Amazon Lookout for Equipment

User Guide

Amazon Lookout for Equipment: User Guide

Table of Contents

What is Amazon Lookout for Equipment?

Amazon Lookout for Equipment is a machine learning (ML) service for monitoring industrial equipment that detects abnormal equipment behavior and identifies potential failures. With Lookout for Equipment, you can implement predictive maintenance programs and identify suboptimal equipment processes.

Amazon Lookout for Equipment doesn't require extensive ML knowledge or experience. You upload historical data generated by your industrial equipment to train a custom ML model that finds potential failures by leveraging up to 300 sensors into a single model. Amazon Lookout for Equipment automatically creates the best model to learn your equipment's normal operating conditions. The model is optimized to find abnormal behavior that occurred before failures in the historical data. Using either the AWS console or the AWS SDK, you run the model to process new sensor data in real time.

To use Amazon Lookout for Equipment, you do the following:

1. Format and upload your historical data to an Amazon Simple Storage Service (Amazon S3) bucket. You can use data from process historians, Supervisory Control and Data Acquisition (SCADA) systems, or another condition monitoring system. Format and upload data showing the periods of failures or abnormal behavior in your historical data, if you have it.
2. Create a dataset from the data that you've uploaded.
3. Choose the data in the dataset that is relevant to the asset whose behavior you want to analyze.
4. Add the periods of historical failures shown in the data, if you have it.
5. Train your ML model using Lookout for Equipment.
6. After fine-tuning the model, deploy it to monitor data in real time.

Amazon Lookout for Equipment is designed to monitor fixed and stationary industrial equipment that operates continuously, with limited variability in operating conditions. This includes rotating equipment such as pumps, compressors, motors, and turbines. It is also targeted for Process industries with applications such as heat exchangers, boilers, and inverters. Amazon Lookout for Equipment may not be ideal for equipment with significantly variable operating conditions, such as vehicles, robots, appliances, and computer numerical control (CNC) machines.

Topics

Are you a first-time user of Lookout for Equipment?

If you are a first-time user of Lookout for Equipment, we recommend that you read the following sections in the listed order:

1. **How Amazon Lookout for Equipment works (p. 3)** – Explains how Lookout for Equipment works and shows you how you can build a predictive maintenance system that meets your specific needs.
2. **Best practices with Lookout for Equipment (p. 43)** – Explains some basic Lookout for Equipment concepts and shows you how to get started with analyzing your data.

3. **Use case: fluid pump (p. 5)** – Provides a detailed example that you can use to explore Lookout for Equipment.

Pricing for Amazon Lookout for Equipment

For information, see Amazon Lookout for Equipment Pricing.

How Amazon Lookout for Equipment works

Amazon Lookout for Equipment uses machine learning to detect abnormal behavior in your equipment and identify potential failures. Each piece of industrial equipment is referred to as an industrial asset, or *asset*. To use Lookout for Equipment to monitor your asset, you do the following:

1. Provide Lookout for Equipment with your asset's data. The data come from sensors that measure different features of your asset. For example, you could have one sensor that measures temperature and another that measures pressure.
2. Train a custom ML model on the data.
3. Monitor your asset with the model that you've trained.

You need to train a model for each of your assets because they each have their own data signatures. A data signature indicates the distinct behavior and characteristics of an individual asset. This signature depends on the age of the equipment, its operating environment, what sensors are installed (including process data), who operates it, and many other factors. You use Amazon Lookout for Equipment to build a custom ML model for each asset. For example, you would build a custom model for each of two assets of the same asset type, *Pump 1* and *Pump 2*.

The model is trained to use data to establish a baseline for the asset. It's trained to know what constitutes normal behavior. As it monitors your equipment, it can identify abnormal behavior that might indicate a precursor to an asset failure. Amazon Lookout for Equipment uses machine learning to detect deviations from normal behavior because asset failures are rare and even the same failure type might have its own unique data pattern. All detectable failures are preceded by behavior or conditions that fall out of the normal behavior of the equipment, so Lookout for Equipment is designed to look for those behaviors or conditions.

If you have the data available, you can highlight abnormal equipment behavior using label data. The trained model can use the anomalous behavior in the dataset to improve its performance.

When you train a model, Amazon Lookout for Equipment evaluates how different types of ML models perform with your asset's data. It chooses the model that performs the best on the dataset to monitor your equipment.

You can now use the model to monitor your asset. You can schedule the frequency with which Amazon Lookout for Equipment monitors the asset.

Although Amazon Lookout for Equipment can warn you of a potential failure, it cannot tell you the exact failure mode. It can use the sensors from which it analyzed data to indicate a failure. You can use this information to see if your equipment is in a state that could lead to a failure.

1. Create and properly format .csv files containing your sensor data. For more information, see Formatting your data (p. 17).
2. Upload your data to Amazon Simple Storage Service (Amazon S3). For more information, see Uploading your data into Amazon S3 (p. 20).
3. Use a schema to create a dataset from the .csv files that you've uploaded. A schema defines the organization of your data in JSON format. The dataset that you generate from this process is a container for the sensor data that you've uploaded. For more information, see Creating a dataset in Amazon Lookout for Equipment (p. 23).

4. Ingest the dataset into Amazon Lookout for Equipment. Ingesting the dataset imports it into a format that the ML model can use for training. For more information, see Ingesting a dataset (p. 27).
5. Train a model on the dataset. For more information about training a model, see Training a model (p. 32).
6. Monitor your asset with the model that you've trained. For more information, see Monitoring your equipment in real time (p. 39).

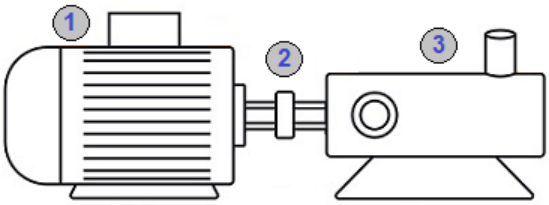
You repeat the preceding steps for each asset that you want to monitor. Before you use Amazon Lookout for Equipment to monitor your equipment, you need to set it up. To set up Amazon Lookout for Equipment, see Setting up Lookout for Equipment (p. 8).

Use case: fluid pump

Example

Lookout for Equipment is designed primarily for stationary industrial equipment that operates continuously. This includes many types of equipment, including pumps, compressors, motors, and turbines.

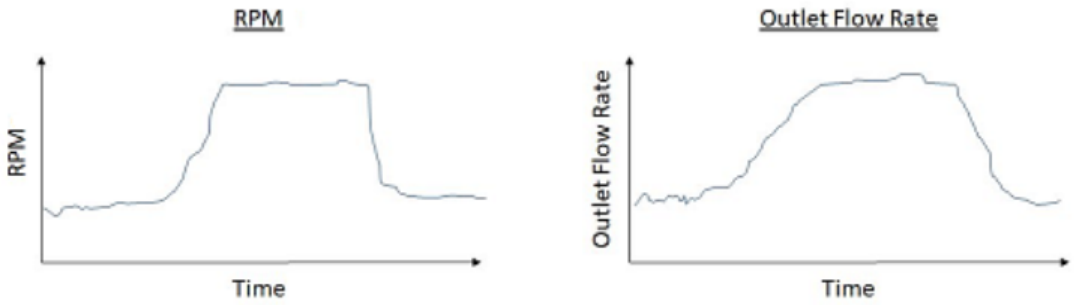
As an example of using Lookout for Equipment on data from a high-level machine , let's look at a fluid pump.



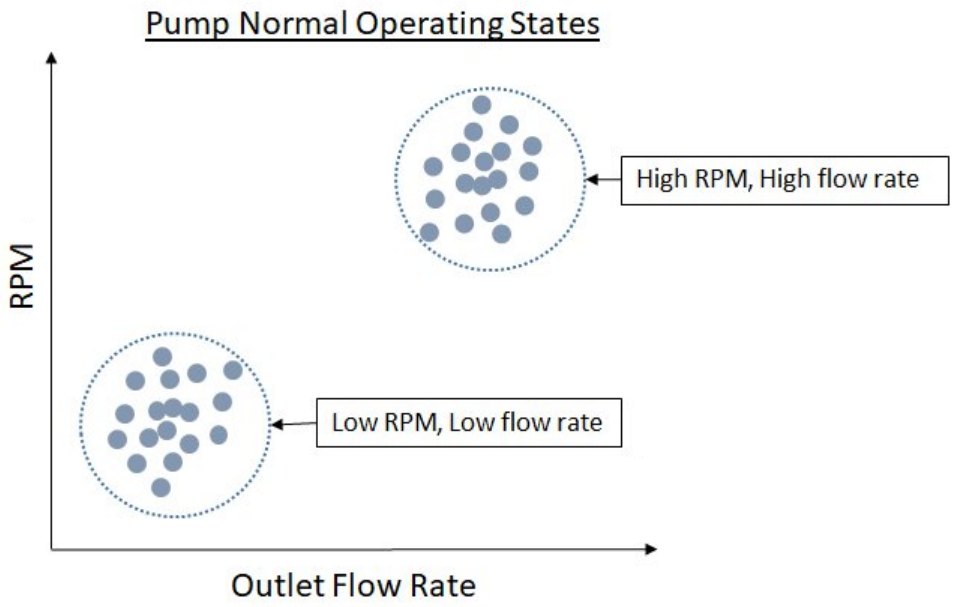
In a simplified form, this fluid pump consists of three major components and their sensors. Note that this is just an example and not a complete list of features and components of such a pump.

1. **Motor** – The motor converts electricity into mechanical rotation. Key sensors might measure the current, voltage, or revolutions per minute (RPM).
2. **Bearing** – Bearings keep the rotating shaft in position while allowing it to rotate with minimal friction. Key sensors measure vibration.
3. **Pump** – The pump is an impeller that rotates on the shaft and pulls fluid from one direction and forces fluid in another direction, similar to a boat propeller. Key sensors measure inlet and outlet flow rate, pressure, and temperature.

For a simple application of Amazon Lookout for Equipment, let's say that the only available data consists of measurements of how fast the pump is spinning in RPM, and the outlet flow rate of the fluid. The following historical time-series plots show both sets of measurements.

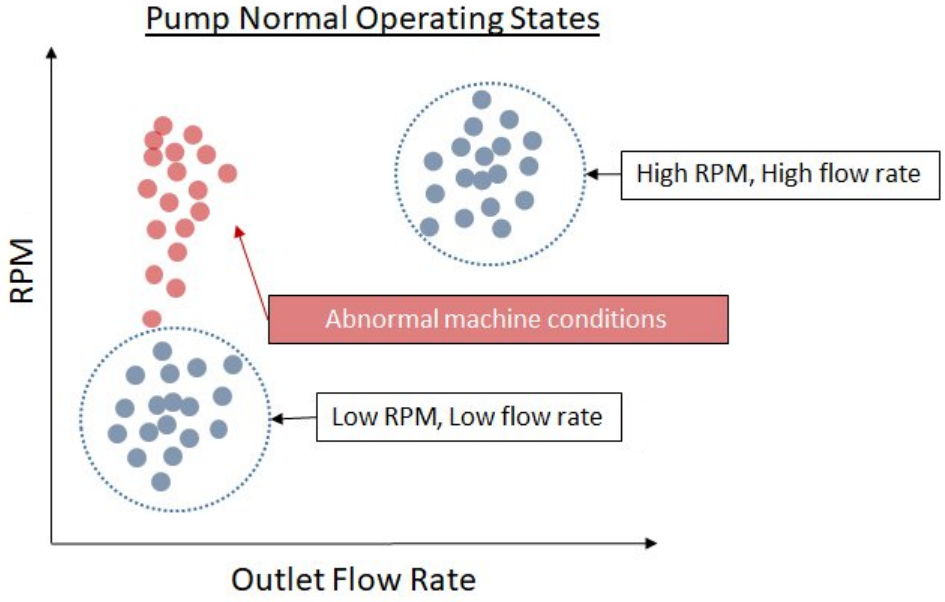


These graphs show the expected relationship between RPM and flow rate: as the pump rotates faster, the fluid flows faster. The graphs show two operating modes: one with low RPM and a low flow rate, and a second mode with high RPM and high flow rate. In this case, Amazon Lookout for Equipment wants to learn this normal relationship in terms of operating modes. .The following graph shows another way to visualize the learned normal operating modes.



The normal behavior of this pump is clear. The operator runs the pump at low RPM and high RPM in order to get a low flow rate or a high flow rate. As the pump continues to run, we expect that the data will continue to fall into one of these two operating modes. However, if the pump starts to have problems, this relationship might not hold true.

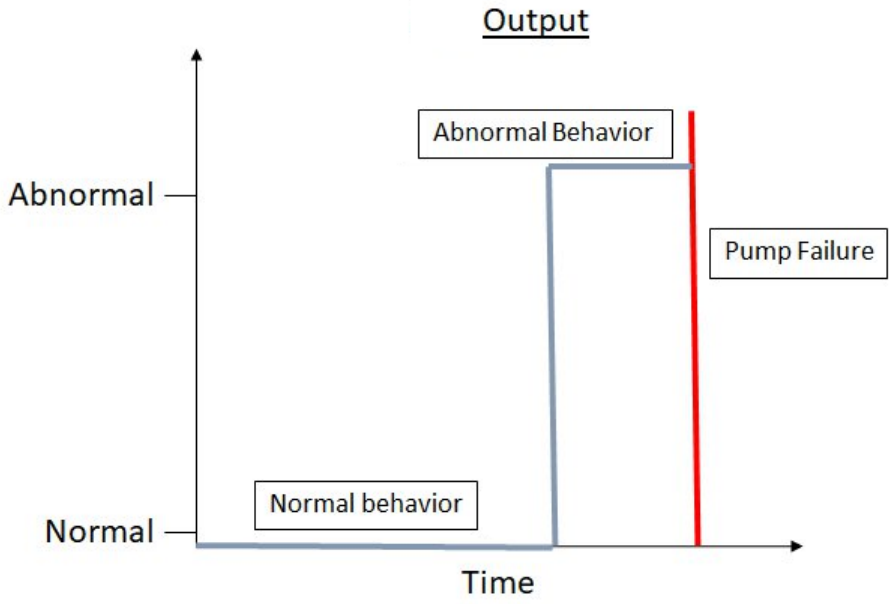
Over time, the impeller (the part similar to a boat propeller) starts to rust, chip, loosen, or become misaligned. As this happens, the data might show abnormal behavior. When the pump rotates at higher a RPM, the flow rate remains low, as shown in the following graph.



These types of issues are precisely what Amazon Lookout for Equipment is designed to detect.

In this case, we see a simple representation of the normal operating states of the pump and abnormal behavior if the pump has an issue. The following graph shows a simplified view of how Lookout for

Equipment detects the output over time. When the relationship between RPM and flow rate is normal, Lookout for Equipment detects that everything is normal. However, as the RPM increases but the flow rate stays the same, Lookout for Equipment starts detecting abnormal behavior. The vertical red line denotes the potential failure point for the pump, at which unplanned downtime occurs.



This is a very simple example of a straightforward application with only two inputs (RPM and flow rate) that have a direct linear relationship with each other. The situation become dramatically more complex when we add additional inputs, such as pressure, temperature, motor current, motor voltage, bearing vibration, and so on. The more you increase the number of inputs, the more complex the relationships between all of the inputs becomes. With some equipment, the number of inputs can easily reach into the hundreds. In addition, this simplified example doesn't attempt to represent the time-series aspect of the problem—the model also has to learn the changes in relationships over time. For example, even subtle changes in vibration over time can be critical to detecting issues.

Amazon Lookout for Equipment works with up to 300 inputs at once. Keep in mind that to accurately analyze the data, Lookout for Equipment requires that the inputs are related to issues that you want to find, and that the historical data used to train (and evaluate) the model represents the equipment's normal behavior.

Setting up Lookout for Equipment

To set up Lookout for Equipment, open an AWS account and create an AWS Identity and Access Management (IAM) user. Then download and configure the AWS Command Line Interface (AWS CLI).

Topics

Step 1: Set up an AWS account and IAM user

Before you can start with Lookout for Equipment, you must sign up for an AWS account if you don't already have one and create an AWS Identity and Access Management (IAM) user.

> **Important**
> Lookout for Equipment provides a management console, which you can use for most tasks. Lookout for Equipment also provides APIs to upload data collected by your existing sensors and build, train, and deploy machine learning models using that data. You must use the AWS Command Line Interface (AWS CLI) to use these operations or write code. For example, to upload samples of normal and abnormal sensor data so that Lookout for Equipment can create a custom model to find potential failures, you must either use the AWS CLI or write code to make requests, using either the APIs directly or by using the AWS SDKs.

Topics

Sign up for AWS

When you sign up for Amazon Web Services (AWS), your AWS account is automatically signed up for all AWS services, including Lookout for Equipment.

If you already have an AWS account, skip to the next topic.

To create an AWS account

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

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

Record your AWS account ID because you'll need it for the next task.

Create an IAM user

Services in AWS, such as Lookout for Equipment, require that you provide credentials when you access them so that the service can determine whether you have permissions to access the resources owned by

that service. The console requires your password. You can create access keys for your AWS account to access the AWS CLI or API.

However, we don't recommend that you access AWS using the credentials for your AWS account. Instead, we recommend that you use AWS Identity and Access Management (IAM). Create an IAM user, add the user to an IAM group with administrative permissions, and then grant administrative permissions to the IAM user that you created. You can then access AWS using a special URL and that IAM user's credentials.

If you signed up for AWS, but you haven't created an IAM user for yourself, you can create one using the IAM console.

To create an administrator user and sign in to the console

1. Create an administrator user called `adminuser` in your AWS account. For instructions, see Creating Your First IAM User and Administrators Group in the *IAM User Guide*.
2. A user can sign in to the AWS Management Console using a special URL. For more information, How Users Sign In to Your Account in the *IAM User Guide*.

You can use this same process to create more groups and users and to give your users access to your AWS account resources.

For more information about IAM, see the following:

- AWS Identity and Access Management (IAM)
- Getting started
- IAM User Guide

Next step

Step 2: Set up IAM Access Roles for Lookout for Equipment (p. 9)

Step 2: Set up IAM Access Roles for Lookout for Equipment

When you create a data source or set up an inference scheduler, Lookout for Equipment needs access to the AWS resources required to create that Lookout for Equipment resource. You must create an AWS Identity and Access Management (IAM) permissions policy before you create the Lookout for Equipment resource. When you call the operation, you must provide the Amazon Resource Name (ARN) of a role with permissions to perform that operation. For example, if you are calling the CreateInferenceScheduler (p. 81) operation, it requires an Amazon S3 bucket for both the input data and for the output data. You would need to provide Lookout for Equipment with a role with a permissions policy to access the bucket.

The AWS console enables you to create a new IAM role to match your specific needs.

Create an IAM role

An IAM role is an IAM identity that you can create in your account that has specific permissions. It's similar to an IAM user in that it's an AWS identity with permission policies that determine what the identity can and cannot do in AWS. However, instead of being uniquely associated with one person, a role can be assumed by anyone in your organization who needs it. As well, a role does not have standard long-term credentials such as a password or access keys associated with it. Instead, when you assume a role, AWS provides you with temporary security credentials for your role session.

Two types of access are discussed here:

- AWS Management Console access, which is used primarily for accessing the Lookout for Equipment console.
- SageMaker, which is a managed service that can be used to easily access Lookout for Equipment programmatically when making API calls.

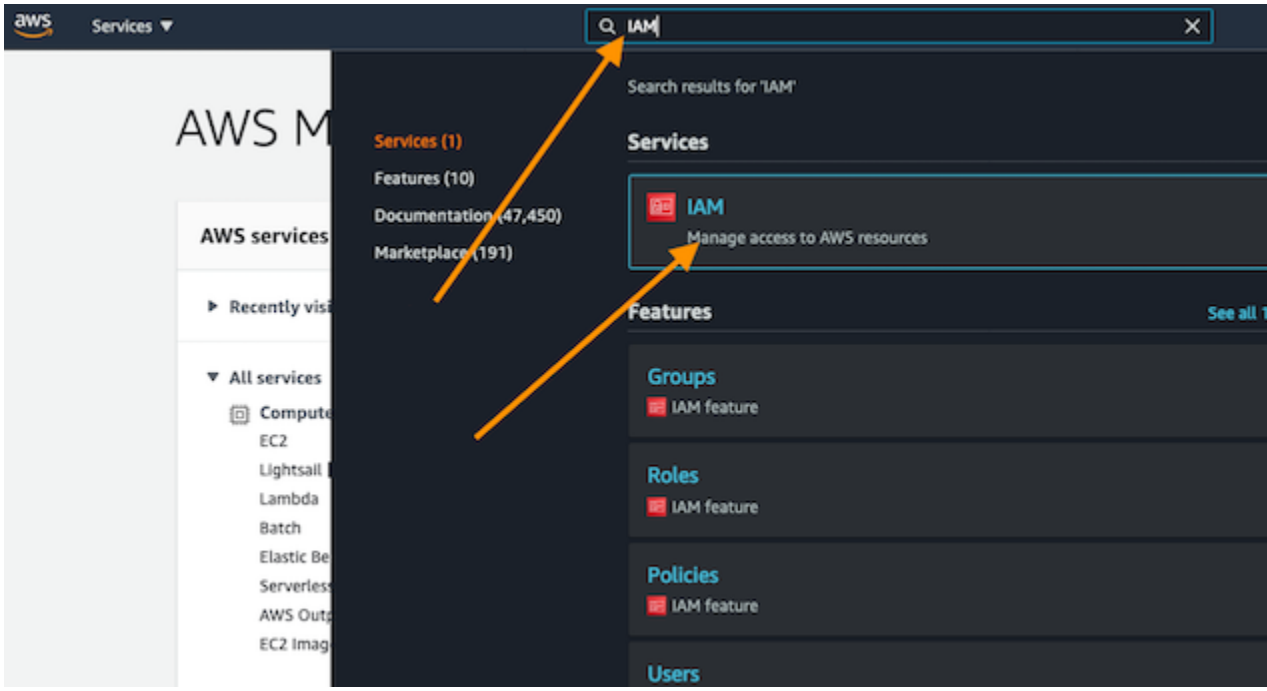
AWS Management Console access: The console always uses your original credentials to authorize a switch to an IAM role. This applies whether you sign in as an IAM user, as a SAML-federated role, or as a web-identity federated role. For example, if you switch to Role A, IAM uses your original user credentials (or those of your federated role) to determine whether you're allowed to assume Role A. If you then switch to Role B while you're using Role A, the console still uses your original user credentials to authorize the switch, not the credentials for Role A.

SageMaker: Amazon SageMaker is a managed service, so it performs operations on your behalf using the hardware that's managed by SageMaker. It can only perform operations that the user permits. In other words, a SageMaker user can grant permissions with an IAM role where the user then passes that role when making an API call. For example, if you want to use the CreateModel (p. 86) operation, you need to pass a `RoleArn`, which is the ARN of an IAM role with permissions to the data source from which the model is to be created.

The following shows you how to create an IAM role to delegate access to Lookout for Equipment from either the AWS Management Console or from an Amazon SageMaker instance.

