for, you can use an SDK to access AWS Elemental MediaStore. SDKs simplify authentication, integrate easily with your development environment, and provide easy access to MediaStore commands. For more information, see Tools for Amazon Web Services.
- **AWS Elemental MediaStore API** – If you're using a programming language that an SDK isn't available for, see the AWS Elemental MediaStore API Reference for information about API actions and about how to make API requests.
- **AWS Command Line Interface** – For more information, see the AWS Command Line Interface User Guide.
- **AWS Tools for Windows PowerShell** – For more information, see the AWS Tools for Windows PowerShell User Guide.

Pricing for AWS Elemental MediaStore

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

Regions for AWS Elemental MediaStore

To reduce data latency in your applications, AWS Elemental MediaStore offers a regional endpoint to make your requests. To view the list of AWS Regions in which MediaStore is available, see https://docs.aws.amazon.com/general/latest/gr/rande.html#mediastore_region.

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

If you do not have an AWS account, use the following procedure to create one.

To sign up for AWS

1. Open https://aws.amazon.com/, and then choose **Create an AWS Account**.

   Note
   If you previously signed in to the AWS Management Console using AWS account root user credentials, choose **Sign in to a different account**. If you previously signed in to the console using IAM credentials, choose **Sign-in using root account credentials**. Then choose **Create a new AWS account**.

2. Follow the online instructions.

   Part of the sign-up procedure involves receiving a phone call and entering a verification code using 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 IAM user for yourself and add the user to an Administrators group

1. Use your AWS account email address and password to sign in as the *AWS account root user* to the IAM console at https://console.aws.amazon.com/iam/.

   Note
   We strongly recommend that you adhere to the best practice of using the **`Administrator`** IAM user below and securely lock away the root user credentials. Sign in as the root user only to perform a few account and service management tasks.
2. In the navigation pane of the console, choose **Users**, and then choose **Add user**.
3. For **User name**, type **`Administrator`**.
4. Select the check box next to **AWS Management Console access**, select **Custom password**, and then type the new user's password in the text box. You can optionally select **Require password reset** to force the user to create a new password the next time the user signs in.
5. Choose **Next: Permissions**.
6. On the **Set permissions** page, choose **Add user to group**.
7. Choose **Create group**.
8. In the **Create group** dialog box, for **Group name** type **`Administrators`**.
9. For **Filter policies**, select the check box for **AWS managed - job function**.
10. In the policy list, select the check box for **AdministratorAccess**. Then choose **Create group**.
11. Back in the list of groups, select the check box for your new group. Choose **Refresh** if necessary to see the group in the list.
12. Choose **Next: Tags** to add metadata to the user by attaching tags as key-value pairs.
13. Choose **Next: Review** to see the list of group memberships to be added to the new user. When you are ready to proceed, choose **Create user**.

You can use this same process to create more groups and users, and to give your users access to your AWS account resources. To learn about using policies to restrict users' permissions to specific AWS resources, go to Access Management and Example Policies.

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 Policies

Create two policies for AWS Elemental MediaStore: one to provide read/write access and one to provide read-only access. Perform these steps one time only for each policy.

To create policies

1. Use your AWS account ID or account alias, and the credentials for your admin IAM user, to sign in to the IAM console.

2. In the navigation pane of the console, choose **Policies**, and then choose **Create policy**.
3. Choose the **JSON** tab and paste the following policy:

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

   This policy allows all actions on all resources in AWS Elemental MediaStore.
4. Choose **Review policy**.
5. On the **Review policy** page, for **Name**, enter **`MediaStoreAllAccess`** , and then choose **Create policy**.
6. On the **Policies** page, repeat steps 1-5 to create a read-only policy. Use the following policy and name it **`MediaStoreReadOnlyAccess`**:

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

Step 2: Create User Groups

You can create a user group for each 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 **MediaStoreAllAccess** policy and one for the **MediaStoreReadOnlyAccess** policy.

To create user groups

1. In the navigation pane of the IAM console, choose **Groups**, and then choose **Create New Group**.
2. On the **Set Group Name** page, enter a name for the group, such as **`MediaStoreAdmins`**.

3. Choose **Next Step**.
4. On the **Attach Policy** page, for **Filter**, choose **Customer Managed**.
5. In the policy list, choose the **MediaStoreAllAccess** policy that you created in the procedure the section called "Step 1: Create Policies" (p. 5).
6. Choose **Next Step**.
7. On the **Review** page, verify that the correct policies are added to this group, and then choose **Create Group**.
8. On the **Groups** page, repeat steps 1-7 to create a user group with read-only permissions. Use the following guidelines:

   - In step 2, enter a group name such as `MediaStoreReaders`.
   - In step 4, choose the **MediaStoreReadOnlyAccess** policy that you created in the procedure the section called "Step 1: Create Policies" (p. 5).