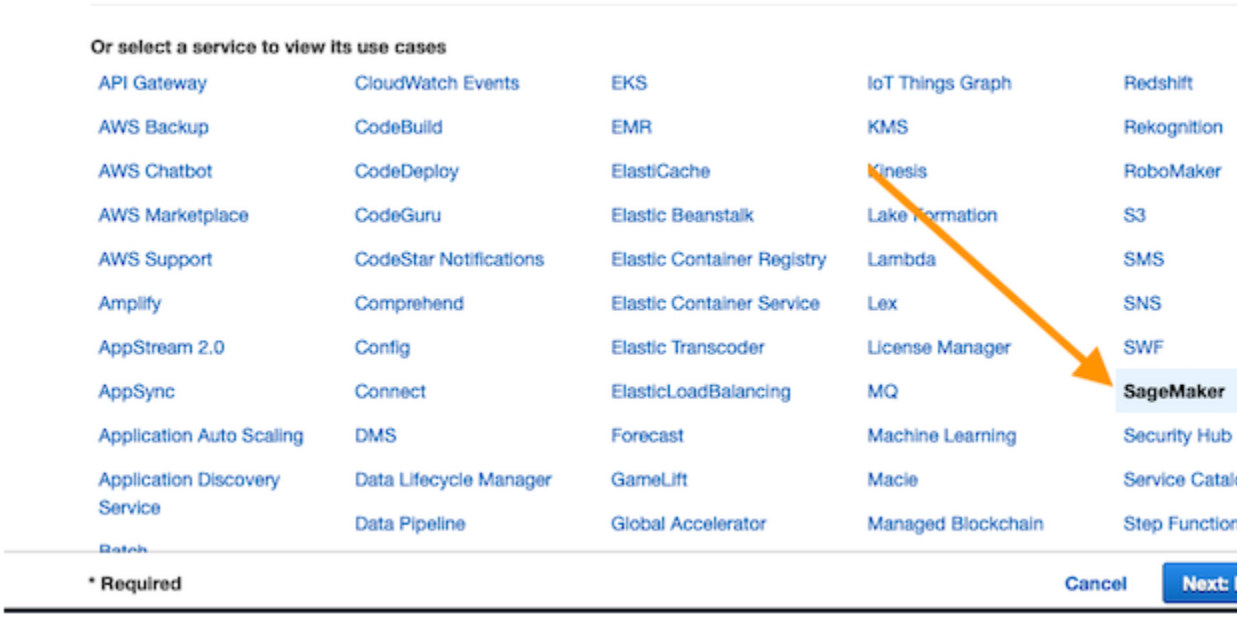
To create an IAM role

1. Open the Lookout for Equipment console at https://console.aws.amazon.com/lookoutforequipment.
2. Enter IAM in the search bar and then choose IAM Service from the results.

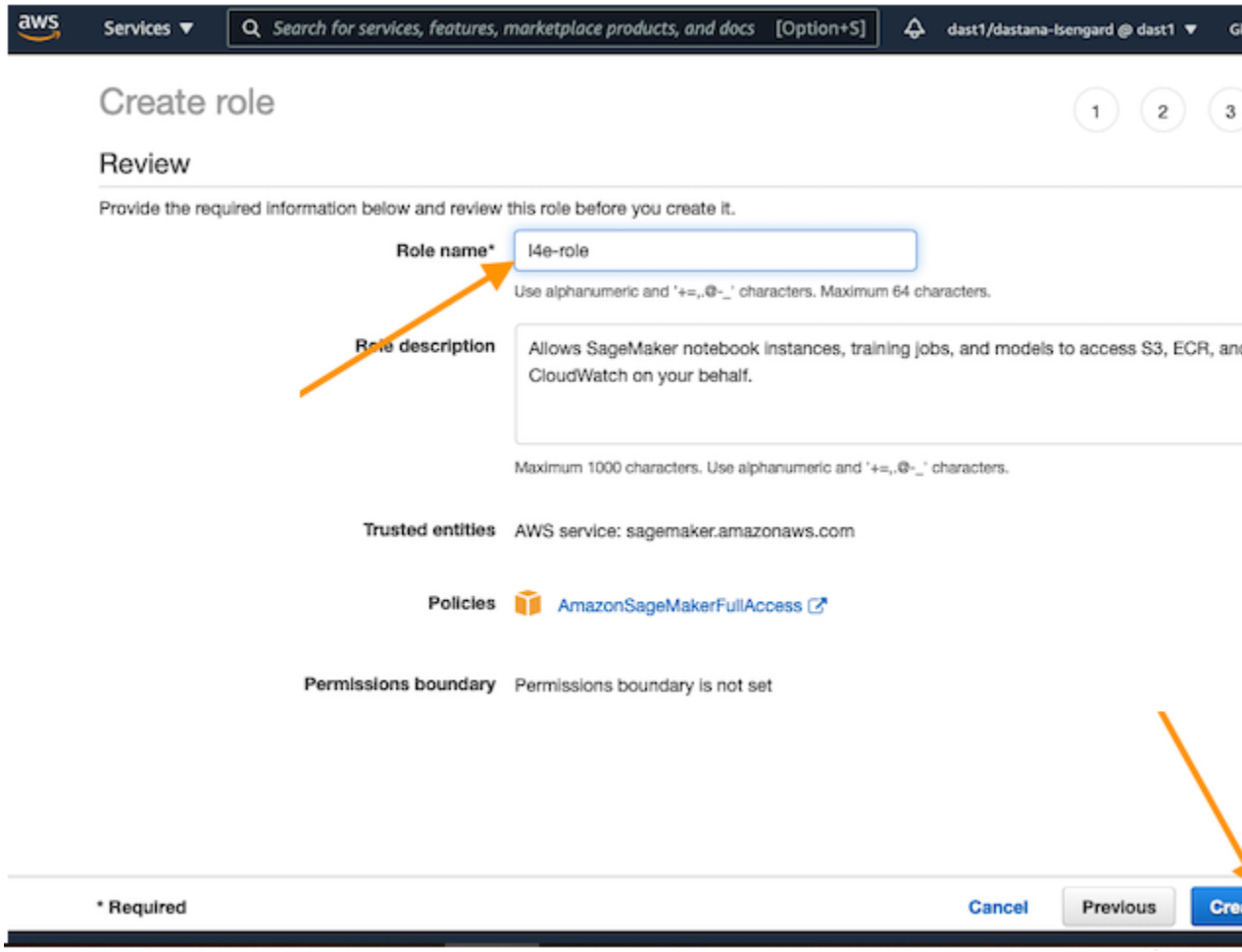


3. Under **Access management** on the left navigation pane, choose **Roles**.
4. Choose **Create role**.
5. Under **Select type of trusted entity**, choose **AWS service**.

6. From the list of services, choose **SageMaker**, and then choose **Next: Permissions**.



7. Choose **Next: Tags**.
8. Choose **Next: Review**.
9. For **Role name**, enter *l4e-role*, and then choose **Create Role**.

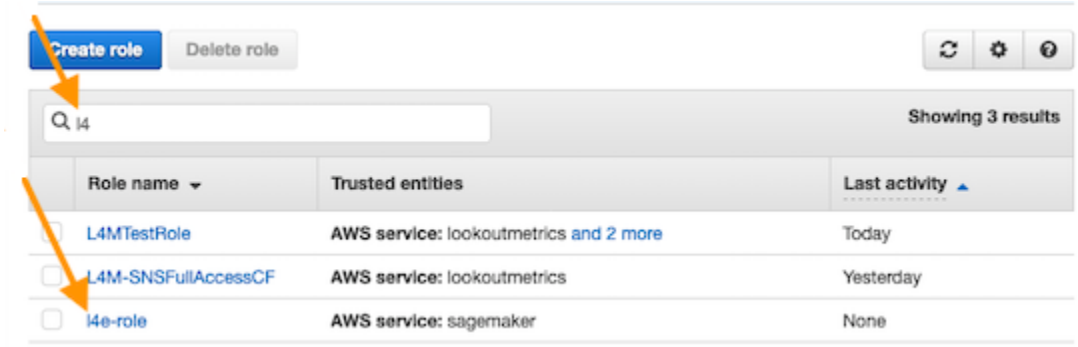


Attach policies to the created IAM role

You now need to attach the access policies that allow Lookout for Equipment to access other required AWS services.

To attach policies to the created IAM role

1. Search for the created IAM role in the search bar and choose it from the returned results.

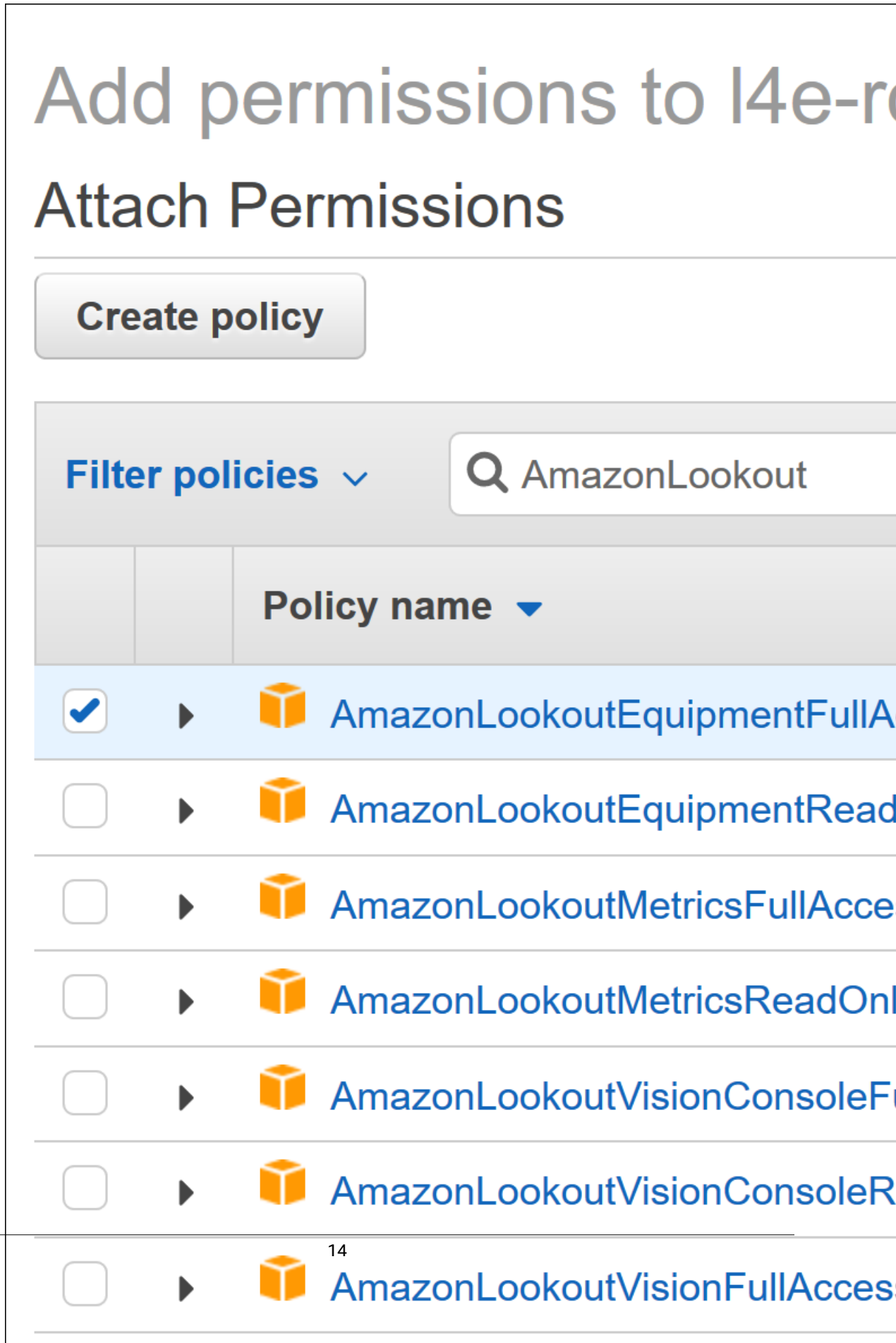


2. Choose **Attach policies**.

3. Search for the following additional managed policies and select them from the returned results:

   - *AmazonS3FullAccess*
   - *IAMFullAccess*
   - *AmazonLookoutEquipmentFullAccess*

4. Choose **Attach policy**.


Add permissions to l4e-r
Attach Permissions
Create policy
Filter policies
AmazonLookout
Policy name
AmazonLookoutEquipmentFullA
AmazonLookoutEquipmentRead
AmazonLookoutMetricsFullAcce
AmazonLookoutMetricsReadOnl
AmazonLookoutVisionConsoleFu
AmazonLookoutVisionConsoleR
AmazonLookoutVisionFullAcces

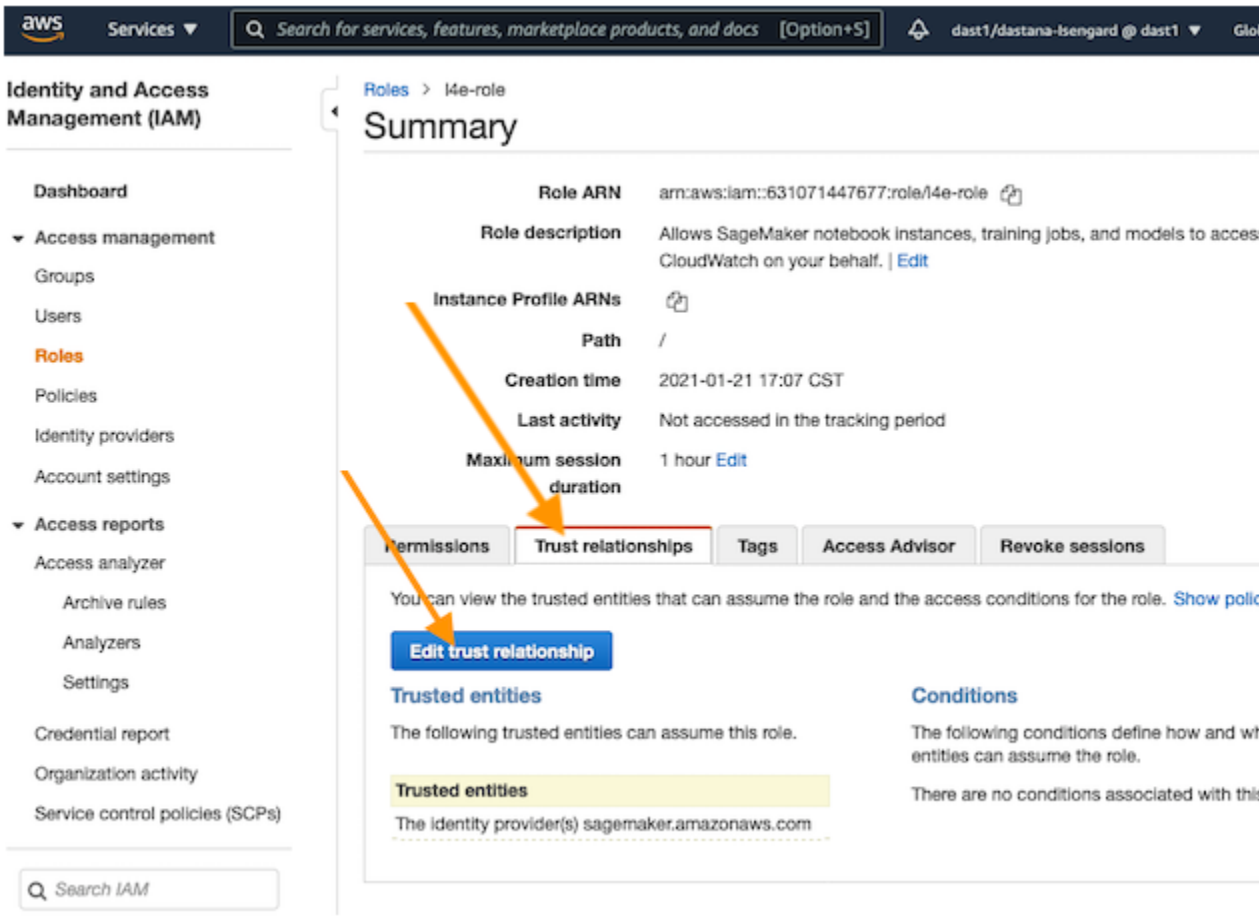
Edit the trust relationship for the created IAM role

A trust relationship defines what entities can assume the role that you've created. When you created the role, you chose SageMaker as a trusted entity. The same role can also be used for Lookout for Equipment Console access. Modify the role so that the trusted relationship is between your AWS account and Lookout for Equipment.

To edit the trust relationship of your created IAM role

Under **Access management** on the left navigation pane, choose **Roles**.

1. Search for the created IAM role in the search bar and choose it from the returned results.



2. On the **Trust relationships** tab, choose **Edit trust relationship**.
3. Under **Policy Document**, paste the following policy.

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

```
      "Effect": "Allow",
      "Principal": {
        "Service": "sagemaker.amazonaws.com"
      },
      "Action": "sts:AssumeRole"
    }
  ]
}
```

Next step

Step 3: Set up the AWS Command Line Interface (AWS CLI) (p. 16)

Step 3: Set up the AWS Command Line Interface (AWS CLI)

To set up the AWS CLI

1. Download and configure the AWS CLI. For instructions, see the following topics in the *AWS Command Line Interface User Guide*:

   - Getting Set Up with the AWS Command Line Interface
   - Configuring the AWS Command Line Interface

2. In the AWS CLI `config` file, add a named profile for the administrator user.

   ```
   [profile adminuser]
   aws_access_key_id = adminuser access key ID
   aws_secret_access_key = adminuser secret access key
   region = aws-region
   ```

   You use this profile when running the AWS CLI commands. For more information about named profiles, see Named Profiles in the *AWS Command Line Interface User Guide*. For a list of AWS Regions, see Regions and Endpoints in the *Amazon Web Services General Reference*.

3. Verify the setup by typing the following help command at the command prompt.

   ```
   aws help
   ```

Transferring historical data into Amazon Lookout for Equipment

To transfer your data into Amazon Lookout for Equipment, you must:

- Properly format your data in .csv files.
- Create a dataset from the .csv files.
- Ingest the dataset into Amazon Lookout for Equipment.

By following the preceding steps, you can train your model on a properly formatted dataset. You can use the model that you've trained to monitor your equipment in real time.

Topics

Next step

Formatting your data (p. 17)

Formatting your data

To monitor your equipment, you must provide Amazon Lookout for Equipment with time-series data from the sensors on your equipment. The data that you're providing to Lookout for Equipment is a series of numerical measurements from the sensors. You provide this data from either a data historian or Amazon Simple Storage Service (Amazon S3). A data historian is a software program that records and retrieves sensor data from your equipment.

To provide Amazon Lookout for Equipment with time-series data from the sensors, you must use properly formatted .csv files to create a dataset. Creating a dataset aggregates the data in a format that is suitable for analysis. You create a dataset for a single piece of equipment, or *asset*. You train an ML model on the dataset that you create. You then use that model to monitor your asset. You don't have to use all the data from the sensors to train a model. You train a model using data from some of the sensors in the dataset.

You can store the data for your asset in one of the following ways:

- Recommended: Using one .csv file for each sensor
- Storing all of the sensor data in one .csv file

Each .csv file must have at least two columns. The first column of the file is a timestamp that indicates the date and time. You must have at least one additional column containing the data from a sensor. Each subsequent column can have data from a different sensor.

You must have a double (numerical) as the data type for your sensor data. You can only train your model on numeric data.

When you are preparing your data, you should keep the following information in mind:

- The data across all of your .csv files must span at least 180 days. For example, you can have two .csv files, with one file having data from January to April, and the other having data from May to August.
- You can create a dataset with up to 3,000 sensors, but you can train a model on up to 300 sensors.
- The maximum length of a sensor name is 200 characters.
- The size of a .csv file can't exceed 5 GB. If you want to create a dataset greater than 5 GB, you must use multiple .csv files.
- The files that you use to create a dataset can't exceed 50 GB in total.
- You can use up to 1,000 files to create a dataset.
- You can use the following delimiters for the data in the timestamp column:
  - '-'
  - '_'
  - ' '

    Note
    Quotation marks are used around the delimiters to make them easier to read.
- The timestamp column can use the following formats:
  - yyyy-MM-dd-HH-mm-ss
  - yyyy-MM-dd'T'HH:mm:ss
  - yyyy-MM-dd HH:mm:ss
  - yyyy-MM-dd-HH:mm:ss
  - yyyy/MM/dd'T'HH:mm:ss
  - yyyy/MM/dd HH:mm:ss
  - yyyy MM dd'T'HH:mm:ss
  - yyyy MM dd HH:mm:ss
  - yyyyMMdd'T'HH:mm:ss
  - yyyyMMdd HH:mm:ss
  - yyyyMMddHHmmss
  - yyyy-MM-dd'T'HH:mm
  - yyyy-MM-dd HH:mm
  - yyyy-MM-dd-HH:mm
  - yyyy/MM/dd'T'HH:mm
  - yyyy/MM/dd HH:mm
  - yyyy MM dd'T'HH:mm
  - yyyy MM dd HH:mm
  - yyyyMMdd'T'HH:mm
  - yyyyMMdd HH:mm
  - yyyyMMddHHmm
- The valid characters that you can use in the column names of the dataset are `0` to `9`, `a` to `z`, `A` to `Z`, `.`, `\ _`,and `-`.

You can use label data to highlight any part of your dataset where your asset functioned abnormally. For more information, see the section called “Label data” (p. 20).

The following examples show you the different ways that you can format a .csv file.

Formatting your data with one .csv file for each sensor

If your are storing the data from each sensor in one .csv file, use the following table to see how to format the data.

SensorData.csv

Timestamp	Sensor 3
1/1/2020 0:00	34
1/1/2020 0:05	33
1/1/2020 0:10	35
1/1/2020 0:15	33
1/1/2020 0:20	34

The following example shows the information from the preceding table as a .csv file.

```
Timestamp,Sensor 3
1/1/2020 0:00,34
1/1/2020 0:05,33
1/1/2020 0:10,35
1/1/2020 0:15,33
1/1/2020 0:20,34
```

We recommend using `"Timestamp"` as the name for the column with the time-series data. For the column with data from the sensor, we recommend using a name that distinguishes it from other sensors.

Formatting all the data for an asset in one .csv file

To store the data for your asset in one .csv file, you arrange the data in the following format.

AssetData.csv

Timestamp	Sensor 1	Sensor 2	Sensor 3
1/1/2020 0:00	12	50	34
1/1/2020 0:05	11	49	33
1/1/2020 0:10	10	48	35
1/1/2020 0:15	9	47	33
1/1/2020 0:20	12	50	34

The following example shows the information from the preceding table as a .csv file.

```
Timestamp,Sensor 1,Sensor 2
1/1/2020 0:00,2,12
1/1/2020 0:05,3,11
1/1/2020 0:10,5,10
1/1/2020 0:15,3,9
1/1/2020 0:20,4,12
```

You can choose your column names. We recommend using `"Timestamp"` as the name for the column with the time-series data. For the names of the columns with data from your sensors, we recommend using names that distinguish one sensor from another.

Label data

If you have them available, we recommend using labels for abnormal equipment behavior in your data. These labels could be applied to periods when the equipment did not function properly. You store the label data as a .csv file that consists of two columns. The file has no header. The first column has the start time of the abnormal behavior. The second column has the end time.

The following example shows how your label data should appear as a .csv file.

```
2020-02-01T20:00:00.000000,2020-02-03T00:00:00.000000
2020-07-01T20:00:00.000000,2020-07-03T00:01:00.000000
```

Next step

Uploading your data into Amazon S3 (p. 20)

Uploading your data into Amazon S3

Amazon Lookout for Equipment needs to access your .csv files to create a dataset. You must store the files in the correct location and provide Lookout for Equipment with the information that it needs to access them.

To understand how to properly store your .csv files, you need to be familiar with the following Amazon Simple Storage Service (Amazon S3) concepts:

- Bucket – A container to store your objects.
- Object – The entity stored in the S3 bucket. In this case, it's your training or tuning text files.
- Key – The unique identifier for an object within a bucket.
- Prefix – Any portion of a key up to the final delimiter. You use prefixes to organize your data and specify objects in the S3 bucket.

To upload your data, you must first create an Amazon S3 bucket. To learn how to create an S3 bucket, see Create your first S3 bucket.

Amazon S3 uses prefixes to organize the storage of your files. A prefix is a logical grouping of objects in a bucket that is separated by a delimiter. The prefix value is similar to a directory name that you use to store similar data under the same directory in a bucket. For more information about prefixes, see Organizing objects using prefixes.

The Amazon S3 console supports the folder concept to help you see how prefixes organize your data. When you create an additional folder, you modify the prefix of your object key by adding another

delimiter. To learn more about using folders, see Organizing objects in the Amazon S3 console using folders.

The Amazon S3 locations that you use to store your .csv files and the prefixes that Amazon Lookout for Equipment uses to access them depend on how your data is organized. For more information, see the following sections.

If the sensor data for your asset is stored in multiple .csv files

You use the following prefix to store the data for each sensor on your asset.

`s3://DOC-EXAMPLE-BUCKET/AssetName/SensorName/SensorName.csv`

If you had sensors on your asset with the following sensor names:

- Sensor1
- Sensor2
- Sensor3

You would use the following objects keys for the sensors:

- `s3://DOC-EXAMPLE-BUCKET/AssetName/Sensor1/Sensor1.csv`
- `s3://DOC-EXAMPLE-BUCKET/AssetName/Sensor2/Sensor2.csv`
- `s3://DOC-EXAMPLE-BUCKET/AssetName/Sensor3/Sensor3.csv`

The object key `s3://DOC-EXAMPLE-BUCKET/AssetName/Sensor1/Sensor1.csv` has the following prefixes:

- `s3://DOC-EXAMPLE-BUCKET/AssetName/`
- `s3://DOC-EXAMPLE-BUCKET/AssetName/Sensor1/`

By using the `s3://DOC-EXAMPLE-BUCKET/AssetName/` prefix, you can access Sensor1.csv, Sensor2.csv, and Sensor3.csv.

The following procedure shows you how to use the Amazon S3 console to upload a .csv file containing data from one sensor to an Amazon S3 bucket.

To upload a .csv file of one sensor to an Amazon S3 bucket

1. Sign in to the AWS Management Console and open the Amazon S3 console at https://console.aws.amazon.com/s3/.
2. In the **Buckets** list, choose the name of the bucket that you've created to store the data for your sensor.
3. Choose **Create folder**.
4. Specify the name of the asset as the folder name (for example, Pump1). Then choose **Create folder**.
5. Choose the folder with the name of the asset.
6. Choose **Create folder**.
7. Specify the name of the sensor as the folder name (for example, Sensor1).
8. Choose the folder with the sensor name.
9. Choose **Upload**. Then choose **Add files**.
10. Choose the .csv file containing your sensor data.

You've successfully uploaded your data. To learn how to use a schema to create a dataset from your .csv files, see Creating a dataset from multiple .csv files (p. 23).

If the sensor data for your asset is stored in one .csv file

You use the following object key structure to store the data for each sensor on your asset:

`s3://DOC-EXAMPLE-BUCKET/FacilityName/AssetName/AssetName.csv`

For `FacilityName`, you specify the location of your asset. For example, `Powerplant1` or `Factory1` could be facility names.

The object key `s3://DOC-EXAMPLE-BUCKET/FacilityName/AssetName/AssetName.csv` has the following prefixes:

- `s3://DOC-EXAMPLE-BUCKET/FacilityName/`
- `s3://DOC-EXAMPLE-BUCKET/FacilityName/AssetName/`

When you ingest your model, you use the `s3://DOC-EXAMPLE-BUCKET/FacilityName/` prefix to access the .csv file.

To upload a .csv file of one sensor to an Amazon S3 bucket

1. Sign in to the AWS Management Console and open the Amazon S3 console at https://console.aws.amazon.com/s3/
2. In the **Buckets** list, choose the name of the bucket that you've created to store the data for your sensor.
3. Choose **Create folder**.
4. Specify the name of the facility as the folder name (for example, Powerplant1). Then choose **Create folder**.
5. Choose the folder with the name of the facility.
6. Choose **Create folder**.
7. Specify the name of the asset as the folder name (for example, Motor1).
8. Choose the folder with the asset name.
9. Choose **Upload**. Then choose **Add files**.
10. Choose the .csv file containing the sensor data from your asset.

You've successfully uploaded your data. To learn how to use a schema to create a dataset from your .csv files, see Creating a dataset from one .csv file (p. 26).

Uploading label data (optional)

After you upload the sensor data, you can upload label data to highlight any abnormal equipment behavior. The following is an Amazon S3 prefix that you should use for your label data:

`s3://DOC-EXAMPLE-BUCKET1/label-data`

The following are example S3 locations where you would store the .csv files of your label data:

`s3://DOC-EXAMPLE-BUCKET1/label-data/label-data01.csv`

`s3://DOC-EXAMPLE-BUCKET1/label-data/label-data02.csv`

After you've uploaded the files, you can create a dataset from your .csv files. For more information, see Creating a dataset in Amazon Lookout for Equipment (p. 23).

Next step

Creating a dataset in Amazon Lookout for Equipment (p. 23)

Creating a dataset in Amazon Lookout for Equipment

After you've uploaded your data to Amazon Simple Storage Service (Amazon S3) using the correct prefixes and object keys, you create a dataset from them. A dataset is a container for all of the data that you've uploaded. You train the machine learning model from Amazon Lookout for Equipment on this dataset.

To create a dataset from your .csv files, you create a schema for both the files and the fields within them. A schema is the organizational framework that Amazon Lookout for Equipment applies to your data. Lookout for Equipment uses this schema to store all of the data in one dataset.

The schema that you define depends on how you've stored the data. Use one of the following sections to help you create a schema.

- Creating a dataset from multiple .csv files (p. 23)
- Creating a dataset from one .csv file (p. 26)

Creating a dataset from multiple .csv files

If you've uploaded multiple .csv files, with each sensor having its own .csv file, you would use the following schema to create a dataset from those files.

```
{
    "Components": [
        {
            "ComponentName": "Sensor1",
            "Columns": [
                {
                    "Name": "Timestamp",
                    "Type": "DATETIME"
                },
                {
                    "Name": "Sensor1",
                    "Type": "DOUBLE"
                }
            ]
        },
        {
            "ComponentName": "Sensor2",
            "Columns": [
                {
                    "Name": "Timestamp",
                    "Type": "DATETIME"
                },
                {
                    "Name": "Sensor2",
                    "Type": "DOUBLE"
                }
            ]
        },
        {
```

```
            "ComponentName": "Sensor3",
            "Columns": [
                {
                    "Name": "Timestamp",
                    "Type": "DATETIME"
                },
                {
                    "Name": "Sensor3",
                    "Type": "DOUBLE"
                }
            ]
        },
        {
            "ComponentName": "Sensor4",
            "Columns": [
                {
                    "Name": "Timestamp",
                    "Type": "DATETIME"
                },
                {
                    "Name": "Sensor4",
                    "Type": "DOUBLE"
                }
            ]
        }
    ]
}
```

In the preceding schema, `Components` refers to a collection of identifiers for the .csv files of your sensors. The `ComponentName` is the portion of a prefix of an Amazon S3 object key that identifies a .csv file. The following examples show you how the values specified for `ComponentName` access the .csv files you've stored in your Amazon S3 buckets:

- `"ComponentName: "Sensor1"` accesses `s3://`*DOC-EXAMPLE-BUCKET*`/`*AssetName*`/`*Sensor1*`/`*Sensor1.csv*
- `"ComponentName: "Sensor2"` accesses `s3://`*DOC-EXAMPLE-BUCKET*`/`*AssetName*`/`*Sensor2*`/`*Sensor2.csv*
- `"ComponentName: "Sensor3"` accesses `s3://`*DOC-EXAMPLE-BUCKET*`/`*AssetName*`/`*Sensor3*`/`*Sensor3.csv*
- `"ComponentName: "Sensor4"` accesses `s3://`*DOC-EXAMPLE-BUCKET*`/`*AssetName*`/`*Sensor4*`/`*Sensor4.csv*

You define a `Columns` object for each `ComponentName` that you define in the schema. The `Name` fields in the `Columns` object must match the columns in your .csv files.

Within each `Columns` object, the `Name` fields that reference the columns containing the timestamp data must have the `Type` field specified as `DATETIME`. The `Name` fields that reference your sensor data must have a `Type` of `DOUBLE`.

You can use a schema to create a dataset for your .csv files in the Amazon Lookout for Equipment console, but we recommend using the API. You can use the following example code with the AWS SDK for Python (Boto3) to create a dataset.

```
import boto3
import json
import pprint
from botocore.config import Config
config = Config(
```

```
    region_name = 'Region' # Choose a valid AWS Region
)
lookoutequipment = boto3.client(service_name="lookoutequipment", config=config)
dataset_schema = {
    "Components": [
        {
            "ComponentName": "Sensor1",
            "Columns": [
                {
                    "Name": "Timestamp",
                    "Type": "DATETIME"
                },
                {
                    "Name": "Sensor1",
                    "Type": "DOUBLE"
                }
            ]
        },
        {
            "ComponentName": "Sensor2",
            "Columns": [
                {
                    "Name": "Timestamp",
                    "Type": "DATETIME"
                },
                {
                    "Name": "Sensor2",
                    "Type": "DOUBLE"
                }
            ]
        },
        {
            "ComponentName": "Sensor3",
            "Columns": [
                {
                    "Name": "Timestamp",
                    "Type": "DATETIME"
                },
                {
                    "Name": "Sensor3",
                    "Type": "DOUBLE"
                }
            ]
        },
        {
            "ComponentName": "Sensor4",
            "Columns": [
                {
                    "Name": "Timestamp",
                    "Type": "DATETIME"
                },
                {
                    "Name": "Sensor4",
                    "Type": "DOUBLE"
                }
            ]
        }
    ]
}
dataset_name = "dataset-name"
data_schema = {
    'InlineDataSchema': json.dumps(dataset_schema),
}
create_dataset_response = lookoutequipment.create_dataset(DatasetName=dataset_name,
 DatasetSchema=data_schema)
pp = pprint.PrettyPrinter(depth=4)
```

```
pp.pprint(create_dataset_response)
```

Next step

Ingesting a dataset (p. 27)

Creating a dataset from one .csv file

If you've uploaded one .csv file containing all of the sensor data for the asset, you would use the following schema to create a dataset from that file.

```
{
  "Components": [
    {
      "ComponentName": "AssetName",
      "Columns": [
        {
          "Name": "Timestamp",
          "Type": "DATETIME"
        },
        {
          "Name": "Sensor1",
          "Type": "DOUBLE"
        },
        {
          "Name": "Sensor2",
          "Type": "DOUBLE"
        },
        {
          "Name": "Sensor3",
          "Type": "DOUBLE"
        },
        {
          "Name": "Sensor4",
          "Type": "DOUBLE"
        },
      ]
    }
  ]
}
```

The `"ComponentName"` is the portion of the prefix of the Amazon S3 object key that identifies the .csv file containing the sensor data for your asset. When you specify the value of `"ComponentName"` as `"AssetName"`, you access `s3://`*`DOC-EXAMPLE-BUCKET`*`/`*`FacilityName`*`/`*`AssetName`*`/`*`AssetName.csv`*. You enter the columns of your dataset in the `Columns` object. The name of each column in your .csv file must match the `Name` in the schema. For the column containing the time stamp data, you must specify the value of `"Type"` as `"DATETIME"` in the schema. For the columns containing data from sensors, you must specify the value of `"Type"` as `"DOUBLE"`.

You can use a schema to create a dataset for your .csv files in the Amazon Lookout for Equipment console, but we recommend using the API. You can use the following example code using the AWS SDK for Python (Boto3) to create a dataset.

```
import boto3
import json
```

```
import pprint
from botocore.config import Config
config = Config(
    region_name = 'Region' # Choose a valid AWS Region.
)
lookoutequipment = boto3.client(service_name="lookoutequipment", config=config)
dataset_schema = {
  "Components": [
    {
      "ComponentName": "AssetName",
      "Columns": [
        {
          "Name": "Timestamp",
          "Type": "DATETIME"
        },
        {
          "Name": "Sensor1",
          "Type": "DOUBLE"
        },
        {
          "Name": "Sensor2",
          "Type": "DOUBLE"
        },
        {
          "Name": "Sensor3",
          "Type": "DOUBLE"
        },
        {
          "Name": "Sensor4",
          "Type": "DOUBLE"
        },
      ]
    }
  ]
}
dataset_name = "dataset-name"
data_schema = {
    'InlineDataSchema': json.dumps(dataset_schema),
}
create_dataset_response = lookoutequipment.create_dataset(DatasetName=dataset_name,
 DatasetSchema=data_schema)
pp = pprint.PrettyPrinter(depth=4)
pp.pprint(create_dataset_response)
```

Next step

Ingesting a dataset (p. 27)

Ingesting a dataset

Amazon Lookout for Equipment requires you to create a dataset from your .csv files containing your sensor data and the schema that you've provided for those files. For more information about using a schema to create a dataset, see Creating a dataset in Amazon Lookout for Equipment (p. 23).

To convert the dataset into a format that is suitable for analysis, you must ingest it. Ingesting a dataset imports it into Amazon Lookout for Equipment and lets you train a machine learning model on it. To ingest your data, you start the data ingestion step and specify the Amazon Simple Storage Service (Amazon S3) location that contains your sensor data.

You can use one the following procedures to ingest a dataset.

Console

To use the console to ingest a dataset, choose the Amazon S3 prefix that selects all of the sensor data for the asset.

To ingest a dataset (console)

1. Sign in to AWS Management Console and open the Amazon Lookout for Equipment console at Amazon Lookout for Equipment console.
2. Choose a dataset that you've created.
3. Choose **Ingest data**.
4. For **S3 location**, provide the Amazon S3 prefix for all of the sensor data for the asset.
   a. If you created the dataset from multiple .csv files, with each file containing data from one sensor, you would use the prefix: **s3://*DOC-EXAMPLE-BUCKET1*/AssetName/**.
   b. If you created the dataset from a single .csv file for your asset, you would use the prefix: **s3://*DOC-EXAMPLE-BUCKET1*/FacilityName/**.
5. For **IAM role**, choose a role that provides permissions to access the .csv files you've stored in Amazon S3. If you don't have a role that provides permissions, choose **Create a role**.
6. Choose **Ingest**.

AWS SDK for Python (Boto3)

Use the following AWS SDK for Python (Boto3) example code to ingest your dataset. You must have the modules installed from the code examples that showed you how to create a dataset to successfully use the following code.

```
import boto3
import time
from botocore.config import Config


config = Config(
    region_name = 'Region' #Choose a valid AWS Region
)

lookoutequipment = boto3.client(service_name="lookoutequipment", config=config)


INGESTION_DATA_SOURCE_BUCKET = 'DOC-EXAMPLE-BUCKET1'
# If you're ingesting multiple .csv files of your sensor data, use the following Amazon S3
 path: s3://DOC-EXAMPLE-BUCKET/AssetName/. If you're ingesting a single .csv file of your
 asset data, use the following Amazon S3 path: s3://DOC-EXAMPLE-BUCKET/FacilityName/.
INGESTION_DATA_SOURCE_PREFIX = 'my_data/sensor_readings/'
# The ROLE_ARN and DATASET_NAME values that are used in this script have been defined in
 the previous SDK for Python example code for creating a dataset.

data_ingestion_role_arn = ROLE_ARN
dataset_name = DATASET_NAME
ingestion_input_config = dict()
ingestion_input_config['S3InputConfiguration'] = dict(
    [
        ('Bucket', INGESTION_DATA_SOURCE_BUCKET),
        ('Prefix', INGESTION_DATA_SOURCE_PREFIX)
    ]
)
```

```
# Start data ingestion
start_data_ingestion_job_response = lookoutequipment.start_data_ingestion_job(
    DatasetName=dataset_name,
    RoleArn=data_ingestion_role_arn,
    IngestionInputConfiguration=ingestion_input_config)

data_ingestion_job_id = start_data_ingestion_job_response['JobId']
data_ingestion_status = start_data_ingestion_job_response['Status']

print(f'=====Data Ingestion job is started. Job ID: {data_ingestion_job_id}=====\n')

# Wait until completes
print("=====Polling Data Ingestion Status=====\n")
print("Data Ingestion Status: " + data_ingestion_status)
while data_ingestion_status == 'IN_PROGRESS':
    time.sleep(30)
    describe_data_ingestion_job_response =
 lookoutequipment.describe_data_ingestion_job(JobId=data_ingestion_job_id)
    data_ingestion_status = describe_data_ingestion_job_response['Status']
    print("Data Ingestion Status: " + data_ingestion_status)
print("\n=====End of Polling Data Ingestion Status=====")
```

Managing your datasets

You've created a dataset or multiple datasets that are in a format that Amazon Lookout for Equipment can use to analyze the data from your equipment. You might want to see which datasets you've created, where the data came from, and who created it. You might be able to use that information to determine whether you need to create a new dataset. You can train a model on the new dataset that might monitor your equipment more accurately than the model that you've trained on the dataset you've previously created. To make managing your datasets easier, you can delete the datasets that you no longer need.

For each dataset, you can view the dataset details and the data source. The data source shows you the IAM role ARN (Amazon Resource Number) used to create the dataset. The IAM role ARN shows you which AWS account number and IAM role were used to create the dataset. It also shows you the Amazon Simple Storage Service (Amazon S3) location of the multiple .csv files, or the single .csv file used to create the dataset.

The dataset details include the schema, the organizational framework, used to create the dataset. The dataset details also tell you when the dataset was created and when it was last modified.

For a dataset, the following procedures enable you to view the dataset details, the data source, and delete that dataset.

To manage your datasets (Console)

To manage your datasets (Console)

This procedure uses the Amazon Lookout for Equipment console to first view a data source, then view the dataset details, and then delete a dataset.

1. Sign in to AWS Management Console and open the Amazon Lookout for Equipment console at Amazon Lookout for Equipment console.
2. For **Datasets**, choose a dataset.
3. For **Step 2. Ingest data**, choose View data source.
4. View both the **IAM role ARN** and the **S3 location** on this page.

5. Preceding **Data source**, for Amazon Lookout for Equipment > Datasets > *dataset-name*, choose the name of your dataset.
6. Navigate to the **Dataset details** section, which follows the **How it works** section.
7. To view the data schema, choose **View** under **Data schema**.
8. To delete a dataset, choose **Delete** under **Dataset details**.
9. Type the word "delete" and choose **Delete**.

To manage your datasets (AWS SDK for Python (Boto3))

Use the following AWS SDK for Python (Boto3) example code to manage your datasets. It will show you how to list all your datasets, get information about a dataset, and delete a dataset. You must have the modules installed from code examples that showed you how to create a dataset to successfully use the following code.

To run the following code, you need to have run the example code in either Creating a dataset from multiple .csv files (p. 23) or Creating a dataset from one .csv file (p. 26).

```
import boto3
import json
import pprint
from botocore.config import Config


config = Config(
    region_name = 'Region' # Choose a valid AWS Region
)

lookoutequipment = boto3.client(service_name="lookoutequipment", config=config)

# Specify a value for the prefixes that your dataset uses to list the
DATASET_NAME_PREFIX = "dataset-name"
kargs = {"MaxResults": 50}
if DATASET_NAME_PREFIX is not None:
    kargs["DatasetNameBeginsWith"] = DATASET_NAME_PREFIX
has_more_records = True

pp = pprint.PrettyPrinter(depth=4)
print("=====Dataset Summaries=====\n")
while has_more_records:
    list_datasets_response = lookoutequipment.list_datasets(**kargs)
    if "NextToken" in list_datasets_response:
        kargs["NextToken"] = list_datasets_response["NextToken"]
    else:
        has_more_records = False
    # print datasets
    dataset_summaries = list_datasets_response["DatasetSummaries"]
    for dataset_summary in dataset_summaries:
        pp.pprint(dataset_summary)
print("\n=====End of Dataset Summaries=====")

# The following code queries a dataset
dataset_name_to_query = "example-dataset-1" # Change this to dataset name that you want to
 query

describe_dataset_response = lookoutequipment.describe_dataset(
    DatasetName=dataset_name_to_query
)

print("=====Dataset Query Response=====\n")
pp = pprint.PrettyPrinter(depth=5)
```

```
pp.pprint(describe_dataset_response)
print("\n=====End of Response=====\n")

print("=====Schema of Dataset =====\n")
pp.pprint(json.loads(describe_dataset_response["Schema"]))
print("\n=====End of Schema of Dataset=====\n")

# The following code deletes a dataset
dataset_name_to_delete = "example-dataset-1"  # Change this to dataset name that you want
 to delete

delete_dataset_response = lookoutequipment.delete_dataset(
    DatasetName=dataset_name_to_delete
)

print("=====Dataset Delete Response=====\n")
pp = pprint.PrettyPrinter(depth=5)
pp.pprint(delete_dataset_response)
print("\n=====End of Response=====\n")
```

Training a model

You create a model with the dataset that you've ingested. You can train your model on up to 300 sensors in your dataset. Because failures or other serious issues with equipment are rare, Amazon Lookout for Equipment uses your dataset to establish a normal mode of behavior for your asset. If you have data showing that the asset has failed or malfunctioned, you can label those failures in the dataset. Labeling the failures or the events that required equipment maintenance can improve the accuracy of the model.

Before you train a model, you must create a dataset and ingest it. For information about creating a dataset, see Creating a dataset in Amazon Lookout for Equipment (p. 23). For information about ingesting a dataset, see Ingesting a dataset (p. 27).

When you create a model, you can use the following work flow to help improve your model's accuracy:

1. Choose the sensors that your model uses.
2. Use labels for asset failures in the dataset if you have them available.
3. Set the time range for training the model and the time range for evaluating how well it performed. For more information, see Evaluating the output (p. 45) and Improving your results (p. 46).
4. Choose a sampling rate from the original dataset. For a dataset that has sensors taking readings every minute, you can use readings that have been taken every hour to train your model. For more information, see Evaluating the output (p. 45) and Improving your results (p. 46).
5. Provide a condition that indicates that the asset is off. Lookout for Equipment will ignore data satisfying this condition,and will not use that data in training the model.
6. Evaluate how the model performed.
7. If you want to improve the performance of the model, repeat this procedure and choose different sensors, labels, sampling rates, or time ranges.

You might believe that some sensors give more insight into the performance of your asset than others. You can choose which sensors are most useful in training your model.