Step 3: 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. If you have already created users, skip to step 6 to modify the permissions for the users.

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 AWS Elemental 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 AWS Elemental MediaStore.
   - The **MediaStoreReaders** group has permissions that allow read-only rights for all resources in AWS Elemental 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

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. 10).
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 AWS Elemental 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 MediaStore Pricing.

Topics

Rules for Container Names

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

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

Creating a Container

You can create up to 100 containers for each AWS account. However, there is no limit to the number of folders that you can create in each of those containers. In addition, there is no limit to the number of objects that you can upload to each container.

To create a container (console)

1. Open the MediaStore console at https://console.aws.amazon.com/mediastore/.
2. On the **Containers** page, choose **Create container**.
3. For **Container** name, type a name for the container. For more information, see Rules for Container Names (p. 10).
4. Choose **Create container**. AWS Elemental MediaStore adds the new container to a list of containers. Initially, the status of the container is **Creating**, and then it changes to **Active**.

To create a container (AWS CLI)

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

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

The following example shows the return value:

```
{
    "Container": {
        "Status": "CREATING",
        "CreationTime": 1506528818.0,
        "Name": "ExampleContainer",
        "ARN": "arn:aws:mediastore:us-west-2:111222333444:container/ExampleContainer"
    }
}
```

Viewing the Details for a Container

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

To view the details for a container (console)

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

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

   - The **Objects** section, which lists the objects and folders in the container.
   - The **Container** policy section, which shows the resource-based policy that is associated with this container. For information about resource policies, see ??? (p. 13).

To view the details for a container (AWS CLI)

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

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

The following example shows the return value:

```
{
    "Container": {
        "Status": "ACTIVE",
        "Endpoint": "https://aaabbbcccdddee.data.mediastore.us-west-2.amazonaws.com",
        "CreationTime": 1506528818.0,
        "Name": "ExampleContainer",
        "ARN": "arn:aws:mediastore:us-west-2:111222333444:container/ExampleContainer"
    }
}
```

Viewing a List of Containers

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

To view a list of containers (console)

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

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

To view a list of containers (AWS CLI)

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

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

The following example shows the return value:

```
{
    "Inputs": [
       {
    "Containers": [
        {
            "Status": "ACTIVE",
            "Endpoint": "https://aaabbbcccdddee.data.mediastore.us-
west-2.amazonaws.com",
            "CreationTime": 1505317931.0,
            "Name": "ExampleLiveDemo",
            "ARN": "arn:aws:mediastore:us-west-2:111222333444:container/
ExampleLiveDemo"
        },
        {
            "Status": "ACTIVE",
            "Endpoint": "https://fffggghhhiiijj.data.mediastore.us-
west-2.amazonaws.com",
            "CreationTime": 1506528818.0,
            "Name": "ExampleContainer",
            "ARN": "arn:aws:mediastore:us-west-2:111222333444:container/
ExampleContainer"
        }
    ]
}
```

Deleting a Container

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

To delete a container (console)

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

To delete a container (AWS CLI)

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

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

This command has no return value.

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 use the AWS CLI to 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

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

To view a container policy (console)

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

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

To view a container policy (AWS CLI)

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

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

The following example shows the return value:

```
{
  "Policy": {
    "Version": "2012-10-17",
    "Statement": [
      {
        "Sid": "MediaStoreFullAccess",
        "Effect": "Allow",
        "Principal": "*",
```

```
        "Action": "mediastore:*",
        "Resource": "arn:aws:mediastore:us-west-2:111222333444:container/
ExampleLiveDemo/*",
        "Condition": {
          "Bool": {
            "aws:SecureTransport": "true"
          }
        }
      }
    ]
  }
}
```

Editing a Container Policy

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

To edit a container policy (console)

1. Open the MediaStore console at https://console.aws.amazon.com/mediastore/.
2. On the **Containers** page, choose the container name.
3. Choose **Edit policy**. For examples that show how to set different permissions, see the section called "Example Container Policies" (p. 14).
4. Make the appropriate changes, and then choose **Save**.

To edit a container policy (AWS CLI)

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

```
aws mediastore put-container-policy --region us-west-2 --container-name ExampleLiveDemo
 --policy-name=default --policy={\"Version\" : \"2012-10-17\",  \"Statement\" : [ {
   \"Sid\" : \"MediaStoreFullAccess\",    \"Effect\" : \"Allow\",    \"Principal\" :
 \"*\",    \"Action\" : \"mediastore:*\",    \"Resource\" : \"arn:aws:mediastore:us-
west-2:111222333444:container/ExampleLiveDemo/*\",    \"Condition\" : {      \"Bool\" :
 {        \"aws:SecureTransport\" : \"true\"      }    }  } ]}"
}
```

This command has no return value.

Example Container Policies

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

Topics

Example Container Policy: Default

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

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

The policy is built into the service, so you don't have to create it. The default policy can't be changed; however, you can edit a container's policy.