Amazon Lookout for Equipment is designed to establish a baseline for normal behavior of your assets and detect when your equipment is behaving abnormally. You can improve the model's ability to detect abnormal behavior by using label data that highlights when the equipment wasn't functioning properly.

Within your dataset, you can specify a time range for training your model and a time range for testing your model's performance. You can evaluate your model's performance only if you provide these time ranges.

You might have a lot of data in your dataset. Sampling from that dataset might help you avoid overtraining your model.

The following procedures show you how to create a model.

Training a model (console)

To train a model (console)

1. Sign in to AWS Management Console and open the Amazon Lookout for Equipment console at Amazon Lookout for Equipment console.
2. Choose a dataset that you've ingested.

3. Choose **Create model**.
4. For **Model name**, choose a name for your model.
5. For **Component name**, under **Fields**, choose the sensors that you want to use to train your model.
6. To improve the accuracy of the model, you have the option to do the following.
   - For **S3 location** under **Historical maintenance label event (labels) data - optional**, provide the Amazon S3 location of the label data. For **IAM role**, you must specify an IAM role that provides Amazon Lookout for Equipment access to your data in S3.
   - For **Training and evaluation setting - optional**, provide the following:
     - **Training data time range** - The time range for training the model on your data.
     - **Evaluation data time range** - The time range for testing the model's performance on your data.
   - For **Time series sample rate**, specify the rate that you want to downsample the data from your dataset.
   - For **Off-time detection - optional**, specify the sensor that you will use to detect when the asset is turned off. Then select the threshold that will indicate the off state for that sensor.

Training a model (AWS SDK for Python (Boto3))

The following example code uses the AWS SDK for Python (Boto3) to train a model.

```
import boto3
import json
import pprint
import time
from datetime import datetime
from botocore.config import Config
config = Config(region_name = 'Region')
lookoutequipment = boto3.client(service_name="lookoutequipment", config=config)
MODEL_NAME = 'model-name'
# You can choose a sampling rate for your data. The valid values are "PT1S", "PT5S",
 "PT10S", "PT15S", "PT30S", "PT1M", "PT5M", "PT10M", "PT15M", "PT30M", "PT1H". S - second,
 M - minute, H - hour
TARGET_SAMPLING_RATE = 'sampling-rate'
# If you have label data, specify the following variables
LABEL_DATA_SOURCE_BUCKET = 'label-data-source-bucket'
LABEL_DATA_SOURCE_PREFIX = 'label-data-source-prefix/' # This must end with "/" if you
 provide a prefix
# The following are example training and evaluation start times. datetime(2018, 8, 13, 0,
 0, 0) generates 2018-08-13 00:00:00
TRAINING_DATA_START_TIME = datetime(2016, 11, 1, 0, 0, 0)
TRAINING_DATA_END_TIME = datetime(2017, 12, 31, 0, 0, 0)
EVALUATION_DATA_START_TIME = datetime(2018, 1, 1, 0, 0, 0)
EVALUATION_DATA_END_TIME = datetime(2018, 8, 13, 0, 0, 0)
# To configure off-time detection, use the format
# OFF_CONDITION = '{component}\\{sensor} < {target}'
# In the following example, Asset X will be considered to be in the off state if the latest
 value
# received by Sensor 1 is less than 10.
OFF_CONDITION = 'AssetX\\Sensor1 < 10.0'
########################################################
# construct request for create_model
########################################################
model_name = MODEL_NAME
DATA_SCHEMA_FOR_MODEL = None # You can use a schema similar to dataset here. The sensors
 used here should be subset of what is present in dataset
create_model_request = {
```

```
    'ModelName': model_name,
    'DatasetName': DATASET_NAME,
}
if DATA_SCHEMA_FOR_MODEL is not None:
    data_schema_for_model = {
        'InlineDataSchema': DATA_SCHEMA_FOR_MODEL,
    }
    create_model_request['DatasetSchema'] = data_schema_for_model
if TARGET_SAMPLING_RATE is not None:
    data_preprocessing_config = {
        'TargetSamplingRate': TARGET_SAMPLING_RATE
    }
    create_model_request['DataPreProcessingConfiguration'] = data_preprocessing_config
if LABEL_DATA_SOURCE_BUCKET is not None:
    labels_input_config = dict()
    labels_input_config['S3InputConfiguration'] = dict(
        [
            ('Bucket', LABEL_DATA_SOURCE_BUCKET),
            ('Prefix', LABEL_DATA_SOURCE_PREFIX)
        ]
    )
    create_model_request['LabelsInputConfiguration'] = labels_input_config
    # We need to set role_arn to access label data
    create_model_request['RoleArn'] = ROLE_ARN
if TRAINING_DATA_START_TIME is not None or TRAINING_DATA_END_TIME is not None:
    create_model_request['TrainingDataStartTime'] = TRAINING_DATA_START_TIME
    create_model_request['TrainingDataEndTime'] = TRAINING_DATA_END_TIME
if EVALUATION_DATA_START_TIME is not None or EVALUATION_DATA_END_TIME is not None:
    create_model_request['EvaluationDataStartTime'] = EVALUATION_DATA_START_TIME
    create_model_request['EvaluationDataEndTime'] = EVALUATION_DATA_END_TIME
if OFF_CONDITION is not None:
    create_model_request['OffCondition'] = OFF_CONDITION
############################################################
# Create_model
############################################################
create_model_response = lookoutequipment.create_model(**create_model_request)
############################################################
# Wait until complete
############################################################
model_status = create_model_response['Status']
print("=====Polling Model Status=====\n")
print("Model Status: " + model_status)
while model_status == 'IN_PROGRESS':
    time.sleep(30)
    describe_model_response = lookoutequipment.describe_model(ModelName=model_name)
    model_status = describe_model_response['Status']
    print("Model Status: " + model_status)
print("\n=====End of Polling Model Status=====")
```

Viewing your models and evaluating their performance

You can view the ML models you've trained on the datasets containing the data from your equipment. If you've used part of your dataset for training and the other part for evaluation, you can see and evaluate the model's performance. You can also see which sensors were used to create a model. If you need better performance, you can use different sensors for training your next model.

Amazon Lookout for Equipment provides an overview of the model's performance and detailed information about abnormal equipment behavior events. An abnormal equipment behavior event

is a situation where the model detected an anomaly in the sensor data that could lead to your asset malfunctioning or failing. You can see how well the model performed in detecting those events.

If you've provided Amazon Lookout for Equipment with label data for your dataset, you can see how the model's predictions compare to the label data. It shows the average forewarning time across all true positives. Forewarning time is the average length of time between when the model first finds evidence that something might be going wrong and when it actually detects the equipment abnormality.

For example, you can have a circumstance where Amazon Lookout for Equipment detects six of the seven abnormal behavior events in your labeled evaluation data. In six out of the seven events, on average, it might have provided an indication that something was off 32 hours before it detected an abnormality. For this situation, we would say that Lookout for Equipment averaged 32 hours of forewarning.

Amazon Lookout for Equipment also reports the results where it incorrectly identified an abnormal behavior event in the label data. The label data that you provide when you create a dataset has a time range for abnormal equipment events. You specify the duration of the abnormal events in the label data. In the evaluation data, the model used by Lookout for Equipment could incorrectly identify abnormal events outside of the equipment range. You can see how often the model identifies these events when you evaluate the model's performance.

You can use the following procedure and example code to view the models you've created. They also show you how to get information about a model, such as how well it performed. For details on how to use this information to find the root cause of an anomaly, see Finding the root cause of an anomaly (p. 37).

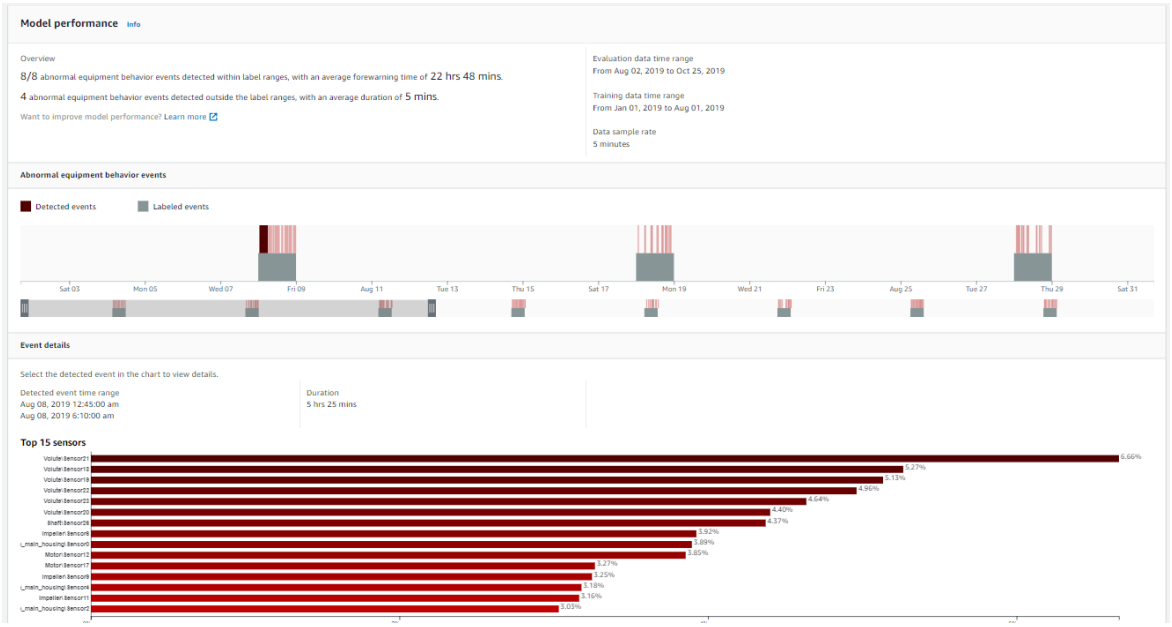
Viewing a model (console)

To view a model (console)

You can use this procedure to view model metrics in the console. To evaluate how the model performed, you must provide label data. If you provide label data, you can see when the model detected abnormal equipment behavior events.

1. Sign in to AWS Management Console and open the Amazon Lookout for Equipment console at Amazon Lookout for Equipment console.
2. Choose a dataset.
3. Choose a model. You can see whether the model is ready to monitor the equipment.
4. Navigate to **Training and evaluation**.

In the following image, you can see metrics related to the performance. You can see how many times the model identified abnormal equipment behavior events incorrectly. You can also see which sensors played the largest role in the model identifying the abnormal equipment behavior events. The console displays the top 15 sensors that contributed to the model identifying an abnormal equipment behavior event.



To view a model (AWS SDK for Python (Boto3))

Use the following example AWS SDK for Python (Boto3) code to list the models that you've trained, to query a model's metadata, and to delete a model that you no longer want to use. If you've used label data when you created a dataset, you can also use this code to see how well the model performed. To run this code successfully, you must use the SDK for Python code in Training a model (p. 32) before you run the code shown here.

```
import boto3
import json
import pprint
import time
from datetime import datetime
from botocore.config import Config
config = Config(region_name = 'Region')
lookoutequipment = boto3.client(service_name="lookoutequipment", config=config)
# List models
MODEL_NAME_PREFIX = None
DATASET_NAME_FOR_LIST_MODELS = None
list_models_request = {}
if MODEL_NAME_PREFIX is not None:
    list_models_request["ModelNameBeginsWith"] = MODEL_NAME_PREFIX
if DATASET_NAME_FOR_LIST_MODELS is not None:
    list_models_request["DatasetNameBeginsWith"] = DATASET_NAME_PREFIX

pp = pprint.PrettyPrinter(depth=5)
print("=====Model Summaries=====\n")
has_more_records = True
while has_more_records:
    list_models_response = lookoutequipment.list_models(**list_models_request)
    if "NextToken" in list_models_response:
        list_models_request["NextToken"] = list_models_response["NextToken"]
    else:
        has_more_records = False
    model_summaries = list_models_response["ModelSummaries"]
    for model_summary in model_summaries:
        pp.pprint(model_summary)
```

```
print("\n=====End of Model Summaries=====")
# Query the metadata for a model
MODEL_NAME_TO_QUERY = MODEL_NAME
describe_model_response = lookoutequipment.describe_model(ModelName=MODEL_NAME_TO_QUERY)
print(f'Model Status: {describe_model_response["Status"]}')
if "FailedReason" in describe_model_response:
    print(f'Model FailedReason: {describe_model_response["FailedReason"]}')
print("\n\n=====DescribeModel Response=====\n")
pp = pprint.PrettyPrinter(depth=5)
pp.pprint(describe_model_response)
print("\n=====End of DescribeModel Response=====")
# Get evaluation metrics for a model
MODEL_NAME_TO_DOWNLOAD_EVALUATION_METRICS = MODEL_NAME
describe_model_response =
 lookoutequipment.describe_model(ModelName=MODEL_NAME_TO_DOWNLOAD_EVALUATION_METRICS)
if 'ModelMetrics' in describe_model_response:
    model_metrics = json.loads(describe_model_response['ModelMetrics'])
    print("===== Model Metrics =====\n")
    pp = pprint.PrettyPrinter(depth=5)
    pp.pprint(model_metrics)
    print("\n=====End of Model Metrics=====\n")
else:
    print('Model metrics is only available if evaluation data is provided during
 training.')
# Delete a model
MODEL_NAME_TO_DELETE = MODEL_NAME

model_name_to_delete = MODEL_NAME_TO_DELETE
delete_model_response = lookoutequipment.delete_model(
    ModelName=model_name_to_delete
)
print("=====DeleteModel Response=====\n")
pp = pprint.PrettyPrinter(depth=5)
pp.pprint(delete_model_response)
print("\n=====End of DeleteModel Response=====\n")
```

Finding the root cause of an anomaly

You can use the diagnostics data from the preceding section to evaluate how well your model performed.

The following is an example response from running the DescribeModel (p. 107) operation in the AWS SDK for Python (Boto3). The model metrics containing the diagnostics data for 7 events (predicted_ranges) are shown in the response. The `diagnostics` fields list all of the sensors that provided data to indicate abnormal behavior. The `name` field indicates the name of the sensor. The `value` field is a number that indicates how much each sensor was weighted in identifying the abnormal event.

```
ModelMetrics: {
    labeled_ranges: [],
    labeled_event_metrics: {
        num_labeled: 0,
        num_identified: 0,
        total_warning_time_in_seconds: 0
    },
    predicted_ranges: [
        {start: 2019-10-17T00:24:00.000000, end: 2019-10-17T02:56:00.000000, diagnostics:
 [{name: component_main\\sensor_0, value: 0.16154573223190147}, {name: component_main
\\sensor_1, value: 0.1697679951259812}, {name: component_main\\sensor_2, value:
 0.16394157603618906}, {name: component_main\\sensor_3, value: 0.17357875229275566}, {name:
 component_main\\sensor_4, value: 0.17352280206054818}, {name: component_main\\sensor_5,
 value: 0.15764314225262444}]},
```

```
        {start: 2019-10-17T03:02:00.000000, end: 2019-10-17T03:02:00.000000, diagnostics:
 [{name: component_main\\sensor_0, value: 0.16849938641427187}, {name: component_main
\\sensor_1, value: 0.16663977201023458}, {name: component_main\\sensor_2, value:
 0.1712862229918278}, {name: component_main\\sensor_3, value: 0.16310608759722545}, {name:
 component_main\\sensor_4, value: 0.16516840127817176}, {name: component_main\\sensor_5,
 value: 0.16530012970826857}]},
        {start: 2019-10-17T05:42:00.000000, end: 2019-10-17T05:42:00.000000, diagnostics:
 [{name: component_main\\sensor_0, value: 0.16091496490383986}, {name: component_main
\\sensor_1, value: 0.15177612221858203}, {name: component_main\\sensor_2, value:
 0.1637097283826505}, {name: component_main\\sensor_3, value: 0.16193058093785526}, {name:
 component_main\\sensor_4, value: 0.17414216426064247}, {name: component_main\\sensor_5,
 value: 0.18752643929642995}]},
        {start: 2019-10-17T06:26:00.000000, end: 2019-10-17T06:28:00.000000, diagnostics:
 [{name: component_main\\sensor_0, value: 0.17469036505563895}, {name: component_main
\\sensor_1, value: 0.18892270917688264}, {name: component_main\\sensor_2, value:
 0.16967105931170198}, {name: component_main\\sensor_3, value: 0.17254470649932127}, {name:
 component_main\\sensor_4, value: 0.15006595417461285}, {name: component_main\\sensor_5,
 value: 0.14410520578184224}]},
        {start: 2019-10-17T06:31:00.000000, end: 2019-10-17T07:34:00.000000, diagnostics:
 [{name: component_main\\sensor_0, value: 0.15928077475120272}, {name: component_main
\\sensor_1, value: 0.18297842892351804}, {name: component_main\\sensor_2, value:
 0.1543784051686879}, {name: component_main\\sensor_3, value: 0.16188977526850865}, {name:
 component_main\\sensor_4, value: 0.18557118349484641}, {name: component_main\\sensor_5,
 value: 0.1559014323932363}]},
        {start: 2019-10-17T10:17:00.000000, end: 2019-10-17T10:18:00.000000, diagnostics:
 [{name: component_main\\sensor_0, value: 0.1409553823637457}, {name: component_main
\\sensor_1, value: 0.17266259515970778}, {name: component_main\\sensor_2, value:
 0.13590734192840642}, {name: component_main\\sensor_3, value: 0.2070527421593214}, {name:
 component_main\\sensor_4, value: 0.15628628577213594}, {name: component_main\\sensor_5,
 value: 0.1871356526166826}]},
        {start: 2019-10-17T10:36:00.000000, end: 2019-10-17T11:38:00.000000, diagnostics:
 [{name: component_main\\sensor_0, value: 0.165355932999884}, {name: component_main
\\sensor_1, value: 0.19432363801205416}, {name: component_main\\sensor_2, value:
 0.15276513885315468}, {name: component_main\\sensor_3, value: 0.13492793133083875}, {name:
 component_main\\sensor_4, value: 0.1868355510176609}, {name: component_main\\sensor_5,
 value: 0.16579180778640745}]},
        ],
    unknown_event_metrics: {num_identified: 40, total_duration_in_seconds: 61020.0}
}
```

Monitoring your equipment in real time

After you create a model, you can use it to monitor your asset in real-time. To use your model to monitor your asset, you do the following.

1. Create an Amazon S3 bucket to store the data that your asset outputs.
2. Create a folder path location to store the output from the asset.
3. Create an Amazon S3 bucket for the model to output its analysis of the data from your asset.
4. Create a folder path location for the model to store the output of its analysis.
5. Create a data pipeline from your asset to the S3 location that stores the output data.
6. In Amazon Lookout for Equipment, specify the frequency that data is uploaded.
7. Schedule the time period that your model performs inference on the data coming from your pipeline.

To create an inference schedule on the data that your model outputs to Amazon S3, use one of the following procedures.

Schedule inference (console)

To schedule an inference (console)

To schedule inference, you specify the model, the schedule, the S3 location of where the model is reading the data, and where it outputs the results of the inference.

1. Sign in to AWS Management Console and open the Amazon Lookout for Equipment console at Amazon Lookout for Equipment console.
2. Choose **Models**. Then choose the model that monitors your asset.
3. Choose **Schedule inference**.
4. For **Inference schedule name**, specify the name for the inference schedule.
5. For **Model**, choose the model that is monitoring the data coming from your asset.
6. For **S3 location** under **Input data**, specify the Amazon S3 location of the input data coming from the asset.
7. For **Data upload frequency**, specify how often your asset sends the data to the S3 bucket.
8. For **S3 location** under **Output data**, specify the S3 location to store the output of the inference results.
9. For **IAM role** under **Access Permissions**, specify the IAM role that provides Amazon Lookout for Equipment with access to your data in Amazon S3.
10. Choose **Schedule inference**.

Schedule inference (AWS SDK for Python (Boto3))

The following example code uses the AWS SDK for Python (Boto3) to schedule an inference for your asset.

```
import boto3
```

```
import json
import pprint
import time
from datetime import datetime
from botocore.config import Config
config = Config(region_name = 'Region')
lookoutequipment = boto3.client(service_name="lookoutequipment", config=config)
# Specify a name for the inference scheduler
INFERENCE_SCHEDULER_NAME = 'inference-scheduler-name'
MODEL_NAME_FOR_CREATING_INFERENCE_SCHEDULER = 'model-name'
# You must specify values for the following variables to successfully schedule an
 inference.
# DATA_UPLOAD_FREQUENCY - The frequency that the data from the asset uploads to the Amazon
 S3 data containing the inference data. The valid values are PT5M, PT10M, PT30M, and PT1H
# INFERENCE_DATA_SOURCE_BUCKET - The S3 bucket that stores the inference data coming from
 your asset.
# INFERENCE_DATA_SOURCE_PREFIX - The S3 prefix that helps you access the inference data
 coming from your asset.
# INFERENCE_DATA_OUTPUT_BUCKET - The S3 bucket that stores the results of the inference.
# INFERENCE_DATA_OUTPUT_PREFIX - The S3 prefix that helps you access the results of the
 inference.
# ROLE_ARN_FOR_INFERENCE - The IAM role that gives Amazon Lookout for Equipment read
 permissions for Amazon S3.
# You can specify values for the following optional variables.
# DATA_DELAY_OFFSET_IN_MINUTES - The number of minutes to account for a delay in uploading
 the data to Amazon S3 from your data pipeline.
# INPUT_TIMEZONE_OFFSET - The default timezone for running inference is in UTC. You can
 offset the default timezone in increments of 30 minutes. This offset only applies to the
 file name. If you choose to use the offset, you must have the timestamps for the sensor in
 UTC as well. The valid values include +00:00, +00:30, -01:00, ... +11:30, +12:00, -00:00,
 -00:30, -01:00, ... -11:30, -12:00.
# TIMESTAMP_FORMAT - You can specify how the model outputs the timestamp in the results of
 the inference. The valid values are `EPOCH`, `yyyy-MM-dd-HH-mm-ss` or `yyyyMMddHHmmss`.
# COMPONENT_TIMESTAMP_DELIMITER - Specifies the character used to separate entries in the
 input data. Default delimiter is - (hyphen). The valid values are `-`, `_` or ` `.
DATA_DELAY_OFFSET_IN_MINUTES = None
INPUT_TIMEZONE_OFFSET = None
COMPONENT_TIMESTAMP_DELIMITER = None
TIMESTAMP_FORMAT = None
# Create an inference scheduler.
scheduler_name = INFERENCE_SCHEDULER_NAME
model_name = MODEL_NAME_FOR_CREATING_INFERENCE_SCHEDULER
INFERENCE_DATA_SOURCE_BUCKET = 'data-source-bucket'
INFERENCE_DATA_SOURCE_PREFIX = 'data-source-prefix'
INFERENCE_DATA_OUTPUT_BUCKET = 'data-output-bucket'
INFERENCE_DATA_OUTPUT_PREFIX = 'data-output-prefix'
ROLE_ARN_FOR_INFERENCE = ROLE_ARN
DATA_UPLOAD_FREQUENCY = 'data-upload-frequency'
create_inference_scheduler_request = {
    'ModelName': model_name,
    'InferenceSchedulerName': scheduler_name,
    'DataUploadFrequency': DATA_UPLOAD_FREQUENCY,
    'RoleArn': ROLE_ARN_FOR_INFERENCE,
}
if DATA_DELAY_OFFSET_IN_MINUTES is not None:
    create_inference_scheduler_request['DataDelayOffsetInMinutes'] =
 DATA_DELAY_OFFSET_IN_MINUTES
# Set up data input configuration.
inference_input_config = dict()
inference_input_config['S3InputConfiguration'] = dict(
    [
        ('Bucket', INFERENCE_DATA_SOURCE_BUCKET),
        ('Prefix', INFERENCE_DATA_SOURCE_PREFIX)
    ]
)
if INPUT_TIMEZONE_OFFSET is not None:
```

```
    inference_input_config['InputTimeZoneOffset'] = INPUT_TIMEZONE_OFFSET
if COMPONENT_TIMESTAMP_DELIMITER is not None or TIMESTAMP_FORMAT is not None:
    inference_input_name_configuration = dict()
    if COMPONENT_TIMESTAMP_DELIMITER is not None:
        inference_input_name_configuration['ComponentTimestampDelimiter'] =
 COMPONENT_TIMESTAMP_DELIMITER
    if TIMESTAMP_FORMAT is not None:
        inference_input_name_configuration['TimestampFormat'] = TIMESTAMP_FORMAT
    inference_input_config['InferenceInputNameConfiguration'] =
 inference_input_name_configuration
create_inference_scheduler_request['DataInputConfiguration'] = inference_input_config
#  Set up output configuration.
inference_output_configuration = dict()
inference_output_configuration['S3OutputConfiguration'] = dict(
    [
        ('Bucket', INFERENCE_DATA_OUTPUT_BUCKET),
        ('Prefix', INFERENCE_DATA_OUTPUT_PREFIX)
    ]
)
create_inference_scheduler_request['DataOutputConfiguration'] =
 inference_output_configuration
########################################################
# Invoke create_inference_scheduler
########################################################
create_scheduler_response =
 lookoutequipment.create_inference_scheduler(**create_inference_scheduler_request)
print("\n\n=====CreateInferenceScheduler Response=====\n")
pp = pprint.PrettyPrinter(depth=5)
pp.pprint(create_scheduler_response)
print("\n=====End of CreateInferenceScheduler Response=====")
########################################################
# Wait until RUNNING
########################################################
scheduler_status = create_scheduler_response['Status']
print("=====Polling Inference Scheduler Status=====\n")
print("Model Status: " + scheduler_status)
while scheduler_status == 'PENDING':
    time.sleep(5)
    describe_scheduler_response =
 lookoutequipment.describe_inference_scheduler(InferenceSchedulerName=INFERENCE_SCHEDULER_NAME)
    scheduler_status = describe_scheduler_response['Status']
    print("Scheduler Status: " + scheduler_status)
print("\n=====End of Polling Inference Scheduler Status=====")
```

Seeing how your equipment operates

After you've scheduled inference, you can see how your equipment is operating. The JSON file containing the inference results is stored in the Amazon Simple Storage Service (Amazon S3) bucket that you've specified. You can only access the results by downloading the JSON file in the Amazon S3 location that you've specified to store the results of the inference.

For the sensor data that your asset sends to Amazon S3, Amazon Lookout for Equipment marks the group of readings as either normal or abnormal. For each group of abnormal readings, you can see the sensors that Lookout for Equipment used to indicate that the equipment is behaving abnormally.

The following code shows an example JSON output.

```
{"timestamp": "2021-03-11T22:25:00.000000", "prediction": 1, "diagnostics": [{"name":
 "component_5feceb66\\sensor0", "value": 0.02346}, {"name": "component_5feceb66\\sensor1",
```

```
 "value": 0.10011}, {"name": "component_5feceb66\\sensor2", "value": 0.11162}, {"name":
 "component_5feceb66\\sensor3", "value": 0.14419}, {"name": "component_5feceb66\\sensor4",
 "value": 0.12219}, {"name": "component_5feceb66\\sensor5", "value": 0.14936}, {"name":
 "component_5feceb66\\sensor6", "value": 0.17829}, {"name": "component_5feceb66\\sensor7",
 "value": 0.00194}, {"name": "component_5feceb66\\sensor8", "value": 0.05446}, {"name":
 "component_5feceb66\\sensor9", "value": 0.11437}]}
{"timestamp": "2021-03-11T22:26:00.000000", "prediction": 0}
{"timestamp": "2021-03-11T22:27:00.000000", "prediction": 0}
{"timestamp": "2021-03-11T22:28:00.000000", "prediction": 0}
{"timestamp": "2021-03-11T22:29:00.000000", "prediction": 1, "diagnostics": [{"name":
 "component_5feceb66\\sensor0", "value": 0.04533}, {"name": "component_5feceb66\\sensor1",
 "value": 0.14063}, {"name": "component_5feceb66\\sensor2", "value": 0.08327}, {"name":
 "component_5feceb66\\sensor3", "value": 0.07303}, {"name": "component_5feceb66\\sensor4",
 "value": 0.18598}, {"name": "component_5feceb66\\sensor5", "value": 0.10839}, {"name":
 "component_5feceb66\\sensor6", "value": 0.08721}, {"name": "component_5feceb66\\sensor7",
 "value": 0.06792}, {"name": "component_5feceb66\\sensor8", "value": 0.1309}, {"name":
 "component_5feceb66\\sensor9", "value": 0.07735}]}
```

For the `prediction` field, a `value` of `1` indicates abnormal equipment behavior. A `value` of `0` indicates normal equipment behavior.

When the `value` is `1`, Amazon Lookout for Equipment returns an object that contains a diagnostic list. The `diagnostics` list has the name of the sensors and the weights of the sensors' contributions in indicating abnormal equipment behavior. For each sensor, the `name` field indicates the name of the sensor. The `value` field indicates the percentage of the sensor's contribution to the prediction value. By seeing the percentage of each sensor's contribution to the prediction value, you can see how the data from each sensor was weighted.

Best practices with Lookout for Equipment

Training a machine learning (ML) model can involve inputs from up to 300 sensors, and you can have up to 3000 sensors represented in a single dataset. We highly recommend that you consult a subject matter expert (SME) when setting up Lookout for Equipment to monitor your equipment. This will help you get the most out of Lookout for Equipment.

We also recommend that you understand and follow the best practices described in this topic. There are three key pillars essential to setting up Lookout for Equipment for the best possible results:

- Selecting the right application
- Selecting the right data inputs
- Working with SMEs to select the inputs and evaluate the results

Choosing the right application

Choosing the right application of Lookout for Equipment involves finding the right combination of business value, equipment operations, and available data. You determine this by working directly with a subject matter experts (SME) on your equipment. Your team should consider the following:

- **The high cost of downtime** – Equipment that can either be costly to fix or that is critical to a process is a prime candidate for monitoring.
- **Consistency in operations** – Lookout for Equipment works best on equipment that is stationary and primarily does a continuous, stable task. A heavy duty pump that is permanently installed in a location is a good example.
- **Relevant data** – Having data that is relevant to the critical aspects of the equipment is essential. Your equipment should have sensors that monitor these critical aspects, so that they can provide data that is relevant to how your equipment could fail. Having this data can make the difference between inference results that can effectively catch potential failures and abnormal behavior, and results that don't.
- **Significant historical data** – Ideally, the data you use to train the machine learning (ML) model should represent all of the equipment's operating modes. For instance, when creating a model for a pump with variable speeds, the dataset should contain measurements that include an adequate amount of historical data for all of the pump speeds. For effective analysis, Lookout for Equipment should have at least six months of historical data, although a longer history is preferred. For equipment affected by seasonality, at least one year of data is highly recommended.
- **List of historical failures (optional)** – Lookout for Equipment uses data on historical failures to enhance the model's knowledge of normal equipment conditions. It looks for abnormal behavior that occurred ahead of historical failures. With more examples of historical failures, Lookout for Equipment can better develop its knowledge of healthy conditions and the unhealthy conditions that occur prior to failures. The definition of a failure can be subjective, but we have found that looking for issues that cause unplanned downtime is a good method to identify failure. For best results, give Lookout for Equipment label data for every known time period where the equipment had issues or abnormal behavior.

> **Note**
> Lookout for Equipment is ultimately dependent on *your* data. We cannot guarantee that there are patterns in your data that will enable Lookout for Equipment to detect failures.

Determining the right set of inputs might require multiple iterations through the Lookout for Equipment model training and monitoring process. For the greatest chance of success, we highly recommend working with a subject matter expert to identify the right application and data.

Choosing and formatting the right data

Your dataset should contain time-series data that's generated from an industrial asset such as a pump, compressor, motor, and so on. Each asset should be generating data from one or more sensors. The data that Lookout for Equipment uses for training should be representative of the condition and operation of the asset. Making sure that you have the right data is crucial. We recommend that you work with a SME. A SME can help you make sure that the data is relevant to the aspect of the asset that you're trying to analyze. We recommend that you remove unnecessary sensor data. With data from too few sensors, you might miss critical information. With data from too many sensors, your model might overfit the data and it might miss out on key patterns.

Important
Choosing the right input data is crucial to the success of using Lookout for Equipment. It might take multiple iterations of trial and error to find the right inputs. We cannot guarantee results. Success is highly dependent on the relevancy of your data to equipment issues.

Use these guidelines to choose the right data:

- **Use only numerical data** – Remove nonnumerical data. Lookout for Equipment can't use non-numerical data for analysis.
- **Use only analog data** – Use only analog data (that is, many values that vary over time). Using digital values (also known as categorical values, or values that can be only one of a limited number of options), such as valve positions or set points, can lead to inconsistent or misleading results.
- **Remove continuously increasing data** – Remove data that is just an ever-increasing number, such as operating hours or mileage.
- **Use data for the relevant component or subcomponent** – You can use Lookout for Equipment to monitor an entire asset (such as a pump) or just a subcomponent (such as a pump motor). Determine where your downtime issues occur and choose the component or subcomponent that has the greater effect on that.

When formatting a predictive maintenance problem, consider these guidelines:

- **Data size** – Although Lookout for Equipment can ingest more than 50 GB of data, it can use only 7 GB with a model. Factors such as the number of sensors used, how far back in history the dataset goes, and the sample rate of the sensors can all determine how many measurements this amount of data can include. This amount of data also includes the missing data imputed by Lookout for Equipment.
- **Missing data** – Lookout for Equipment automatically fills in missing data (known as imputing). It does this by forward filling previous sensor readings. However, if too much original data is missing, it might affect your results.
- **Sample rate** – *Sample rate* is the interval at which the sensor readings are recorded. Use the highest frequency sample rate possible without exceeding the data size limit. The sample rate and data size might also increase your ML model training time. Lookout for Equipment handles any timestamp misalignment.
- **Number of sensors** – Lookout for Equipment can train a model with data from up to 300 sensors. However, having the right data is more important than the quantity of data. More is not necessarily better.
- **Vibration** – Although vibration data is usually important for identifying potential failure, Lookout for Equipment does not work with raw high-frequency data. When using high-frequency vibration data, first generate the key values from the vibration data, such as RMS and FFT.

Filtering for normal data

Make sure that you use only data from normal (standard) operations. To do this, identify a key operating metric that indicates that the equipment is operating in a standard fashion. For example, when operating a compressor in a refinery, the key metric is usually production flow rate. In this case, you would need to filter out times when the production flow rate is below normal due to reduced production or any reason other than abnormal behavior. Other examples of key metrics might be RPM, fuel efficiency, "run" state, availability, and so on. Lookout for Equipment assumes that the data is normal. Making sure that the data fits this assumption is very important.

Using failure labels

To provide insight into past events, Lookout for Equipment uses labels that call out these events for the ML model. Providing this data is optional, but if it's available, it can help train your model more accurately and efficiently.

Lookout for Equipment takes this information in as two timestamps in a CSV file stored in an Amazon Simple Storage Service (Amazon S3) bucket. The first timestamp indicates when abnormal behavior is expected to have started. The second timestamp is when the failure or abnormal behavior was first noticed. Alternatively, the second timestamp can indicate a maintenance event. Lookout for Equipment uses this window as the basis for looking for signs of an upcoming event so it can better understand what those events look like on this machine. Ideally, the timestamps correspond to data during a maintenance event. We recommend that you filter out data from any restart procedure.

The following is an example of such a CSV file.

Row	Timestamp 1	Timestamp 2
1	1/3/2020 0:00	1/5/2020 0:00
2	2/2/2020 0:05	2/7/2020 0:05
3	4/11/2020 0:10	4/21/2020 0:10

Row 1 represents a maintenance event on January 3rd with a 2-day window for Lookout for Equipment to look for abnormal behavior.

Row 2 represents a maintenance event on February 7th with a 5-day window for Lookout for Equipment to look for abnormal behavior.

Row 3 represents a maintenance event on April 21st with a 10-day window for Lookout for Equipment to look for abnormal behavior.

Lookout for Equipment uses all of these time windows to look for an optimal model that finds abnormal behavior within these windows. Note that not all events are detectable and most are highly dependent on the data provided.

Evaluating the output

After a model is trained, Lookout for Equipment evaluates its performance on a subset of the dataset that you've specified for evaluation purposes. It displays results that provide an overview of the

performance and detailed information about the abnormal equipment behavior events and how well the model performed when detecting those.

Using the data and failure labels that you provided for training and evaluating the model, Lookout for Equipment reports how many times the model's predictions were true positives (how often the model found the equipment anomaly that was noted within the ranges shown in the labels). Within a labeled time range, the forewarning time represents the duration between the earliest time when the model found an anomaly and the end of the labeled time range.

For example, if Lookout for Equipment reports that "6/7 abnormal equipment behavior events were detected within label ranges with an average forewarning time of 32 hrs," in 6 out of the 7 labeled events, the model detected that event and averaged 32 hours of forewarning. In one case, it did not detect the event.

Lookout for Equipment also reports the abnormal behavior events that were not related to a failure, along with the duration of these abnormal behavior events. For example, if it reports that "5 abnormal equipment behavior events were detected outside the label range with an average duration of 4 hrs," the model thought an event was occurring in 5 cases. An abnormal behavior event such as this one might be attributed to someone erroneously operating the equipment for a period of time or a normal operating mode that you haven't seen previously.

Lookout for Equipment also displays this information graphically on a chart that shows the days and events and in a table.

Lookout for Equipment provides detailed information about the anomalous events that it detects. It displays a list of sensors that provided the data to indicate an anomalous event. This might help you determine which part of your asset is behaving abnormally.

Improving your results

To improve the results, consider the following:

- Did unrecorded maintenance events, system inefficiencies, or a new normal operating mode happen during the time of flagged anomalies in the test set? If so, the results indicate those situations. Change your train-evaluation splits so that each normal mode is captured during model training.
- Are the sensor inputs relevant to the failure labels? In other words, is it possible that the labels are related to one component of the equipment but the sensors are monitoring a different component? If so, consider building a new model where the sensor inputs and labels are relevant to each other and drop any irrelevant sensors. Alternatively, drop the labels you're using and train the model only on the sensor data.
- Is the label time zone the same as the sensor data time zone? If not, consider adjusting the time zone of your label data to align with sensor data time zone.
- Is the failure label range inadequate? In other words, could there be anomalous behavior outside of the label range? This can happen for a variety of reasons, such as when the anomalous behavior was observed much earlier than the actual repair work. If so, consider adjusting the range accordingly.
- Are there data integrity issues with your sensor data? For example, do some of the sensors become nonfunctional during the training or evaluation data? In that case, consider dropping those sensors when you run the model. Alternatively, use a training-evaluation split that filters out the non-functional part of the sensor data.
- Does the sensor data include uninteresting normal-operating modes, such as off-periods or ramp-up or ramp-down periods? Consider filtering those out of the sensor data.
- We recommend that you avoid using data that contains monotonically increasing values, such as operating hours or mileage.

Consulting subject matter experts

Lookout for Equipment identifies patterns in the dataset that help to detect critical issues, but it's the responsibility of a technician or subject matter expert (SME) to diagnose the problem and take corrective action, if needed. To ensure that you are getting the right output, we highly recommend that you work with a SME. The SME should help you make sure that you are using the right input data and that your output results are actionable and relevant.

Quotas for using Lookout for Equipment

Supported Regions

For a list of AWS Regions where Lookout for Equipment is available, see AWS Regions and Endpoints in the *AWS General Reference*.

Quotas

Service quotas, also referred to as limits, are the maximum number of service resources for your AWS account. For more information, see AWS Service Quotas in the *AWS General Reference*.

Description	Quota
Data ingestion	
Maximum number of components per dataset	3,000
Maximum number of datasets per account	15
Maximum number of pending data ingestion jobs per account	5
Maximum number of models per account	15
Maximum number of columns across components per dataset (excluding timestamp)	3,000
Maximum number of files per component (per dataset)	1,000
Maximum length of component name	200 characters
Maximum size per dataset	50 GB
Maximum size per file	5 GB
Maximum number of pending models per account	5
Maximum number of inference schedulers per model	1
Training and evaluation	
Maximum number of rows in training data (after resampling)	1.5 million
Maximum number of rows in evaluation data (after resampling)	1.5 million
Maximum number of components in training data	300
Maximum number of columns across components in training data (excluding timestamp)	300

Description	Quota
Minimum timespan of training data	180 days
Inference	
Maximum size of raw data in inference input data (5-min scheduling frequency)	5 MB
Maximum size of raw data in inference input data (10-min scheduling frequency)	10 MB
Maximum size of raw data in inference input data (15-min scheduling frequency)	15 MB
Maximum size of raw data in inference input data (30-min scheduling frequency)	30 MB
Maximum size of raw data in inference input data (1-hour scheduling frequency)	60 MB
Maximum number of rows in inference input data, after resampling (5-min scheduling frequency)	300
Maximum number of rows in inference input data, after resampling (10-min scheduling frequency)	600
Maximum number of rows in inference input data, after resampling (15-min scheduling frequency)	900
Maximum number of rows in inference input data, after resampling (30-min scheduling frequency)	1,800
Maximum number of rows in inference input data, after resampling (1-hour scheduling frequency)	3,600
Maximum number of files per component (per inference execution)	60

Security in Amazon Lookout for Equipment

Cloud security at AWS is the highest priority. As an AWS customer, you benefit from data centers and network architectures that are built to meet the requirements of the most security-sensitive organizations.

Security is a shared responsibility between AWS and you. The shared responsibility model describes this as security *of* the cloud and security *in* the cloud:

- **Security of the cloud** – AWS is responsible for protecting the infrastructure that runs AWS services in the AWS Cloud. AWS also provides you with services that you can use securely. Third-party auditors regularly test and verify the effectiveness of our security as part of the AWS Compliance Programs. To learn about the compliance programs that apply to Amazon Lookout for Equipment, see AWS Services in Scope by Compliance Program.
- **Security in the cloud** – Your responsibility is determined by the AWS service that you use. You are also responsible for other factors including the sensitivity of your data, your company's requirements, and applicable laws and regulations.

This documentation helps you understand how to apply the shared responsibility model when using Lookout for Equipment. The following topics show you how to configure Lookout for Equipment to meet your security and compliance objectives. You also learn how to use other AWS services that help you to monitor and secure your Lookout for Equipment resources.

Topics

Data protection in Amazon Lookout for Equipment

Amazon Lookout for Equipment conforms to the AWS shared responsibility model, which includes regulations and guidelines for data protection. AWS is responsible for protecting the global infrastructure that runs all AWS services. AWS maintains control over data hosted on this infrastructure, including the security configuration controls for handling customer content and personal data. AWS customers and APN partners, acting either as data controllers or data processors, are responsible for any personal data that they put in the AWS Cloud.

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

- Use multi-factor authentication (MFA) with each account.
- Use SSL/TLS to communicate with AWS resources. We recommend TLS 1.2 or later.

- Set up API and user activity logging with AWS CloudTrail.
- Use AWS encryption solutions, along with all default security controls within AWS services.
- Use advanced managed security services such as Amazon Macie, which assists in discovering and securing personal data that is stored in Amazon Simple Storage Service (Amazon S3).

We strongly recommend that you never put sensitive identifying information, such as your customers' account numbers, into free-form fields such as a `Name` field. This includes when you work with Amazon Lookout for Equipment or other AWS services using the console, API, AWS CLI, or AWS SDKs. Any data that you enter into Amazon Lookout for Equipment or other services might get picked up for inclusion in diagnostic logs. When you provide a URL to an external server, don't include credentials information in the URL to validate your request to that server.

For more information about data protection, see the AWS Shared Responsibility Model and GDPR blog post on the *AWS Security Blog*.

Topics

Encryption at rest

Amazon Lookout for Equipment encrypts your data at rest with your choice of an encryption key. You can choose one of the following:

- An AWS owned key. If you don't specify an encryption key, your data is encrypted with this key by default.
- A customer managed key. You can provide the Amazon Resource Name (ARN) of an encryption key that you created in your account. When you use a customer managed key, you must give the key a key policy that enables Amazon Lookout for Equipment to use the key. You must choose a symmetric customer managed key. Amazon Lookout for Equipment doesn't support asymmetric customer managed keys. For more information, see Key management (p. 51).
- Amazon Lookout for Equipment follows the Amazon S3 bucket encryption policy. You have to set Amazon S3 default encryption on your bucket to encrypt objects stored in your bucket by Amazon Lookout for Equipment. For more information ,see S3 bucket encryption.

Encryption in transit

Amazon Lookout for Equipment copies data out of your account and processes it in an internal AWS system. Amazon Lookout for Equipment uses TLS 1.2 with AWS certificates to encrypt data sent to other AWS services.

Key management

Amazon Lookout for Equipment encrypts your data using one of the following types of keys:

- An AWS owned key. This is the default.
- A customer managed key. You can create the key when you create an Amazon Lookout for Equipment dataset, model, or inference, or you can create the key using the AWS Key Management Service (AWS KMS) console. Choose a symmetric customer managed key, Amazon Lookout for Equipment doesn't support asymmetric customer managed keys. For more information, see Using symmetric and asymmetric keys in the *AWS Key Management Service Developer Guide*.

When you create a key using the AWS KMS console, you can give the key the following policy, which enables users or roles to use the key with Amazon Lookout for Equipment. For more information, see Using key policies in AWS KMS in the *AWS Key Management Service Developer Guide*.

```
{
    "Effect": "Allow",
    "Sid": "Allow to use the key with Amazon Lookout for Equipment",
    "Principal": {
        "AWS": "IAM USER OR ROLE ARN"
    },
    "Action": [
        "kms:DescribeKey",
        "kms:CreateGrant",
        "kms:RetireGrant"
    ],
    "Resource": "*",
    "Condition": {
        "StringEquals": {
            "kms:ViaService": [
                "lookoutequipment.Region.amazonaws.com"
            ]
        }
    }
},
{
    "Effect": "Allow",
    "Sid": "Allow to view the key in the console"
    "Principal": {
        "AWS": "IAM USER OR ROLE ARN"
    },
    "Action": [
        "kms:DescribeKey"
    ],
    "Resource": "*"
},
{
    "Effect": "Allow",
    "Sid": "Allow inference scheduler pass-in role to encrypt output data"
    "Principal": {
        "AWS": "INFERENCE SCHEDULER PASS-IN ROLE ARN"
    },
    "Action": [
        "kms:GenerateDataKey"
    ],
    "Resource": "*"
}
```

Identity and access management for Amazon Lookout for Equipment

AWS Identity and Access Management (IAM) is an AWS service that helps an administrator securely control access to AWS resources. IAM administrators control who can be *authenticated* (signed in) and *authorized* (have permissions) to use Amazon Lookout for Equipment resources. IAM is an AWS service that you can use with no additional charge.