The default policy that is assigned 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.

Example Container Policy: Public Read Access over HTTPS

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

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

```
        "Bool": {
            "aws:SecureTransport": "true"
        }
      }
    }
  ]
}
```

Example Container Policy: Public Read Access over HTTP or HTTPS

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

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

Example Container Policy: Cross-Account Read Access—HTTP Enabled

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

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

Example Container Policy: Cross-Account Read Access over HTTPS

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

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

Example Container Policy: Cross-Account Read Access to a Role

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

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

Example Container Policy: Cross-Account Full Access to a Role

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

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

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

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

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

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

Example Container Policy: Post Access to an AWS Service to a Folder

This policy allows another AWS service to post objects in AWS Elemental MediaStore. It allows access to `PutObject` on any object and allows this access to a specific AWS service. 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`):

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

```
  "Statement": [
    {
      "Sid": "MediaStorePostToSpecificPath",
      "Effect": "Allow",
      "Action": "mediastore:PutObject",
      "Principal":{
        "AWS": "<aws service principal>"},
      "Resource": "arn:aws:mediastore:<region>:<owner acct number>:container/<container
 name>/<specific path>/*",
      "Condition": {
        "Bool": {
            "aws:SecureTransport": "true"
        }
      }
    }
  ]
}
```

Example Container Policy: Post Access to an AWS Service to Multiple Folders

This policy is a variation on `MediaStorePostToSpecificPath` that shows how to set up access to two different paths:

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "MediaStorePostToSeveralPaths",
      "Effect": "Allow",
      "Action": "mediastore:PutObject",
      "Principal":{
        "AWS": "<aws service principal>"},
      "Resource": [
        "arn:aws:mediastore:<region>:<owner acct number>:container/<container name>/
<specific path 1>/*",
        "arn:aws:mediastore:<region>:<owner acct number>:container/<container name>/
<specific path 2>/*",
      ],
      "Condition": {
        "Bool": {
            "aws:SecureTransport": "true"
        }
      }
    }
  ]
}
```

Cross-Origin Resource Sharing (CORS) in AWS Elemental MediaStore

Cross-origin resource sharing (CORS) defines a way for client web applications that are loaded in one domain to interact with resources in a different domain. With CORS support in AWS Elemental MediaStore, you can build rich client-side web applications with MediaStore and selectively allow cross-origin access to your MediaStore resources.

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

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

Topics

CORS Use-case Scenarios

The following are example scenarios for using CORS:

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

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

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

To add a CORS policy (AWS CLI)

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

```
aws mediastore put-cors-policy --container-name ExampleContainer --cors-
policy '[{"AllowedOrigins": ["*"],"AllowedMethods": ["GET"],"AllowedHeaders":
 ["*"],"ExposeHeaders": ["*"], "MaxAgeSeconds":3000}]' --region ap-southeast-2 --
endpoint https://mediastore.ap-southeast-2.amazonaws.com/
```

This command has no return value.

Viewing a CORS Policy

Cross-origin resource sharing (CORS) defines a way for client web applications that are loaded in one domain to interact with resources in a different domain.

To view a CORS policy (console)

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

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

To view a CORS policy (AWS CLI)

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

```
aws mediastore get-cors-policy --container-name ExampleContainer --region ap-
southeast-2 --endpoint https://mediastore.ap-southeast-2.amazonaws.com/
```

The following example shows the return value:

```
[
 {
  "AllowedOrigins": ["http://example.com"],
```

```
  "AllowedMethods": ["GET"],
  "AllowedHeaders": ["*"],
  "MaxAgeSeconds": 3000
 },
 {
  "AllowedOrigins": ["https://*"],
  "AllowedMethods": ["GET", "PUT"],
  "AllowedHeaders": ["x-amzn*"],
  "MaxAgeSeconds": 0
 }
]
```

Editing a CORS Policy

Cross-origin resource sharing (CORS) defines a way for client web applications that are loaded in one domain to interact with resources in a different domain.

To edit a CORS policy (console)

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

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

Deleting a CORS Policy

Cross-origin resource sharing (CORS) defines a way for client web applications that are loaded in one domain to interact with resources in a different domain. Deleting the CORS policy from a container removes permissions for cross-origin requests.

To delete a CORS policy (console)

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

   The container details page appears.
3. In the **Container CORS policy** section, choose **Edit CORS policy**.
4. Clear the text from the text box, 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 ap-
southeast-2 --endpoint https://mediastore.ap-southeast-2.amazonaws.com/
```

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

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

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

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

Example CORS Policy: Read Access for a Specific Domain

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

```
[
  {
    "AllowedHeaders": [
      "*"
    ],
    "AllowedMethods": [
      "GET",
      "HEAD"
    ],
    "AllowedOrigins": [
      "https://www.example.com"
    ],
    "MaxAgeSeconds": 3000
  }
]
```

Object Lifecycle Policies in AWS Elemental MediaStore

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

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 the section called "Components of an Object Lifecycle Policy" (p. 25).