Topics

Audience

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

Service user – If you use the Amazon Lookout for Equipment service to do your job, then your administrator provides you with the credentials and permissions that you need. As you use more Amazon Lookout for Equipment features to do your work, you might need additional permissions. Understanding how access is managed can help you request the right permissions from your administrator. If you cannot access a feature in Amazon Lookout for Equipment, see Troubleshooting Amazon Lookout for Equipment identity and access (p. 66).

Service administrator – If you're in charge of Amazon Lookout for Equipment resources at your company, you probably have full access to Amazon Lookout for Equipment. It's your job to determine which Amazon Lookout for Equipment features and resources your employees should access. You must then submit requests to your IAM administrator to change the permissions of your service users. Review the information on this page to understand the basic concepts of IAM. To learn more about how your company can use IAM with Amazon Lookout for Equipment, see AWS Identity and Access Management for Amazon Lookout for Equipment (p. 57).

IAM administrator – If you're an IAM administrator, you might want to learn details about how you can write policies to manage access to Amazon Lookout for Equipment. To view example Amazon Lookout for Equipment identity-based policies that you can use in IAM, see Identity-based policy examples for Amazon Lookout for Equipment (p. 60).

Authenticating with identities

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

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

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

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

AWS account root user

When you first create an AWS account, you begin with a single sign-in identity that has complete access to all AWS services and resources in the account. This identity is called the AWS account *root user* and is accessed by signing in with the email address and password that you used to create the account. We strongly recommend that you do not use the root user for your everyday tasks, even the administrative ones. Instead, adhere to the best practice of using the root user only to create your first IAM user. Then securely lock away the root user credentials and use them to perform only a few account and service management tasks.

IAM users and groups

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

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

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

IAM roles

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

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

- **Temporary IAM user permissions** – An IAM user can assume an IAM role to temporarily take on different permissions for a specific task.
- **Federated user access** – Instead of creating an IAM user, you can use existing identities from AWS Directory Service, your enterprise user directory, or a web identity provider. These are known as *federated users*. AWS assigns a role to a federated user when access is requested through an identity provider. For more information about federated users, see Federated users and roles in the *IAM User Guide*.
- **Cross-account access** – You can use an IAM role to allow someone (a trusted principal) in a different account to access resources in your account. Roles are the primary way to grant cross-account access. However, with some AWS services, you can attach a policy directly to a resource (instead of using a role as a proxy). To learn the difference between roles and resource-based policies for cross-account access, see How IAM roles differ from resource-based policies in the *IAM User Guide*.
- **Cross-service access** – Some AWS services use features in other AWS services. For example, when you make a call in a service, it's common for that service to run applications in Amazon EC2 or store objects in Amazon S3. A service might do this using the calling principal's permissions, using a service role, or using a service-linked role.

  - **Principal permissions** – When you use an IAM user or role to perform actions in AWS, you are considered a principal. Policies grant permissions to a principal. When you use some services, you might perform an action that then triggers another action in a different service. In this case, you must have permissions to perform both actions. To see whether an action requires additional dependent actions in a policy, see Actions, Resources, and Condition Keys for Amazon Lookout for Equipment in the *Service Authorization Reference*.
  - **Service role** – A service role is an IAM role that a service assumes to perform actions on your behalf. An IAM administrator can create, modify, and delete a service role from within IAM. For more information, see Creating a role to delegate permissions to an AWS service in the *IAM User Guide*.
  - **Service-linked role** – A service-linked role is a type of service role that is linked to an AWS service. The service can assume the role to perform an action on your behalf. Service-linked roles appear in your IAM account and are owned by the service. An IAM administrator can view, but not edit the permissions for service-linked roles.
- **Applications running on Amazon EC2** – You can use an IAM role to manage temporary credentials for applications that are running on an EC2 instance and making AWS CLI or AWS API requests. This is preferable to storing access keys within the EC2 instance. To assign an AWS role to an EC2 instance and make it available to all of its applications, you create an instance profile that is attached to the instance. An instance profile contains the role and enables programs that are running on the EC2 instance to get temporary credentials. For more information, see Using an IAM role to grant permissions to applications running on Amazon EC2 instances in the *IAM User Guide*.

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

Managing access using policies

You control access in AWS by creating policies and attaching them to IAM identities or AWS resources. A policy is an object in AWS that, when associated with an identity or resource, defines their permissions. You can sign in as the root user or an IAM user, or you can assume an IAM role. When you then make a request, AWS evaluates the related identity-based or resource-based policies. Permissions in the policies determine whether the request is allowed or denied. Most policies are stored in AWS as JSON documents. For more information about the structure and contents of JSON policy documents, see Overview of JSON policies in the *IAM User Guide*.

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

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

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

Identity-based policies

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

Identity-based policies can be further categorized as *inline policies* or *managed policies*. Inline policies are embedded directly into a single user, group, or role. Managed policies are standalone policies that

you can attach to multiple users, groups, and roles in your AWS account. Managed policies include AWS managed policies and customer managed policies. To learn how to choose between a managed policy or an inline policy, see Choosing between managed policies and inline policies in the *IAM User Guide*.

Resource-based policies

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

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

Access control lists (ACLs)

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

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

Other policy types

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

- **Permissions boundaries** – A permissions boundary is an advanced feature in which you set the maximum permissions that an identity-based policy can grant to an IAM entity (IAM user or role). You can set a permissions boundary for an entity. The resulting permissions are the intersection of entity's identity-based policies and its permissions boundaries. Resource-based policies that specify the user or role in the `Principal` field are not limited by the permissions boundary. An explicit deny in any of these policies overrides the allow. For more information about permissions boundaries, see Permissions boundaries for IAM entities in the *IAM User Guide*.
- **Service control policies (SCPs)** – SCPs are JSON policies that specify the maximum permissions for an organization or organizational unit (OU) in AWS Organizations. AWS Organizations is a service for grouping and centrally managing multiple AWS accounts that your business owns. If you enable all features in an organization, then you can apply service control policies (SCPs) to any or all of your accounts. The SCP limits permissions for entities in member accounts, including each AWS account root user. For more information about Organizations and SCPs, see How SCPs work in the *AWS Organizations User Guide*.
- **Session policies** – Session policies are advanced policies that you pass as a parameter when you programmatically create a temporary session for a role or federated user. The resulting session's permissions are the intersection of the user or role's identity-based policies and the session policies. Permissions can also come from a resource-based policy. An explicit deny in any of these policies overrides the allow. For more information, see Session policies in the *IAM User Guide*.

Multiple policy types

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

AWS Identity and Access Management for Amazon Lookout for Equipment

Before you use IAM to manage access to Amazon Lookout for Equipment, learn what IAM features are available to use with Amazon Lookout for Equipment.

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

Lookout for Equipment identity-based policies

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

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

Actions

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

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

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

Policy actions in Lookout for Equipment use the following prefix before the action: `lookoutequipment:`. For example, to grant someone permission to list Lookout for Equipment datasets with the `ListDatasets` API operation, you include the `lookoutequipment:ListDatasets` action in their policy. Policy statements must include either an `Action` or `NotAction` element. Lookout for Equipment defines its own set of actions that describe tasks that you can perform with this service.

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

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

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

```
"Action": "lookoutequipment:Describe*"
```

Resources

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

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

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

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

The Lookout for Equipment dataset resource has the following Amazon Resource Name (ARN).

```
arn:${Partition}:lookoutequipment:${Region}:${Account}:dataset/${datasetName}/${GUID}
```

For example, to specify a dataset in your statement, use the full ARN:

```
"Resource": "arn:aws:lookoutequipment:${Region}:${Account}:dataset/${datasetName}/${GUID}"
```

Some Lookout for Equipment actions, such as those for creating resources, cannot be performed on a specific resource. In those cases, you must use the wildcard (*).

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

To see a list of Lookout for Equipment resource types and their ARNs, see Resources Defined by Amazon Lookout for Equipment in the *Service Authorization Reference*. To learn with which actions you can specify the ARN of each resource, see Actions defined by Amazon Lookout for Equipment. For more information about the format of ARNs, see Amazon Resource Names (ARNs) and AWS Service Namespaces.

Condition keys

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

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

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

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

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

To view examples of Amazon Lookout for Equipment identity-based policies, see Identity-based policy examples for Amazon Lookout for Equipment (p. 60).

Access control lists (ACLs) in Lookout for Equipment

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

Access control lists (ACLs) are lists of grantees that you can attach to resources. They grant accounts permissions to access the resource to which they are attached. You can attach ACLs to an Amazon S3 *bucket* resource.

With Amazon S3 access control lists (ACLs), you can manage access to *bucket* resources. Each *bucket* has an ACL attached to it as a subresource. It defines which AWS accounts, IAM users or groups of users, or IAM roles are granted access and the type of access. When a request is received for a resource, AWS checks the corresponding ACL to verify that the requester has the necessary access permissions.

When you create a *bucket* resource, Amazon S3 creates a default ACL that grants the resource owner full control over the resource. In the following example *bucket* ACL, John Doe is listed as the owner of the *bucket* and is granted full control over that *bucket*. An ACL can have up to 100 grantees.

```
<?xml version="1.0" encoding="UTF-8"?>
<AccessControlPolicy xmlns="http://lookoutequipment.amazonaws.com/doc/2006-03-01/">
  <Owner>
    <ID>c1daexampleaaf850ea79cf0430f33d72579fd1611c97f7ded193374c0b163b6</ID>
    <DisplayName>john-doe</DisplayName>
  </Owner>
  <AccessControlList>
    <Grant>
      <Grantee xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
               xsi:type="Canonical User">
        <ID>c1daexampleaaf850ea79cf0430f33d72579fd1611c97f7ded193374c0b163b6</ID>
        <DisplayName>john-doe</DisplayName>
      </Grantee>
      <Permission>FULL_CONTROL</Permission>
    </Grant>
  </AccessControlList>
</AccessControlPolicy>
```

The ID field in the ACL is the AWS account canonical user ID. To learn how to view this ID in an account that you own, see Finding an AWS account canonical user ID.

Attribute-based access control (ABAC) with Lookout for Equipment

Attribute-based access control (ABAC) is an authorization strategy that defines permissions based on attributes. In AWS, these attributes are called *tags*. You can attach tags to IAM entities (users or roles) and to many AWS resources. Tagging entities and resources is the first step of ABAC. Then you design ABAC policies to allow operations when the principal's tag matches the tag on the resource that they are trying to access.

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

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

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

Using Temporary credentials with Lookout for Equipment

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

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

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

Cross-service principal permissions for Lookout for Equipment

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

Service roles for Lookout for Equipment

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

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

Choosing an IAM role in Lookout for Equipment

When you create a resource in Lookout for Equipment, you must choose a role to allow Lookout for Equipment to access Amazon S3 on your behalf. If you have previously created a service role or service-linked role, then Lookout for Equipment provides you with a list of roles to choose from. It's important to choose a role that allows access to read and write to your Amazon S3 bucket. instances

Identity-based policy examples for Amazon Lookout for Equipment

By default, IAM users and roles don't have permission to create or modify Amazon Lookout for Equipment resources. They also can't perform tasks using the AWS Management Console, AWS CLI, or AWS API. An IAM administrator must create IAM policies that grant users and roles permission to perform actions on the resources that they need. The administrator must then attach those policies to the IAM users or groups that require those permissions.

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

Topics

Policy best practices

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

- **Get started using AWS managed policies** – To start using Amazon Lookout for Equipment quickly, use AWS managed policies to give your employees the permissions they need. These policies are already available in your account and are maintained and updated by AWS. For more information, see Get started using permissions with AWS managed policies in the *IAM User Guide*.
- **Grant least privilege** – When you create custom policies, grant only the permissions required to perform a task. Start with a minimum set of permissions and grant additional permissions as necessary. Doing so is more secure than starting with permissions that are too lenient and then trying to tighten them later. For more information, see Grant least privilege in the *IAM User Guide*.
- **Enable MFA for sensitive operations** – For extra security, require IAM users to use multi-factor authentication (MFA) to access sensitive resources or API operations. For more information, see Using multi-factor authentication (MFA) in AWS in the *IAM User Guide*.
- **Use policy conditions for extra security** – To the extent that it's practical, define the conditions under which your identity-based policies allow access to a resource. For example, you can write conditions to specify a range of allowable IP addresses that a request must come from. You can also write conditions to allow requests only within a specified date or time range, or to require the use of SSL or MFA. For more information, see IAM JSON policy elements: Condition in the *IAM User Guide*.

Using the Lookout for Equipment console

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

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

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

Allow users to view their own permissions

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

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "ViewOwnUserInfo",
            "Effect": "Allow",
            "Action": [
                "iam:GetUserPolicy",
                "iam:ListGroupsForUser",
                "iam:ListAttachedUserPolicies",
                "iam:ListUserPolicies",
                "iam:GetUser"
```

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

Accessing a single Lookout for Equipment dataset

In this example, you grant an IAM user in your AWS account access to an Lookout for Equipment dataset.

```
{"Version": "2012-10-17",
    "Statement": [
        {"Sid": "GetAccessOfDataset",
            "Effect": "Allow",
            "Action": [
                "lookoutequipment:DescribeDataset"
            ],
            "Resource": "arn:aws:lookoutequipment:${Region}:${Account}:dataset/
${datasetName}*"
        }
    ]
}
```

Tag-based policy examples

Tag-based policies are JSON policy documents that specify the actions that a principal can perform on tagged resources.

Example: Use a tag to access a resource

This example policy grants an IAM user or role in your AWS account permission to use the `CreateDataset` operation with any resource tagged with the key `machine` and the value `myMachine1`.

```
{
  "Version": "2012-10-17",
    "Statement": [
        {"Effect": "Allow",
            "Action": [
                "lookoutequipment:CreateDataset",
                "lookoutequipment:TagResource"
            ],
            "Resource": "*",
            "Condition": {
                "StringEquals": {"aws:RequestTag/machine": "myMachine1"
                }
            }
```

```
        }
    ]
}
```

Example: Use a tag to enable Lookout for Equipment operations

This example policy grants an IAM user or role in your AWS account permission to use any Lookout for Equipment operation except the `TagResource` operation with any resource tagged with the key `machine` and the value `myMachine1`.

```
{
 "Version": "2012-10-17",
    "Statement": [
        {"Effect": "Allow",
            "Action": "lookoutequipment:*",
            "Resource": "*"
        },
        {"Effect": "Deny",
            "Action": [
                "lookoutequipment:TagResource"
            ],
            "Resource": "*",
            "Condition": {
                "StringEquals": {"aws:ResourceTag/machine": "myMachine1"
                }
            }
        }
    ]
}
```

Example: Use a tag to restrict access to an operation

This example policy restricts access for an IAM user or role in your AWS account to use the `CreateDataset` operation unless the user provides the `machine` tag and it has the allowed values `myMachine1` and `myMachine2`.

```
{
 "Version": "2012-10-17",
    "Statement": [
        {"Effect": "Allow",
            "Action": "lookoutequipment:TagResource",
            "Resource": "*"
        },
        {"Effect": "Deny",
            "Action": "lookoutequipment:CreateDataset",
            "Resource": "*",
            "Condition": {
                "Null": {
                    "aws:RequestTag/machine": "true"
                }
            }
        },
        {"Effect": "Deny",
            "Action": "lookoutequipment:CreateDataset",
            "Resource": "*",
            "Condition": {
                "ForAnyValue:StringNotEquals": {"aws:RequestTag/machine": [
                        "myMachine1",
                        "myMachine2"
                    ]
                }
            }
        }
```

```
    ]
}
```

AWS managed policies for Amazon Lookout for Equipment

To add permissions to users, groups, and roles, it is easier to use AWS managed policies than to write policies yourself. It takes time and expertise to create IAM customer managed policies that provide your team with only the permissions they need. To get started quickly, you can use our AWS managed policies. These policies cover common use cases and are available in your AWS account. For more information about AWS managed policies, see AWS managed policies in the *IAM User Guide*.

AWS services maintain and update AWS managed policies. You can't change the permissions in AWS managed policies. Services occasionally add additional permissions to an AWS managed policy to support new features. This type of update affects all identities (users, groups, and roles) where the policy is attached. Services are most likely to update an AWS managed policy when a new feature is launched or when new operations become available. Services do not remove permissions from an AWS managed policy, so policy updates won't break your existing permissions.

Additionally, AWS supports managed policies for job functions that span multiple services. For example, the **ReadOnlyAccess** AWS managed policy provides read-only access to all AWS services and resources. When a service launches a new feature, AWS adds read-only permissions for new operations and resources. For a list and descriptions of job function policies, see AWS managed policies for job functions in the *IAM User Guide*.

AWS managed policy: AmazonLookoutEquipmentReadOnlyAccess

You can attach AmazonLookoutEquipmentReadOnlyAccess to your IAM entities. Lookout for Equipment also attaches this policy to a service role that allows Lookout for Equipment to perform actions on your behalf.

This policy grants *read-only* permissions that allow read-only access to all Lookout for Equipment resources.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "lookoutequipment:DescribeDataset",
                "lookoutequipment:DescribeDataIngestionJob",
                "lookoutequipment:DescribeModel",
                "lookoutequipment:DescribeInferenceScheduler",
                "lookoutequipment:ListDatasets",
                "lookoutequipment:ListDataIngestionJobs",
                "lookoutequipment:ListModels",
```

```
                "lookoutequipment:ListInferenceSchedulers",
                "lookoutequipment:ListInferenceExecutions",
                "lookoutequipment:ListTagsForResource"
            ],
            "Resource": "*"
        }
}
```

AWS managed policy: AmazonLookoutEquipmentFullAccess

You can attach AmazonLookoutEquipmentFullAccess to your IAM entities. Lookout for Equipment also attaches this policy to a service role that allows Lookout for Equipment to perform actions on your behalf.

This policy grants *administrative* permissions that allow access to all Lookout for Equipment resources and operations. This policy enables you to use any IAM role or AWS KMS key with Lookout for Equipment.

```
{
 "Version": "2012-10-17",
 "Statement": [
  {
   "Effect": "Allow",
   "Action": [
    "lookoutequipment:*"
   ],
   "Resource": "*"
  },
  {
   "Effect": "Allow",
   "Action": [
    "iam:PassRole"
   ],
   "Resource": "*",
   "Condition": {
    "StringEquals": {
     "iam:PassedToService": [
      "lookoutequipment.amazonaws.com"
     ]
    }
   }
  },
  {
   "Effect": "Allow",
   "Action": [
    "kms:CreateGrant"
   ],
   "Resource": "*",
   "Condition": {
    "StringLike": {
     "kms:ViaService": "lookoutequipment.*.amazonaws.com"
    }
   }
  },
  {
   "Effect": "Allow",
   "Action": [
```

```
        "kms:DescribeKey",
                                    "kms:ListAliases"
      ],
      "Resource": "*"
     }
    ]
   }
```

Lookout for Equipment updates to AWS managed policies

View details about updates to AWS managed policies for Lookout for Equipment since this service began tracking these changes. For automatic alerts about changes to this page, subscribe to the RSS feed on the Lookout for Equipment Document history page.

Change	Description	Date
AmazonLookoutEquipmentFullAccess (p. 65) – Update to grant retirement policy	Lookout for Equipment removed RetireGrant from the managed policy as the service will be using retiring grant principal to retire the grants. You dont need to provide the retire grant permissions in the managed policy.	November 22, 2021
AmazonLookoutEquipmentReadOnlyAccess (p. 64) – New policy	Lookout for Equipment added a new policy to allow read only access for all Lookout for Equipment resources.	May 05, 2021
AmazonLookoutEquipmentFullAccess (p. 65) – Update to an existing policy	Lookout for Equipment added permissions to describe AWS KMS managed encryption keys. You must use these permissions to use the Lookout for Equipment console to display information about AWS KMS keys across AWS accounts.	May 05, 2021
Lookout for Equipment started tracking changes	Lookout for Equipment started tracking changes for its AWS managed policies.	April 08, 2021

Troubleshooting Amazon Lookout for Equipment identity and access

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

Topics

I am not authorized to perform an action in Lookout for Equipment

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

The following example error occurs when the `mateojackson` IAM user tries to use the console to view details about a fictional dataset but does not have the fictional `lookoutequipment:DescribeDataset` permissions.

```
User: arn:aws:iam::123456789012:user/mateojackson is not authorized to perform:
 lookoutequipment:DescribeDataset on resource: my-example-dataset
```

In this case, Mateo asks his administrator to update his policies to allow him to access the dataset using the `lookoutequipment:DescribeDataset` action.

I am not authorized to perform iam:PassRole

If you receive an error that you're not authorized to perform the `iam:PassRole` action, then you must contact your administrator for assistance. Your administrator is the person that provided you with your user name and password. Ask that person to update your policies to allow you to pass a role to Amazon Lookout for Equipment.

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

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

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

In this case, Mary asks her administrator to update her policies to allow her to perform the `iam:PassRole` action.

I want to view my access keys

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

Access keys consist of two parts: an access key ID (for example, `AKIAIOSFODNN7EXAMPLE`) and a secret access key (for example, `wJalrXUtnFEMI/K7MDENG/bPxRfiCYEXAMPLEKEY`). Like a user name and

password, you must use both the access key ID and secret access key together to authenticate your requests. Manage your access keys as securely as you do your user name and password.

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

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

I'm an administrator and want to allow others to access Lookout for Equipment

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

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

I want to allow people outside of my AWS account to access my Lookout for Equipment resources

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

To learn more, consult the following:

- To learn whether Amazon Lookout for Equipment supports these features, see AWS Identity and Access Management for Amazon Lookout for Equipment (p. 57).
- To learn how to provide access to your resources across AWS accounts that you own, see Providing access to an IAM user in another AWS account that you own in the *IAM User Guide*.
- To learn how to provide access to your resources to third-party AWS accounts, see Providing access to AWS accounts owned by third parties in the *IAM User Guide*.
- To learn how to provide access through identity federation, see Providing access to externally authenticated users (identity federation) in the *IAM User Guide*.
- To learn the difference between using roles and resource-based policies for cross-account access, see How IAM roles differ from resource-based policies in the *IAM User Guide*.

Monitoring Amazon Lookout for Equipment

Monitoring is an important part of maintaining the reliability, availability, and performance of your Lookout for Equipment applications. To monitor Lookout for Equipment API calls, you can use AWS CloudTrail.

Topics

- Monitoring with Amazon CloudWatch (p. 71)

Monitoring Amazon Lookout for Equipment with AWS CloudTrail

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

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

Amazon Lookout for Equipment information in CloudTrail

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

For an ongoing record of events in your AWS account, including events for Amazon Lookout for Equipment, create a trail. A trail is a configuration that enables CloudTrail to deliver events as log files to a specified 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 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:

- Creating a trail for your AWS account
- CloudTrail Supported Services and Integrations
- Configuring Amazon SNS Notifications for CloudTrail
- Receiving CloudTrail Log Files from Multiple Regions
- Receiving CloudTrail Log Files from Multiple Accounts

CloudTrail logs all Amazon Lookout for Equipment actions. For example, calls to the StartDataIngestionJob (p. 130), CreateModel (p. 86), and CreateInferenceScheduler (p. 81) operations generate entries in the CloudTrail log files.

Every event or log entry contains information about who generated the request. For more information, see the CloudTrail userIdentify Element.

A trail is a configuration that enables delivery of events as log files to a specified S3 bucket. 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 aren't an ordered stack trace of the public API calls, so they don't appear in any specific order.

The following example shows the log entry created by using the `ListDatasets` operation.

```
{
   "eventVersion":"1.05",
   "userIdentity":{
      "type":"AssumedRole | FederatedUser | IAMUser | Root | SAMLUser | WebIdentityUser",
      "principalId":"principal ID",
      "arn":"ARN",
      "accountId":"account ID",
      "accessKeyId":"access key",
      "sessionContext":{
         "sessionIssuer":{
            "type":"Role",
            "principalId":"principal ID",
            "arn":"ARN",
            "accountId":"account ID",
            "userName":"user name"
         },
         "attributes":{
            "mfaAuthenticated":"true | false",
            "creationDate":"timestamp"
         }
      }
   },
   "eventTime":"timestamp",
   "eventSource":"lookoutequipment.amazonaws.com",
   "eventName":"ListDatasets",
   "awsRegion":"region",
   "sourceIPAddress":"source IP address",
   "userAgent":"user agent",
   "requestParameters": null,
   "responseElements":null,
   "requestID":"request ID",
   "eventID":"event ID",
   "readOnly":"true | false",
   "eventType":"AwsApiCall",
   "recipientAccountId":"account id"
}
```

The following example shows the log entry created by using the DescribeDataset (p. 100) operation.

```
{
   "eventVersion":"1.05",
   "userIdentity":{
      "type":"AssumedRole | FederatedUser | IAMUser | Root | SAMLUser | WebIdentityUser",
      "principalId":"principal ID",
      "arn":"ARN",
      "accountId":"account ID",
      "accessKeyId":"access key",
      "sessionContext":{
         "sessionIssuer":{
            "type":"AssumedRole | FederatedUser | IAMUser | Root | SAMLUser |
 WebIdentityUser",
            "principalId":"principal ID",
            "arn":"ARN",
            "accountId":"account ID",
            "userName":"user name"
         },
         "attributes":{
            "mfaAuthenticated":"true | false",
            "creationDate":"timestamp"
         }
      }
```

```
   },
   "eventTime":"timestamp",
   "eventSource":"lookoutequipment.amazonaws.com",
   "eventName":"DescribeDataset",
   "awsRegion":"region",
   "sourceIPAddress":"source IP address",
   "userAgent":"user agent",
   "requestParameters":{
      "datasetName":"dataset name"
   },
   "responseElements":null,
   "requestID":"request ID",
   "eventID":"event ID",
   "readOnly":"true | false",
   "eventType":"AwsApiCall",
   "recipientAccountId":"account id"
}
```

Manual monitoring tools

Another important part of monitoring Amazon Lookout for Equipment involves manually monitoring those items that the CloudWatch alarms don't cover. The Amazon Lookout for Equipment, Amazon CloudWatch, AWS Trusted Advisor, and other AWS console dashboards provide an at-a-glance view of the state of your AWS environment.

- The CloudWatch home page shows:
  - Current alarms and status
  - Graphs of alarms and resources
  - Service health status

  In addition, you can use CloudWatch to do the following:
  - Create customized dashboards to monitor the services you care about
  - Graph metric data to troubleshoot issues and discover trends
  - Search and browse all your AWS resource metrics
  - Create and edit alarms to be notified of problems
- Trusted Advisor can help you monitor your AWS resources to improve performance, reliability, security, and cost effectiveness. Four Trusted Advisor checks are available to all users; and more than 50 checks are available to users with a Business or Enterprise support plan. For more information, see AWS Trusted Advisor.

Monitoring with Amazon CloudWatch

You can monitor *Amazon Lookout for Equipment* 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.

When you create an inference scheduler, Amazon Lookout for Equipment sends the following metrics and dimensions to CloudWatch every 5 minutes. You can use the following procedures to view the metrics for Amazon Lookout for Equipment. Metrics are grouped first by the service namespace, and then by the various dimension combinations within each namespace.

The *Amazon Lookout for Equipment* service reports the following metrics in the `AWS/LookoutEquipment` namespace.

CloudWatch displays the following metrics for Amazon Lookout for Equipment:

Metric	Description
`InferenceSucceeded`	If the value is `1`, the inference succeeded. If the value is `0`, the inference failed. ModelName: The name of the model. InferenceSchedulerName: Name of the inference scheduler
`InferenceFailed`	If the value is `1`, the inference failed. If the value is `0`, the inference succeeded. ModelName: The name of the model. InferenceSchedulerName: Name of the inference scheduler
`InferenceInvalidInput`	If the value is `1`, you've provided invalid data or an invalid configuration for the inference. An invalid configuration could be invalid permissions, an invalid S3 bucket, or invalid KMS keys. ModelName: The name of the model. InferenceSchedulerName: Name of the inference scheduler

The following dimensions are supported for the *`Amazon Lookout for Equipment`* metrics.

Dimension	Description
ModelName	The name of the ML model that you've trained to monitor your equipment.
`InferenceSchedulerName`	The inference scheduler schedules the times when your model monitors your equipment.

You can use the following procedure to view metrics using the Amazon Lookout for Equipment console.

To view metrics using the CloudWatch console

1. Open the CloudWatch console at https://console.aws.amazon.com/cloudwatch/.
2. In the navigation pane, choose **Metrics**.
3. In the **CloudWatch Metrics by Category** section, under the metrics category for Amazon Lookout for Equipment, choose a metrics category, and then in the upper pane, scroll down to view the full list of metrics.

You can use the following command to view metrics with the AWS CLI.

To view metrics using the AWS CLI

- At a command prompt, use the following command:

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

Creating CloudWatch alarms to monitor Amazon Lookout for Equipment

You can create a CloudWatch alarm that sends an Amazon Simple Notification Service (Amazon SNS) message when the alarm changes state. An alarm watches a single metric over a time period you specify, and performs one or more actions based on the value of the metric relative to a given threshold over a number of time periods. The action is a notification sent to an Amazon SNS topic or Amazon EC2 Auto Scaling policy. Alarms invoke actions for sustained state changes only. CloudWatch alarms do not invoke actions simply because they are in a particular state; the state must have changed and been maintained for a specified number of periods.

Amazon Lookout for Equipment and interface VPC endpoints (AWS PrivateLink)

You can establish a private connection between your VPC and Amazon Lookout for Equipment by creating an *interface VPC endpoint*. Interface endpoints are powered by AWS PrivateLink, a technology that you can use to privately access Lookout for Equipment APIs without an internet gateway, network address translation (NAT) device, VPN connection, or AWS Direct Connect connection. Instances in your VPC don't need public IP addresses to communicate with Lookout for Equipment APIs. Traffic between your VPC and Lookout for Equipment does not leave the Amazon network.

Each interface endpoint is represented by one or more Elastic Network Interfaces in your subnets.

For more information, see Interface VPC endpoints (AWS PrivateLink) in the *Amazon VPC User Guide*.

Considerations for Lookout for Equipment VPC endpoints

Before you set up an interface VPC endpoint for Lookout for Equipment, ensure that you review Interface endpoint properties and limitations in the *Amazon VPC User Guide*.

Lookout for Equipment supports making calls to all of its API actions from your VPC.

Creating an interface VPC endpoint for Lookout for Equipment

You can create a VPC endpoint for the Lookout for Equipment service using either the Amazon VPC console or the AWS Command Line Interface (AWS CLI). For more information, see Creating an interface endpoint in the *Amazon VPC User Guide*.

Create a VPC endpoint for Lookout for Equipment using the following service name:

- com.amazonaws.*region*.lookoutequipment

If you enable private DNS for the endpoint, you can make API requests to Lookout for Equipment using its default DNS name for the Region, for example, `lookoutequipment.us-east-1.amazonaws.com`.

For more information, see Accessing a service through an interface endpoint in the *Amazon VPC User Guide*.

Creating a VPC endpoint policy for Lookout for Equipment

You can attach an endpoint policy to your VPC endpoint that controls access to Lookout for Equipment. The policy specifies the following information:

- The principal that can perform actions.
- The actions that can be performed.
- The resources on which actions can be performed.

For more information, see Controlling access to services with VPC endpoints in the *Amazon VPC User Guide*.

Example: VPC endpoint policy for Lookout for Equipment actions

The following is an example of an endpoint policy for Lookout for Equipment. When attached to an endpoint, this policy grants access to the listed Lookout for Equipment actions for all principals on all resources.

```
{
   "Statement":[
      {
         "Principal":"*",
         "Effect":"Allow",
         "Action":[
            "lookoutequipment:ListDatasets",
            "lookoutequipment:CreateDataset",
            "lookoutequipment:DescribeDataset",
            "lookoutequipment:DeleteDataset",
            "lookoutequipment:StartDataIngestionJob",
            "lookoutequipment:DescribeDataIngestionJob",
            "lookoutequipment:ListDataIngestionJobs",
            "lookoutequipment:CreateModel",
            "lookoutequipment:DescribeModel",
            "lookoutequipment:ListModels",
            "lookoutequipment:DeleteModel",
            "lookoutequipment:CreateInferenceScheduler",
            "lookoutequipment:StartInferenceScheduler",
            "lookoutequipment:StopInferenceScheduler",
            "lookoutequipment:UpdateInferenceScheduler",
            "lookoutequipment:DescribeInferenceScheduler",
            "lookoutequipment:ListInferenceSchedulers",
            "lookoutequipment:DeleteInferenceScheduler",
            "lookoutequipment:ListInferenceExecutions"
         ],
         "Resource":"*"
      }
   ]
}
```

Compliance validation for Amazon Lookout for Equipment

Third-party auditors assess the security and compliance of Amazon Lookout for Equipment as part of multiple AWS compliance programs. Amazon Lookout for Equipment is a data compliant service.

For a list of AWS services in scope of specific compliance programs, see AWS Services in Scope by Compliance Program. For general information, see AWS Compliance Programs. You can download third-party audit reports using AWS Artifact. For more information, see Downloading Reports in AWS Artifact.

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

- Security and Compliance Quick Start Guides – These deployment guides discuss architectural considerations and provide steps for deploying security- and compliance-focused baseline environments on AWS.
- Architecting for HIPAA Security and Compliance Whitepaper – This whitepaper describes how companies can use AWS to create HIPAA-compliant applications.
- AWS Compliance Resources – This collection of workbooks and guides might apply to your industry and location.
- Evaluating Resources with Rules in the AWS Config Developer Guide– The AWS Config service assesses how well your resource configurations comply with internal practices, industry guidelines, and regulations.
- Security Hub – This AWS service provides a comprehensive view of your security state within AWS that helps you check your compliance with security industry standards and best practices.

Resilience in Amazon Lookout for Equipment

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

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

Infrastructure security in Amazon Lookout for Equipment

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

You use published AWS API calls to access Amazon Lookout for Equipment through the network. Clients must support Transport Layer Security (TLS) 1.2 or later. Clients must also support cipher suites with perfect forward secrecy (PFS) such as Ephemeral Diffie-Hellman (DHE) or Elliptic Curve Ephemeral Diffie-Hellman (ECDHE). Most modern systems such as Java 7 and later support these modes.

Requests must be signed by using an access key ID and a secret access key that is associated with an IAM principal. If you don't have an access key and a secret access key, you can use the AWS Security Token Service to generate temporary security credentials to sign requests.

Creating Amazon Lookout for Equipment resources with AWS CloudFormation

Amazon Lookout for Equipment is integrated with AWS CloudFormation, a service that helps you to model and set up your AWS resources so that you can spend less time creating and managing your Lookout for Equipment resources and infrastructure. You create a template that describes all the AWS resources that you want (such as Amazon S3 buckets), and AWS CloudFormation provisions and configures those resources for you.

When you use AWS CloudFormation, you can reuse your template to set up your Lookout for Equipment resources consistently and repeatedly. Describe your resources once, and then provision the same resources over and over in multiple AWS accounts and Regions.

Lookout for Equipment and AWS CloudFormation templates

To provision and configure resources for Lookout for Equipment and related services, you must understand AWS CloudFormation templates. Templates are formatted text files in JSON or YAML. These templates describe the resources that you want to provision in your AWS CloudFormation stacks. If you're unfamiliar with JSON or YAML, you can use AWS CloudFormation Designer to help you get started with AWS CloudFormation templates. For more information, see What is AWS CloudFormation Designer? in the *AWS CloudFormation User Guide*.

Lookout for Equipment supports creating Amazon S3 buckets in AWS CloudFormation. For more information, including examples of JSON and YAML templates for Amazon S3 buckets, see the Lookout For Equipment resource type reference in the *AWS CloudFormation User Guide*.

Learn more about AWS CloudFormation

To learn more about AWS CloudFormation, see the following resources:

- AWS CloudFormation
- AWS CloudFormation User Guide
- AWS CloudFormation API Reference
- AWS CloudFormation Command Line Interface User Guide

API Reference

This section contains the API Reference documentation.

Actions

The following actions are supported:

CreateDataset

Creates a container for a collection of data being ingested for analysis. The dataset contains the metadata describing where the data is and what the data actually looks like. In other words, it contains the location of the data source, the data schema, and other information. A dataset also contains any tags associated with the ingested data.

Request Syntax

```
{
   "ClientToken": "string",
   "DatasetName": "string",
   "DatasetSchema": {
      "InlineDataSchema": "string"
   },
   "ServerSideKmsKeyId": "string",
   "Tags": [
      {
         "Key": "string",
         "Value": "string"
      }
   ]
}
```

Request Parameters

The request accepts the following data in JSON format.

ClientToken (p. 78)

A unique identifier for the request. If you do not set the client request token, Amazon Lookout for Equipment generates one.

Type: String

Length Constraints: Minimum length of 1. Maximum length of 256.

Pattern: `\p{ASCII}{1,256}`

Required: Yes

DatasetName (p. 78)

The name of the dataset being created.

Type: String

Length Constraints: Minimum length of 1. Maximum length of 200.

Pattern: `^[0-9a-zA-Z_-]{1,200}$`

Required: Yes

DatasetSchema (p. 78)

A JSON description of the data that is in each time series dataset, including names, column names, and data types.

Type: DatasetSchema (p. 150) object

Required: Yes

ServerSideKmsKeyId (p. 78)

Provides the identifier of the AWS KMS key used to encrypt dataset data by Amazon Lookout for Equipment.

Type: String

Length Constraints: Minimum length of 1. Maximum length of 2048.

Pattern: `^[A-Za-z0-9][A-Za-z0-9:_/+=,@.-]{0,2048}$`

Required: No

Tags (p. 78)

Any tags associated with the ingested data described in the dataset.

Type: Array of Tag (p. 169) objects

Array Members: Minimum number of 0 items. Maximum number of 200 items.

Required: No

Response Syntax

```
{
   "DatasetArn": "string",
   "DatasetName": "string",
   "Status": "string"
}
```

Response Elements

If the action is successful, the service sends back an HTTP 200 response.

The following data is returned in JSON format by the service.

DatasetArn (p. 79)

The Amazon Resource Name (ARN) of the dataset being created.

Type: String

Length Constraints: Minimum length of 20. Maximum length of 2048.

Pattern: `arn:aws(-[^:]+)?:lookoutequipment:[a-zA-Z0-9\-]*:[0-9]{12}:dataset\/.+`

DatasetName (p. 79)

The name of the dataset being created.

Type: String

Length Constraints: Minimum length of 1. Maximum length of 200.

Pattern: `^[0-9a-zA-Z_-]{1,200}$`

Status (p. 79)

Indicates the status of the `CreateDataset` operation.

Type: String

Valid Values: `CREATED | INGESTION_IN_PROGRESS | ACTIVE`

Errors

AccessDeniedException

The request could not be completed because you do not have access to the resource.

HTTP Status Code: 400

ConflictException

The request could not be completed due to a conflict with the current state of the target resource.

HTTP Status Code: 400

InternalServerException

Processing of the request has failed because of an unknown error, exception or failure.

HTTP Status Code: 500

ServiceQuotaExceededException

Resource limitations have been exceeded.

HTTP Status Code: 400

ThrottlingException

The request was denied due to request throttling.

HTTP Status Code: 400

ValidationException

The input fails to satisfy constraints specified by Amazon Lookout for Equipment or a related AWS service that's being utilized.

HTTP Status Code: 400

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS Command Line Interface
- AWS SDK for .NET
- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Java V2
- AWS SDK for JavaScript
- AWS SDK for PHP V3
- AWS SDK for Python
- AWS SDK for Ruby V3

CreateInferenceScheduler

Creates a scheduled inference. Scheduling an inference is setting up a continuous real-time inference plan to analyze new measurement data. When setting up the schedule, you provide an S3 bucket location for the input data, assign it a delimiter between separate entries in the data, set an offset delay if desired, and set the frequency of inferencing. You must also provide an S3 bucket location for the output data.

Request Syntax

```
{
   "ClientToken": "string",
   "DataDelayOffsetInMinutes": number,
   "DataInputConfiguration": {
      "InferenceInputNameConfiguration": {
         "ComponentTimestampDelimiter": "string",
         "TimestampFormat": "string"
      },
      "InputTimeZoneOffset": "string",
      "S3InputConfiguration": {
         "Bucket": "string",
         "Prefix": "string"
      }
   },
   "DataOutputConfiguration": {
      "KmsKeyId": "string",
      "S3OutputConfiguration": {
         "Bucket": "string",
         "Prefix": "string"
      }
   },
   "DataUploadFrequency": "string",
   "InferenceSchedulerName": "string",
   "ModelName": "string",
   "RoleArn": "string",
   "ServerSideKmsKeyId": "string",
   "Tags": [
      {
         "Key": "string",
         "Value": "string"
      }
   ]
}
```

Request Parameters

The request accepts the following data in JSON format.

ClientToken (p. 81)

A unique identifier for the request. If you do not set the client request token, Amazon Lookout for Equipment generates one.

Type: String

Length Constraints: Minimum length of 1. Maximum length of 256.

Pattern: `\p{ASCII}{1,256}`

Required: Yes

DataDelayOffsetInMinutes (p. 81)

A period of time (in minutes) by which inference on the data is delayed after the data starts. For instance, if you select an offset delay time of five minutes, inference will not begin on the data until the first data measurement after the five minute mark. For example, if five minutes is selected, the inference scheduler will wake up at the configured frequency with the additional five minute delay time to check the customer S3 bucket. The customer can upload data at the same frequency and they don't need to stop and restart the scheduler when uploading new data.

Type: Long

Valid Range: Minimum value of 0. Maximum value of 60.

Required: No

DataInputConfiguration (p. 81)

Specifies configuration information for the input data for the inference scheduler, including delimiter, format, and dataset location.

Type: InferenceInputConfiguration (p. 155) object

Required: Yes

DataOutputConfiguration (p. 81)

Specifies configuration information for the output results for the inference scheduler, including the S3 location for the output.

Type: InferenceOutputConfiguration (p. 157) object

Required: Yes

DataUploadFrequency (p. 81)

How often data is uploaded to the source S3 bucket for the input data. The value chosen is the length of time between data uploads. For instance, if you select 5 minutes, Amazon Lookout for Equipment will upload the real-time data to the source bucket once every 5 minutes. This frequency also determines how often Amazon Lookout for Equipment starts a scheduled inference on your data. In this example, it starts once every 5 minutes.

Type: String

Valid Values: `PT5M | PT10M | PT15M | PT30M | PT1H`

Required: Yes

InferenceSchedulerName (p. 81)

The name of the inference scheduler being created.

Type: String

Length Constraints: Minimum length of 1. Maximum length of 200.

Pattern: `^[0-9a-zA-Z_-]{1,200}$`

Required: Yes

ModelName (p. 81)

The name of the previously trained ML model being used to create the inference scheduler.

Type: String

Length Constraints: Minimum length of 1. Maximum length of 200.

Pattern: `^[0-9a-zA-Z_-]{1,200}$`

Required: Yes

RoleArn (p. 81)

The Amazon Resource Name (ARN) of a role with permission to access the data source being used for the inference.

Type: String

Length Constraints: Minimum length of 20. Maximum length of 2048.

Pattern: `arn:aws(-[^:]+)?:iam::[0-9]{12}:role/.+`

Required: Yes

ServerSideKmsKeyId (p. 81)

Provides the identifier of the AWS KMS key used to encrypt inference scheduler data by Amazon Lookout for Equipment.

Type: String

Length Constraints: Minimum length of 1. Maximum length of 2048.

Pattern: `^[A-Za-z0-9][A-Za-z0-9:_/+=,@.-]{0,2048}$`

Required: No

Tags (p. 81)

Any tags associated with the inference scheduler.

Type: Array of Tag (p. 169) objects

Array Members: Minimum number of 0 items. Maximum number of 200 items.

Required: No

Response Syntax

```
{
   "InferenceSchedulerArn": "string",
   "InferenceSchedulerName": "string",
   "Status": "string"
}
```

Response Elements

If the action is successful, the service sends back an HTTP 200 response.

The following data is returned in JSON format by the service.

InferenceSchedulerArn (p. 83)

The Amazon Resource Name (ARN) of the inference scheduler being created.

Type: String

Length Constraints: Minimum length of 20. Maximum length of 2048.

Pattern: `arn:aws(-[^:]+)?:lookoutequipment:[a-zA-Z0-9\-]*:[0-9]{12}:inference-scheduler\/.+`

InferenceSchedulerName (p. 83)

The name of inference scheduler being created.

Type: String

Length Constraints: Minimum length of 1. Maximum length of 200.

Pattern: `^[0-9a-zA-Z_-]{1,200}$`

Status (p. 83)

Indicates the status of the `CreateInferenceScheduler` operation.

Type: String

Valid Values: `PENDING | RUNNING | STOPPING | STOPPED`

Errors

AccessDeniedException

The request could not be completed because you do not have access to the resource.

HTTP Status Code: 400

ConflictException

The request could not be completed due to a conflict with the current state of the target resource.

HTTP Status Code: 400

InternalServerException

Processing of the request has failed because of an unknown error, exception or failure.

HTTP Status Code: 500

ResourceNotFoundException

The resource requested could not be found. Verify the resource ID and retry your request.

HTTP Status Code: 400

ServiceQuotaExceededException

Resource limitations have been exceeded.

HTTP Status Code: 400

ThrottlingException

The request was denied due to request throttling.

HTTP Status Code: 400

ValidationException

The input fails to satisfy constraints specified by Amazon Lookout for Equipment or a related AWS service that's being utilized.

HTTP Status Code: 400

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS Command Line Interface
- AWS SDK for .NET
- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Java V2
- AWS SDK for JavaScript
- AWS SDK for PHP V3
- AWS SDK for Python
- AWS SDK for Ruby V3

CreateModel

Creates an ML model for data inference.

A machine-learning (ML) model is a mathematical model that finds patterns in your data. In Amazon Lookout for Equipment, the model learns the patterns of normal behavior and detects abnormal behavior that could be potential equipment failure (or maintenance events). The models are made by analyzing normal data and abnormalities in machine behavior that have already occurred.