Topics

Components of an Object Lifecycle Policy

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

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

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

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

```
                "days_since_create": [
                    {"numeric": [">" , 15]}
                ]
            },
            "action": "EXPIRE"
        }
        {
            "definition": {
                "path": [ { "prefix": "" }  ],
                "days_since_create": [
                    {"numeric": [">" , 40]}
                ]
            },
            "action": "EXPIRE"
        }
    ]
}
```

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 after they are older than 15 days.
- 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.

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. To create the object lifecycle policy and attach it to a container, you use the AWS CLI. If the container already has an object lifecycle policy, the service replaces the existing policy with the new policy. It takes up to 20 minutes for the new policy to take effect.

To create an object lifecycle policy (AWS CLI)

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

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

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

```
aws mediastore put-lifecycle-policy --container-name LiveEvents --lifecycle-
policy file://LiveEventsLifecyclePolicy
```

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 stored in a container. To view an object lifecycle policy that is attached to a container, use the AWS CLI.

To view an object lifecycle policy (AWS CLI)

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

```
aws mediastore get-lifecycle-policy --container-name LiveEvents
```

The following example shows the return value:

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

Changing 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 updated policy to take effect.

To change an object lifecycle policy (AWS CLI)

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

```
{
    "rules": [
         {
            "definition": {
                "path": [
                    {"prefix": "Football/"},
                    {"prefix": "Baseball/"}
                    {"prefix": "Basketball/"}
```

```
                ],
                "days_since_create": [
                    {"numeric": [">" , 28]}
                ]
            },
            "action": "EXPIRE"
        }
    ]
}
```

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

```
aws mediastore put-lifecycle-policy --container-name LiveEvents --lifecycle-policy
 file://LiveEvents2LifecyclePolicy
```

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

Deleting an Object Lifecycle Policy

When you delete an object lifecycle policy, it takes up to 20 minutes for the change to take effect.

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
```

This command has no return value. The service deletes the object lifecycle policy within 20 minutes.

Folders in AWS Elemental MediaStore

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

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

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

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

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

Topics

Rules for Folder Names

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

- The name 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 name can contain uppercase letters, lowercase letters, numbers, periods (.), hyphens (-), and tildes (~).
- The name must start with a number or letter.
- The name must be from 3 to 63 characters long.
- The name must not be formatted as an IP address (for example, 192.168.5.4).
- The name must not contain underscores (_).
- The name must not end with a hyphen.
- The name can't contain two, adjacent periods.

- The name can't contain dashes next to periods (e.g., my-.container.com and my.-container are invalid).
- 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 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. 31).

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. 35).

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 AWS Elemental 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 AWS Elemental MediaStore.
- **Download** – Copy an object from AWS Elemental MediaStore to another location. This does not remove the object from MediaStore.
- **Delete** – Remove an object from AWS Elemental MediaStore completely. You can delete objects individually, or you can add an object lifecycle policy (p. 26) to automatically delete objects within a container after a specified duration.

AWS Elemental MediaStore accepts all file types.

Topics

Uploading an Object

You can upload objects to a container or to a folder within a container. To upload an object to a folder, you specify the path to the folder. If the folder already exists, AWS Elemental MediaStore stores the object in the folder. If the folder doesn't exist, the service creates it, and then stores the object in the folder. For more information about folders, see Folders in AWS Elemental MediaStore (p. 29).

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

AWS Elemental 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 using the AWS CLI. If you don't specify this header in your request, AWS Elemental MediaStore assigns the default value of `standard` for the object's upload availability.

Object sizes are limited to 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 (~), and hyphens (-).

To upload an object (console)

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

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

To upload an object (AWS CLI)

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

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

```
aws mediastore-data put-object --region us-west-2 --endpoint=https://
aaabbbcccdddee.data.mediastore.us-west-2.amazonaws.com --body=README.md --path=/test/
document/README3.md --cache-control "max-age=6, public" --content-type binary/octet-
stream --upload-availability streaming
```

The following example shows the return value:

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

Viewing a List of Objects

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

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

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

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

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

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

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

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

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

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

The following example shows the return value:

```
{
    "Items": [
        {
            "Type": "FOLDER",
            "Name": "ExampleLiveDemo"
        },
        {
            "Type": "FOLDER",
            "Name": "folder_1"
        }
    ]
}
```

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

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

```
aws mediastore-data --region us-west-2 list-items --endpoint=https://
aaabbbcccdddee.data.mediastore.us-west-2.amazonaws.com --path=/folder_1
```

The following example shows the return value:

```
{
    "Items": [
        {
            "Type": "OBJECT",
            "Name": "1512519711640.ts"
        },
        {
            "Type": "OBJECT",
            "Name": "test_file.pdf"
        }
    ]
}
```

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 view the details of an object (console)

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

   A details page appears, showing information about the object.

To view the details of an object (AWS CLI)

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

```
aws mediastore-data --region us-west-2 describe-object --endpoint=https://
aaabbbcccdddee.data.mediastore.us-west-2.amazonaws.com --path=/test/document/README3.md
```

The following example shows the return value:

```
{
    "LastModified": "Mon, 20 Nov 2017 19:30:18 GMT"
    "ContentLength": "2774",
    "ETag": "2aa333bbcc8d8d22d777e999c88d4aa9eeeeee4dd89ff7f555555555555da6d3",
    "ContentType": "binary/octet-stream",
}
```

Downloading an Object

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

To download an object (console)

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

To download an object (AWS CLI)

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

```
aws mediastore-data --region us-west-2 get-object --endpoint=https://
aaabbbcccdddee.data.mediastore.us-west-2.amazonaws.com --path=/test/document/
README3.md README3.md
```

The following example shows the return value:

```
{
    "ContentType": "binary/octet-stream",
    "ContentLength": "2774",
    "CacheControl": "pre-commit",
    "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 --region us-west-2 get-object --endpoint=https://
aaabbbcccdddee.data.mediastore.us-west-2.amazonaws.com --path=/test/document/README3.md
 --range="bytes=0-100" README4.md
```

The following example shows the return value:

```
{
    "ContentType": "binary/octet-stream",
    "ContentRange": "bytes 0-100/2774",
    "CacheControl": "pre-commit",
    "ContentLength": "101",
    "StatusCode": 206
}
```

Deleting an Object

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

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

To delete an object (console)

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

To delete an object (AWS CLI)

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

  Example:

```
aws mediastore-data --region us-west-2 delete-object --endpoint=https://
aaabbbcccdddee.data.mediastore.us-west-2.amazonaws.com --path=/test/document/README3.md
```

  This command has no return value.

To automatically delete objects based on their age in the container (AWS CLI)

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

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

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

```
aws mediastore put-lifecycle-policy --container-name LiveEvents --lifecycle-
policy file://LiveEventsLifecyclePolicy
```

  This command has no return value.

AWS CLI Commands for AWS Elemental MediaStore

The following table shows the AWS CLI commands that you can use to create or modify containers and objects in AWS Elemental MediaStore.

Applies to...	Command	Description
containers (mediastore)	`create-container`	Creates a container.
containers (mediastore)	`delete-container`	Deletes a container. You can't delete a container that has objects; you can delete only empty containers.
containers (mediastore)	`delete-container-policy`	Removes a container policy from a container.
containers (mediastore)	`delete-lifecycle-policy`	Removes an object lifecycle policy from a container.
containers (mediastore)	`get-container-policy`	Retrieves the current policy of a container.
containers (mediastore)	`get-lifecycle-policy`	Retrieves the object lifecycle policy that is assigned to a container.
containers (mediastore)	`list-containers`	Lists all of your containers.
containers (mediastore)	`put-container-policy`	Replaces the current policy of a container with the specified policy.
containers (mediastore)	`put-lifecycle-policy`	Writes an object lifecycle policy to a container. If the container already has an object lifecycle policy, the service replaces the previous policy with the new policy.
objects (mediastore-data)	`delete-object`	Deletes an object that is stored in a container.
objects (mediastore-data)	`describe-object`	Retrieves information about an object that is stored in a container.
objects (mediastore-data)	`get-object`	Downloads an object from AWS Elemental MediaStore to a specified endpoint. You can provide a byte range to download only the part of the object that corresponds to the range.
objects (mediastore-data)	`help`	Displays information about the command being called. Append the keyword `help` to the end of any partial command line.

Applies to...	Command	Description
objects (mediastore-data)	`list-items`	Lists folders and objects stored in a container.
objects (mediastore-data)	`put-object`	Writes an object to a container.

Monitoring 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:

- *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.
- *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.

Topics

Logging AWS Elemental MediaStore API Calls with AWS CloudTrail

AWS Elemental MediaStore is integrated with AWS CloudTrail, a service that provides a record of actions taken by a user, role, or an AWS service in MediaStore. CloudTrail captures a subset of API calls for MediaStore as events, including calls from the MediaStore console and from code calls to the MediaStore API. If you create a trail, you can enable continuous delivery of CloudTrail events to an Amazon S3 bucket, including events for MediaStore. If you don't configure a trail, you can still view the most recent events in the CloudTrail console in **Event history**. Using the information collected by CloudTrail, you can determine the request that was made to MediaStore, the IP address from which the request was made, who made the request, when it was made, and more.

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

Topics

AWS Elemental MediaStore Information in CloudTrail

CloudTrail is enabled on your AWS account when you create the account. When supported event activity occurs in AWS Elemental MediaStore, that activity is recorded in a CloudTrail event along with other AWS service events in **Event history**. You can view, search, and download recent events in your AWS account. For more information, see Viewing Events with CloudTrail Event History.

For an ongoing record of events in your AWS account, including events for AWS Elemental 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:


```
{
    "eventVersion": "1.05",
    "userIdentity": {
        "type": "IAMUser",
        "principalId": "ABCDEFGHIJKL123456789",
        "arn": "arn:aws:iam::111122223333:user/testUser",
        "accountId": "111122223333",
        "accessKeyId": "AKIAIOSFODNN7EXAMPLE",
        "userName": "testUser",
        "sessionContext": {
            "attributes": {
                "mfaAuthenticated": "false",
                "creationDate": "2018-07-09T12:55:42Z"
            }
        },
        "invokedBy": "signin.amazonaws.com"
    },
    "eventTime": "2018-07-09T12:56:54Z",
    "eventSource": "mediastore.amazonaws.com",
    "eventName": "CreateContainer",
    "awsRegion": "ap-northeast-1",
    "sourceIPAddress": "54.239.119.16",
    "userAgent": "signin.amazonaws.com",
    "requestParameters": {
        "containerName": "TestContainer"
    },
    "responseElements": {
        "container": {
            "status": "CREATING",
            "creationTime": "Jul 9, 2018 12:56:54 PM",
            "name": " TestContainer ",
            "aRN": "arn:aws:mediastore:ap-northeast-1:111122223333:container/TestContainer"
        }
    },
    "requestID":
 "MNCTGH4HRQJ27GRMBVDPIVHEP4LO2BN6MUVHBCPSHOAWNSOKSXCO24B2UEOBBND5DONRXTMFK3TOJ4G7AHWMESI",
    "eventID": "7085b140-fb2c-409b-a329-f567912d704c",
    "eventType": "AwsApiCall",
    "recipientAccountId": "111122223333"
}
```


Monitoring AWS Elemental MediaStore with Amazon CloudWatch

You can monitor AWS Elemental MediaStore using CloudWatch, which collects raw data and processes it into readable, 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.

- 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 AWS Elemental 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 AWS Elemental 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, AWS Elemental 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 limit on the number of log streams that can belong to one log group.

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

Setting Up Permissions for Amazon CloudWatch

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

To allow MediaStore access to CloudWatch

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

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "logs:DescribeLogGroups",
                "logs:CreateLogGroup"
            ],
            "Resource": "*"
        },
        {
            "Effect": "Allow",
            "Action": [
                "logs:CreateLogStream",
                "logs:DescribeLogStreams",
                "logs:PutLogEvents"
```

```
            ],
            "Resource": "arn:aws:logs:*:*:log-group:/aws/mediastore/*"
        }
    ]
}
```

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

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

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

The entire policy should read as follows:

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

18. Choose **Update Trust Policy**.

Enabling Access Logging for a Container

By default, AWS Elemental MediaStore doesn't collect access logs. When you enable access logging on a container, MediaStore delivers access logs for objects stored in that container to Amazon CloudWatch. The access logs provide detailed records for requests that are made to any object stored in the container. This information can include the request type, the resources that are specified in the request, and the time and date that the request was processed.

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

To enable access logging (AWS CLI)

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

```
aws mediastore start-access-logging --container-name LiveEvents
```

This command has no return value.

Disabling Access Logging for a Container

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

To disable access logging (AWS CLI)

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

```
aws mediastore stop-access-logging --container-name LiveEvents
```

This command has no return value.

Troubleshooting Access Logging in AWS Elemental MediaStore

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

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

Symptom	The Problem Might Be...	Try This...
You don't see any CloudTrail events, even though CloudTrail logs are enabled.	The IAM role either does not exist or it has the incorrect name, permissions, or trust policy.	Create a role with the correct name, permissions, and trust policy. See the section called "Setting Up Permissions for CloudWatch" (p. 42).
You submitted a `DescribeContainer` API request, but the response shows that the `AccessLoggingEnabled` parameter has a value of `False`. In addition, you don't see any CloudTrail events for the `MediaStoreAccessLogs` role making a successful `DescribeLogGroup`, `CreateLogGroup`, `DescribeLogStream`, or `CreateLogStream` call.	The IAM role either does not exist or it has the incorrect name, permissions, or trust policy.	Create a role with the correct name, permissions, and trust policy. See the section called "Setting Up Permissions for CloudWatch" (p. 42).
	Access logging is not enabled on the container.	Enable access logs for the container. See the section called "Enabling Access Logging" (p. 43).

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

On the CloudTrail console, you see an event with an access denied error related to the `MediaStoreAccessLogs` role. The CloudTrail event might include lines such as the following:
 	
`"eventSource": "logs.amazonaws.com",`
 	
`"errorCode": "AccessDenied",`
 	
`"errorMessage": "User: arn:aws:sts::111122223333:assumed-role/MediaStoreAccessLogs/MediaStoreAccessLogsSession is not authorized to perform: logs:DescribeLogGroups on resource: arn:aws:logs:us-west-2:111122223333:log-group::log-stream:",`
	The IAM role doesn't have the correct permissions for AWS Elemental MediaStore.	Update the IAM role to have the correct permissions and trust policy. See the section called "Setting Up Permissions for CloudWatch" (p. 42).

You don't see any logs for an entire container or containers.	Your account might have exceeded the CloudWatch limit for log groups per account per Region. See the limits for log groups in the Amazon CloudWatch Logs User Guide.	On the CloudWatch console, determine if your account has met the CloudWatch limit for log groups. If necessary, request a limit increase.
You see some logs in CloudWatch, but not all logs that you expect to see.	Your account might have exceeded the CloudWatch limit for transactions per second per account per Region. See the limits for `PutLogEvents` in the Amazon CloudWatch Logs User Guide.	Request a limit increase for CloudWatch transactions per second per account per Region.

Access Log Format

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

```
{
  "Path": "/FootballMatch/West",
  "Requester": "arn:aws:iam::111122223333:user/maria-garcia",
  "AWSAccountId": "111122223333",
  "RequestID":
 "aaaAAA111bbbBBB222cccCCC333dddDDD444eeeEEE555fffFFF666gggGGG777hhhHHH888iiiIII999jjjJJJ",
  "ContainerName": "LiveEvents",
```

```
  "TotalTime": 147,
  "BytesSent": 184,
  "ReceivedTime": "2018-12-13T12:22:06.245Z",
  "Operation": "PutObject",
  "ErrorCode": null,
  "Source": "192.0.2.3",
  "HTTPStatus": 200,
  "TurnAroundTime': 7,
}
 {
  "Path": "/FootballMatch/West",
  "Requester": "arn:aws:iam::111122223333:user/maria-garcia",
  "AWSAccountId": "111122223333",
  "RequestID":
 "dddDDD444eeeEEE555fffFFF666gggGGG777hhhHHH888iiiIII999jjjJJJ000cccCCC333bbbBBB222aaaAAA",
  "ContainerName": "LiveEvents",
  "TotalTime": 3,
  "BytesSent": 163,
  "ReceivedTime": "2018-12-13T12:22:51.779Z",
  "Operation": "PutObject",
  "ErrorCode": "ValidationException",
  "Source": "198.51.100.15",
  "HTTPStatus": 400,
  "TurnAroundTime": 1,
}
```

The following list describes the log record fields:

Path

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

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.

Requester

The user Amazon Resource Name (ARN) of the account that was used to make the request. For unauthenticated requests, this value is `anonymous`.

AWSAccountId

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

RequestID

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

ContainerName

The name of the container that received the request.

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.

BytesSent

The number of bytes within the server response body, or for `DescribeObject` (HEAD) requests, the number of bytes that would have been sent had the request been a `GetObject` (GET). This value often is the same as the value of the `Content-Length` header included with server responses.

ReceivedTime

The time of day when the request was received. The value is `ISO86-1` date time and is based on the system clock of the host that served the request.

Operation

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

ErrorCode

The AWS Elemental 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).

HTTPStatus

The numeric HTTP status code of the response.

TurnAroundTime

The number of milliseconds that AWS Elemental 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. 48)
2. the section called “Container State Change Event” (p. 49)

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

The actions that can be automatically triggered include the following:

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

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

- Activating a Lambda function whenever a container is created
- Notifying an Amazon SNS topic when an object is deleted

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

Topics

AWS Elemental MediaStore Object State Change Event

This event is published when an object's state has changed (when the object has been uploaded or deleted). For information about subscribing to this event, see Amazon CloudWatch.

Object Updated

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

```
    "Operation": "UPDATE",
    "Path":"TVShow/Episode1/Pilot.avi",
    "ObjectSize":111122223333,
    "URL": "https://a832p1qeaznlp9.files.mediastore-us-west-2.com/Movies/MondayMornings/
Episode1/Introduction.avi"
  }
}
```

Object Removed

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

AWS Elemental MediaStore Container State Change Event

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

Container Created

```
{
  "version": "1",
  "id": "6a7e8feb-b491-4cf7-a9f1-bf3703467718",
  "detail-type": "MediaStore Container State Change",
  "source": "aws.mediastore",
  "account": "111122223333",
  "time": "2017-02-22T18:43:48Z",
  "region": "us-east-1",
  "resources": [
    "arn:aws:mediastore:us-east-1:111122223333:container/Movies"
  ],
  "detail": {
    "ContainerName": "Movies",
    "Operation": "CREATE"
    "Endpoint": "https://a832p1qeaznlp9.mediastore-us-west-2.amazonaws.com"
  }
}
```

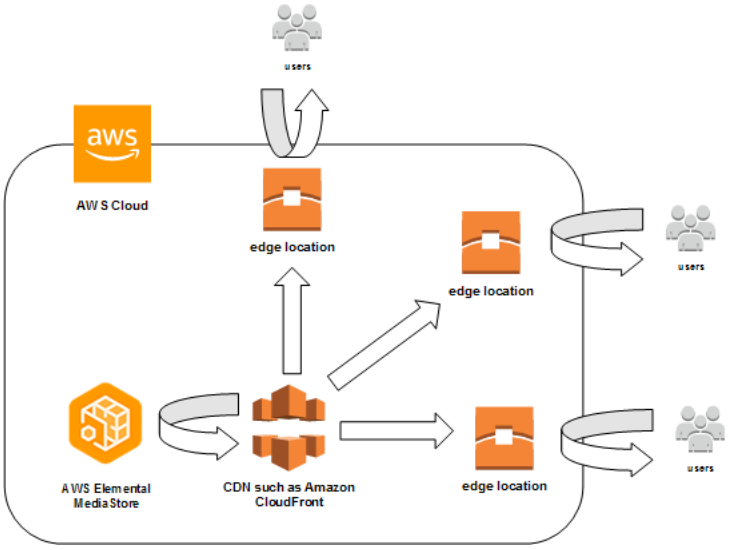
Container Removed

```
{
  "version": "1",
  "id": "6a7e8feb-b491-4cf7-a9f1-bf3703467718",
```

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

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

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)

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

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

Note
The example policy for Public Read Access over HTTPS (p. 15) 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. 20) 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.

Limits in AWS Elemental MediaStore

The following table describes limits in AWS Elemental MediaStore.

Resource or Operation	Default Limit	Comments
Containers	100	The maximum number of containers that you c account.
DeleteObject	100 transactions per second (TPS)	The maximum number of operation requests tl per second. Additional requests are throttled. You can request a limit increase.
DescribeObject	1,000 TPS	The maximum number of operation requests tl per second. Additional requests are throttled. You can request a limit increase.
Folder Levels	10	The maximum number of folder levels that you container. You can create an unlimited number as they are not nested more than 10 levels with
Folders	Unlimited	There is no limit to the number of folders that container. You can create an unlimited number as they are not nested more than 10 levels with
GetObject – Standard Upload Availability	1,000 TPS	The maximum number of operation requests tl per second. Additional requests are throttled. You can request a limit increase.
GetObject – Streaming Upload Availability	25 TPS	The maximum number of operation requests tl per second. Additional requests are throttled. You can request a limit increase.
ListItems	5 TPS	The maximum number of operation requests tl per second. Additional requests are throttled. You can request a limit increase.
Object Size	25 MB	The maximum file size of a single object.
Objects	Unlimited	There is no limit to the number of objects that folder or container in your account.
PutObject – Standard Upload Availability	100 TPS	The maximum number of operation requests tl per second. Additional requests are throttled. You can request a limit increase.
PutObject – Streaming Upload Availability	10 TPS	The maximum number of operation requests tl per second. Additional requests are throttled. You can request a limit increase.

Resource or Operation	Default Limit	Comments
Rules in an Object Lifecycle Policy	10	The maximum number of rules that you can in lifecycle policy.

AWS Elemental MediaStore Related Information

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

- **Classes & Workshops** – Links to role-based and specialty courses as well as 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 web page 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.

update-history-change	update-history-description	update-history-date
Limits for streaming upload availability (p. 53)	For objects with streaming upload availability (chunked transfer of objects), the `PutObject` operation is limited to 10 TPS and the `GetObject` operation is limited to 25 TPS.	April 8, 2019
Chunked transfer of objects (p. 31)	Added support for chunked transfer of objects. This capability allows you to specify that an object is available for downloading before the object is uploaded completely.	April 5, 2019
Access logging (p. 42)	AWS Elemental MediaStore now supports access logging, which provides detailed records for the requests that are made to objects in a container.	February 25, 2019
Object lifecycle policies (p. 25)	Added support for object lifecycle policies, which govern the expiration date of objects within the current container.	December 12, 2018
Increased object size limit (p. 53)	The limit for an object's size is now 25 MB.	October 10, 2018
Increased object size limit (p. 53)	The limit for an object's size is now 20 MB.	September 6, 2018
AWS CloudTrail integration (p. 39)	The CloudTrail integration content has been updated to align with recent changes to the CloudTrail service.	July 12, 2018
CDN collaboration (p. 51)	Added information about how to use AWS Elemental MediaStore with a content delivery network (CDN) such as Amazon CloudFront.	April 14, 2018
CORS configurations (p. 20)	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.	February 7, 2018

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*.