Your model is trained using a portion of the data from your dataset and uses that data to learn patterns of normal behavior and abnormal patterns that lead to equipment failure. Another portion of the data is used to evaluate the model's accuracy.

Request Syntax

```
{
   "ClientToken": "string",
   "DataPreProcessingConfiguration": {
      "TargetSamplingRate": "string"
   },
   "DatasetName": "string",
   "DatasetSchema": {
      "InlineDataSchema": "string"
   },
   "EvaluationDataEndTime": number,
   "EvaluationDataStartTime": number,
   "LabelsInputConfiguration": {
      "S3InputConfiguration": {
         "Bucket": "string",
         "Prefix": "string"
      }
   },
   "ModelName": "string",
   "OffCondition": "string",
   "RoleArn": "string",
   "ServerSideKmsKeyId": "string",
   "Tags": [
      {
         "Key": "string",
         "Value": "string"
      }
   ],
   "TrainingDataEndTime": number,
   "TrainingDataStartTime": number
}
```

Request Parameters

The request accepts the following data in JSON format.

ClientToken (p. 86)

A unique identifier for the request. If you do not set the client request token, Amazon Lookout for Equipment generates one.

Type: String

Length Constraints: Minimum length of 1. Maximum length of 256.

Pattern: `\p{ASCII}{1,256}`

Required: Yes

DataPreProcessingConfiguration (p. 86)

The configuration is the `TargetSamplingRate`, which is the sampling rate of the data after post processing by Amazon Lookout for Equipment. For example, if you provide data that has been collected at a 1 second level and you want the system to resample the data at a 1 minute rate before training, the `TargetSamplingRate` is 1 minute.

When providing a value for the `TargetSamplingRate`, you must attach the prefix "PT" to the rate you want. The value for a 1 second rate is therefore *PT1S*, the value for a 15 minute rate is *PT15M*, and the value for a 1 hour rate is *PT1H*

Type: DataPreProcessingConfiguration (p. 149) object

Required: No

DatasetName (p. 86)

The name of the dataset for the ML model being created.

Type: String

Length Constraints: Minimum length of 1. Maximum length of 200.

Pattern: `^[0-9a-zA-Z_-]{1,200}$`

Required: Yes

DatasetSchema (p. 86)

The data schema for the ML model being created.

Type: DatasetSchema (p. 150) object

Required: No

EvaluationDataEndTime (p. 86)

Indicates the time reference in the dataset that should be used to end the subset of evaluation data for the ML model.

Type: Timestamp

Required: No

EvaluationDataStartTime (p. 86)

Indicates the time reference in the dataset that should be used to begin the subset of evaluation data for the ML model.

Type: Timestamp

Required: No

LabelsInputConfiguration (p. 86)

The input configuration for the labels being used for the ML model that's being created.

Type: LabelsInputConfiguration (p. 164) object

Required: No

ModelName (p. 86)

The name for the ML model to be created.

Type: String

Length Constraints: Minimum length of 1. Maximum length of 200.

Pattern: `^[0-9a-zA-Z_-]{1,200}$`

Required: Yes

OffCondition (p. 86)

Indicates that the asset associated with this sensor has been shut off. As long as this condition is met, Lookout for Equipment will not use data from this asset for training, evaluation, or inference.

Type: String

Length Constraints: Minimum length of 1. Maximum length of 2048.

Required: No

RoleArn (p. 86)

The Amazon Resource Name (ARN) of a role with permission to access the data source being used to create the ML model.

Type: String

Length Constraints: Minimum length of 20. Maximum length of 2048.

Pattern: `arn:aws(-[^:]+)?:iam::[0-9]{12}:role/.+`

Required: No

ServerSideKmsKeyId (p. 86)

Provides the identifier of the AWS KMS key used to encrypt model data by Amazon Lookout for Equipment.

Type: String

Length Constraints: Minimum length of 1. Maximum length of 2048.

Pattern: `^[A-Za-z0-9][A-Za-z0-9:_/+=,@.-]{0,2048}$`

Required: No

Tags (p. 86)

Any tags associated with the ML model being created.

Type: Array of Tag (p. 169) objects

Array Members: Minimum number of 0 items. Maximum number of 200 items.

Required: No

TrainingDataEndTime (p. 86)

Indicates the time reference in the dataset that should be used to end the subset of training data for the ML model.

Type: Timestamp

Required: No

TrainingDataStartTime (p. 86)

Indicates the time reference in the dataset that should be used to begin the subset of training data for the ML model.

Type: Timestamp

Required: No

Response Syntax

```
{
   "ModelArn": "string",
   "Status": "string"
}
```

Response Elements

If the action is successful, the service sends back an HTTP 200 response.

The following data is returned in JSON format by the service.

ModelArn (p. 89)

The Amazon Resource Name (ARN) of the model being created.

Type: String

Length Constraints: Minimum length of 20. Maximum length of 2048.

Pattern: `arn:aws(-[^:]+)?:lookoutequipment:[a-zA-Z0-9\-]*:[0-9]{12}:model\/.+`

Status (p. 89)

Indicates the status of the `CreateModel` operation.

Type: String

Valid Values: `IN_PROGRESS | SUCCESS | FAILED`

Errors

AccessDeniedException

The request could not be completed because you do not have access to the resource.

HTTP Status Code: 400

ConflictException

The request could not be completed due to a conflict with the current state of the target resource.

HTTP Status Code: 400

InternalServerException

Processing of the request has failed because of an unknown error, exception or failure.

HTTP Status Code: 500

ResourceNotFoundException

The resource requested could not be found. Verify the resource ID and retry your request.

HTTP Status Code: 400

ServiceQuotaExceededException

Resource limitations have been exceeded.

HTTP Status Code: 400

ThrottlingException

The request was denied due to request throttling.

HTTP Status Code: 400

ValidationException

The input fails to satisfy constraints specified by Amazon Lookout for Equipment or a related AWS service that's being utilized.

HTTP Status Code: 400

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS Command Line Interface
- AWS SDK for .NET
- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Java V2
- AWS SDK for JavaScript
- AWS SDK for PHP V3
- AWS SDK for Python
- AWS SDK for Ruby V3

DeleteDataset

Deletes a dataset and associated artifacts. The operation will check to see if any inference scheduler or data ingestion job is currently using the dataset, and if there isn't, the dataset, its metadata, and any associated data stored in S3 will be deleted. This does not affect any models that used this dataset for training and evaluation, but does prevent it from being used in the future.

Request Syntax

```
{
   "DatasetName": "string"
}
```

Request Parameters

The request accepts the following data in JSON format.

DatasetName (p. 91)

The name of the dataset to be deleted.

Type: String

Length Constraints: Minimum length of 1. Maximum length of 200.

Pattern: `^[0-9a-zA-Z_-]{1,200}$`

Required: Yes

Response Elements

If the action is successful, the service sends back an HTTP 200 response with an empty HTTP body.

Errors

AccessDeniedException

The request could not be completed because you do not have access to the resource.

HTTP Status Code: 400

ConflictException

The request could not be completed due to a conflict with the current state of the target resource.

HTTP Status Code: 400

InternalServerException

Processing of the request has failed because of an unknown error, exception or failure.

HTTP Status Code: 500

ResourceNotFoundException

The resource requested could not be found. Verify the resource ID and retry your request.

HTTP Status Code: 400

ThrottlingException

The request was denied due to request throttling.

HTTP Status Code: 400

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS Command Line Interface
- AWS SDK for .NET
- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Java V2
- AWS SDK for JavaScript
- AWS SDK for PHP V3
- AWS SDK for Python
- AWS SDK for Ruby V3

DeleteInferenceScheduler

Deletes an inference scheduler that has been set up. Already processed output results are not affected.

Request Syntax

```
{
   "InferenceSchedulerName": "string"
}
```

Request Parameters

The request accepts the following data in JSON format.

InferenceSchedulerName (p. 93)

The name of the inference scheduler to be deleted.

Type: String

Length Constraints: Minimum length of 1. Maximum length of 200.

Pattern: `^[0-9a-zA-Z_-]{1,200}$`

Required: Yes

Response Elements

If the action is successful, the service sends back an HTTP 200 response with an empty HTTP body.

Errors

AccessDeniedException

The request could not be completed because you do not have access to the resource.

HTTP Status Code: 400

ConflictException

The request could not be completed due to a conflict with the current state of the target resource.

HTTP Status Code: 400

InternalServerException

Processing of the request has failed because of an unknown error, exception or failure.

HTTP Status Code: 500

ResourceNotFoundException

The resource requested could not be found. Verify the resource ID and retry your request.

HTTP Status Code: 400

ThrottlingException

The request was denied due to request throttling.

HTTP Status Code: 400

ValidationException

The input fails to satisfy constraints specified by Amazon Lookout for Equipment or a related AWS service that's being utilized.

HTTP Status Code: 400

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS Command Line Interface
- AWS SDK for .NET
- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Java V2
- AWS SDK for JavaScript
- AWS SDK for PHP V3
- AWS SDK for Python
- AWS SDK for Ruby V3

DeleteModel

Deletes an ML model currently available for Amazon Lookout for Equipment. This will prevent it from being used with an inference scheduler, even one that is already set up.

Request Syntax

```
{
   "ModelName": "string"
}
```

Request Parameters

The request accepts the following data in JSON format.

ModelName (p. 95)

The name of the ML model to be deleted.

Type: String

Length Constraints: Minimum length of 1. Maximum length of 200.

Pattern: `^[0-9a-zA-Z_-]{1,200}$`

Required: Yes

Response Elements

If the action is successful, the service sends back an HTTP 200 response with an empty HTTP body.

Errors

AccessDeniedException

The request could not be completed because you do not have access to the resource.

HTTP Status Code: 400

ConflictException

The request could not be completed due to a conflict with the current state of the target resource.

HTTP Status Code: 400

InternalServerException

Processing of the request has failed because of an unknown error, exception or failure.

HTTP Status Code: 500

ResourceNotFoundException

The resource requested could not be found. Verify the resource ID and retry your request.

HTTP Status Code: 400

ThrottlingException

The request was denied due to request throttling.

HTTP Status Code: 400

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS Command Line Interface
- AWS SDK for .NET
- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Java V2
- AWS SDK for JavaScript
- AWS SDK for PHP V3
- AWS SDK for Python
- AWS SDK for Ruby V3

DescribeDataIngestionJob

Provides information on a specific data ingestion job such as creation time, dataset ARN, status, and so on.

Request Syntax

```
{
   "JobId": "string"
}
```

Request Parameters

The request accepts the following data in JSON format.

JobId (p. 97)

The job ID of the data ingestion job.

Type: String

Length Constraints: Maximum length of 32.

Pattern: `[A-Fa-f0-9]{0,32}`

Required: Yes

Response Syntax

```
{
   "CreatedAt": number,
   "DatasetArn": "string",
   "FailedReason": "string",
   "IngestionInputConfiguration": {
      "S3InputConfiguration": {
         "Bucket": "string",
         "Prefix": "string"
      }
   },
   "JobId": "string",
   "RoleArn": "string",
   "Status": "string"
}
```

Response Elements

If the action is successful, the service sends back an HTTP 200 response.

The following data is returned in JSON format by the service.

CreatedAt (p. 97)

The time at which the data ingestion job was created.

Type: Timestamp

DatasetArn (p. 97)

The Amazon Resource Name (ARN) of the dataset being used in the data ingestion job.

Type: String

Length Constraints: Minimum length of 20. Maximum length of 2048.

Pattern: `arn:aws(-[^:]+)?:lookoutequipment:[a-zA-Z0-9\-]*:[0-9]{12}:dataset\/.+`

FailedReason (p. 97)

Specifies the reason for failure when a data ingestion job has failed.

Type: String

Length Constraints: Minimum length of 1. Maximum length of 5000.

Pattern: `[\P{M}\p{M}]{1,5000}`

IngestionInputConfiguration (p. 97)

Specifies the S3 location configuration for the data input for the data ingestion job.

Type: IngestionInputConfiguration (p. 162) object

JobId (p. 97)

Indicates the job ID of the data ingestion job.

Type: String

Length Constraints: Maximum length of 32.

Pattern: `[A-Fa-f0-9]{0,32}`

RoleArn (p. 97)

The Amazon Resource Name (ARN) of an IAM role with permission to access the data source being ingested.

Type: String

Length Constraints: Minimum length of 20. Maximum length of 2048.

Pattern: `arn:aws(-[^:]+)?:iam::[0-9]{12}:role/.+`

Status (p. 97)

Indicates the status of the `DataIngestionJob` operation.

Type: String

Valid Values: `IN_PROGRESS | SUCCESS | FAILED`

Errors

AccessDeniedException

The request could not be completed because you do not have access to the resource.

HTTP Status Code: 400

InternalServerException

Processing of the request has failed because of an unknown error, exception or failure.

HTTP Status Code: 500

ResourceNotFoundException

The resource requested could not be found. Verify the resource ID and retry your request.

HTTP Status Code: 400

ThrottlingException

The request was denied due to request throttling.

HTTP Status Code: 400

ValidationException

The input fails to satisfy constraints specified by Amazon Lookout for Equipment or a related AWS service that's being utilized.

HTTP Status Code: 400

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS Command Line Interface
- AWS SDK for .NET
- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Java V2
- AWS SDK for JavaScript
- AWS SDK for PHP V3
- AWS SDK for Python
- AWS SDK for Ruby V3

DescribeDataset

Provides a JSON description of the data that is in each time series dataset, including names, column names, and data types.

Request Syntax

```
{
   "DatasetName": "string"
}
```

Request Parameters

The request accepts the following data in JSON format.

DatasetName (p. 100)

The name of the dataset to be described.

Type: String

Length Constraints: Minimum length of 1. Maximum length of 200.

Pattern: `^[0-9a-zA-Z_-]{1,200}$`

Required: Yes

Response Syntax

```
{
   "CreatedAt": number,
   "DatasetArn": "string",
   "DatasetName": "string",
   "IngestionInputConfiguration": {
      "S3InputConfiguration": {
         "Bucket": "string",
         "Prefix": "string"
      }
   },
   "LastUpdatedAt": number,
   "Schema": "string",
   "ServerSideKmsKeyId": "string",
   "Status": "string"
}
```

Response Elements

If the action is successful, the service sends back an HTTP 200 response.

The following data is returned in JSON format by the service.

CreatedAt (p. 100)

Specifies the time the dataset was created in Amazon Lookout for Equipment.

Type: Timestamp

DatasetArn (p. 100)

The Amazon Resource Name (ARN) of the dataset being described.

Type: String

Length Constraints: Minimum length of 20. Maximum length of 2048.

Pattern: `arn:aws(-[^:]+)?:lookoutequipment:[a-zA-Z0-9\-]*:[0-9]{12}:dataset\/.+`

DatasetName (p. 100)

The name of the dataset being described.

Type: String

Length Constraints: Minimum length of 1. Maximum length of 200.

Pattern: `^[0-9a-zA-Z_-]{1,200}$`

IngestionInputConfiguration (p. 100)

Specifies the S3 location configuration for the data input for the data ingestion job.

Type: IngestionInputConfiguration (p. 162) object

LastUpdatedAt (p. 100)

Specifies the time the dataset was last updated, if it was.

Type: Timestamp

Schema (p. 100)

A JSON description of the data that is in each time series dataset, including names, column names, and data types.

Type: String

Length Constraints: Minimum length of 1. Maximum length of 1000000.

ServerSideKmsKeyId (p. 100)

Provides the identifier of the AWS KMS key used to encrypt dataset data by Amazon Lookout for Equipment.

Type: String

Length Constraints: Minimum length of 1. Maximum length of 1024.

Pattern: `arn:aws[a-z\-]*:kms:[a-z0-9\-]*:\d{12}:[\w\-\/]+`

Status (p. 100)

Indicates the status of the dataset.

Type: String

Valid Values: `CREATED | INGESTION_IN_PROGRESS | ACTIVE`

Errors

AccessDeniedException

The request could not be completed because you do not have access to the resource.

HTTP Status Code: 400

InternalServerException

Processing of the request has failed because of an unknown error, exception or failure.

HTTP Status Code: 500

ResourceNotFoundException

The resource requested could not be found. Verify the resource ID and retry your request.

HTTP Status Code: 400

ThrottlingException

The request was denied due to request throttling.

HTTP Status Code: 400

ValidationException

The input fails to satisfy constraints specified by Amazon Lookout for Equipment or a related AWS service that's being utilized.

HTTP Status Code: 400

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS Command Line Interface
- AWS SDK for .NET
- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Java V2
- AWS SDK for JavaScript
- AWS SDK for PHP V3
- AWS SDK for Python
- AWS SDK for Ruby V3

DescribeInferenceScheduler

Specifies information about the inference scheduler being used, including name, model, status, and associated metadata

Request Syntax

```
{
   "InferenceSchedulerName": "string"
}
```

Request Parameters

The request accepts the following data in JSON format.

InferenceSchedulerName (p. 103)

The name of the inference scheduler being described.

Type: String

Length Constraints: Minimum length of 1. Maximum length of 200.

Pattern: `^[0-9a-zA-Z_-]{1,200}$`

Required: Yes

Response Syntax

```
{
   "CreatedAt": number,
   "DataDelayOffsetInMinutes": number,
   "DataInputConfiguration": {
      "InferenceInputNameConfiguration": {
         "ComponentTimestampDelimiter": "string",
         "TimestampFormat": "string"
      },
      "InputTimeZoneOffset": "string",
      "S3InputConfiguration": {
         "Bucket": "string",
         "Prefix": "string"
      }
   },
   "DataOutputConfiguration": {
      "KmsKeyId": "string",
      "S3OutputConfiguration": {
         "Bucket": "string",
         "Prefix": "string"
      }
   },
   "DataUploadFrequency": "string",
   "InferenceSchedulerArn": "string",
   "InferenceSchedulerName": "string",
   "ModelArn": "string",
   "ModelName": "string",
   "RoleArn": "string",
   "ServerSideKmsKeyId": "string",
   "Status": "string",
   "UpdatedAt": number
```

```
}
```

Response Elements

If the action is successful, the service sends back an HTTP 200 response.

The following data is returned in JSON format by the service.

CreatedAt (p. 103)

Specifies the time at which the inference scheduler was created.

Type: Timestamp

DataDelayOffsetInMinutes (p. 103)

A period of time (in minutes) by which inference on the data is delayed after the data starts. For instance, if you select an offset delay time of five minutes, inference will not begin on the data until the first data measurement after the five minute mark. For example, if five minutes is selected, the inference scheduler will wake up at the configured frequency with the additional five minute delay time to check the customer S3 bucket. The customer can upload data at the same frequency and they don't need to stop and restart the scheduler when uploading new data.

Type: Long

Valid Range: Minimum value of 0. Maximum value of 60.

DataInputConfiguration (p. 103)

Specifies configuration information for the input data for the inference scheduler, including delimiter, format, and dataset location.

Type: InferenceInputConfiguration (p. 155) object

DataOutputConfiguration (p. 103)

Specifies information for the output results for the inference scheduler, including the output S3 location.

Type: InferenceOutputConfiguration (p. 157) object

DataUploadFrequency (p. 103)

Specifies how often data is uploaded to the source S3 bucket for the input data. This value is the length of time between data uploads. For instance, if you select 5 minutes, Amazon Lookout for Equipment will upload the real-time data to the source bucket once every 5 minutes. This frequency also determines how often Amazon Lookout for Equipment starts a scheduled inference on your data. In this example, it starts once every 5 minutes.

Type: String

Valid Values: `PT5M | PT10M | PT15M | PT30M | PT1H`

InferenceSchedulerArn (p. 103)

The Amazon Resource Name (ARN) of the inference scheduler being described.

Type: String

Length Constraints: Minimum length of 20. Maximum length of 2048.

Pattern: `arn:aws(-[^:]+)?:lookoutequipment:[a-zA-Z0-9\-]*:[0-9]{12}:inference-scheduler\/.+`

InferenceSchedulerName (p. 103)

The name of the inference scheduler being described.

Type: String

Length Constraints: Minimum length of 1. Maximum length of 200.

Pattern: `^[0-9a-zA-Z_-]{1,200}$`

ModelArn (p. 103)

The Amazon Resource Name (ARN) of the ML model of the inference scheduler being described.

Type: String

Length Constraints: Minimum length of 20. Maximum length of 2048.

Pattern: `arn:aws(-[^:]+)?:lookoutequipment:[a-zA-Z0-9\-]*:[0-9]{12}:model\/.+`

ModelName (p. 103)

The name of the ML model of the inference scheduler being described.

Type: String

Length Constraints: Minimum length of 1. Maximum length of 200.

Pattern: `^[0-9a-zA-Z_-]{1,200}$`

RoleArn (p. 103)

The Amazon Resource Name (ARN) of a role with permission to access the data source for the inference scheduler being described.

Type: String

Length Constraints: Minimum length of 20. Maximum length of 2048.

Pattern: `arn:aws(-[^:]+)?:iam::[0-9]{12}:role/.+`

ServerSideKmsKeyId (p. 103)

Provides the identifier of the AWS KMS key used to encrypt inference scheduler data by Amazon Lookout for Equipment.

Type: String

Length Constraints: Minimum length of 1. Maximum length of 1024.

Pattern: `arn:aws[a-z\-]*:kms:[a-z0-9\-]*:\d{12}:[\w\-\/]+`

Status (p. 103)

Indicates the status of the inference scheduler.

Type: String

Valid Values: `PENDING | RUNNING | STOPPING | STOPPED`

UpdatedAt (p. 103)

Specifies the time at which the inference scheduler was last updated, if it was.

Type: Timestamp

Errors

AccessDeniedException

The request could not be completed because you do not have access to the resource.

HTTP Status Code: 400

InternalServerException

Processing of the request has failed because of an unknown error, exception or failure.

HTTP Status Code: 500

ResourceNotFoundException

The resource requested could not be found. Verify the resource ID and retry your request.

HTTP Status Code: 400

ThrottlingException

The request was denied due to request throttling.

HTTP Status Code: 400

ValidationException

The input fails to satisfy constraints specified by Amazon Lookout for Equipment or a related AWS service that's being utilized.

HTTP Status Code: 400

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS Command Line Interface
- AWS SDK for .NET
- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Java V2
- AWS SDK for JavaScript
- AWS SDK for PHP V3
- AWS SDK for Python
- AWS SDK for Ruby V3

DescribeModel

Provides a JSON containing the overall information about a specific ML model, including model name and ARN, dataset, training and evaluation information, status, and so on.

Request Syntax

```
{
   "ModelName": "string"
}
```

Request Parameters

The request accepts the following data in JSON format.

ModelName (p. 107)

The name of the ML model to be described.

Type: String

Length Constraints: Minimum length of 1. Maximum length of 200.

Pattern: `^[0-9a-zA-Z_-]{1,200}$`

Required: Yes

Response Syntax

```
{
   "CreatedAt": number,
   "DataPreProcessingConfiguration": {
      "TargetSamplingRate": "string"
   },
   "DatasetArn": "string",
   "DatasetName": "string",
   "EvaluationDataEndTime": number,
   "EvaluationDataStartTime": number,
   "FailedReason": "string",
   "LabelsInputConfiguration": {
      "S3InputConfiguration": {
         "Bucket": "string",
         "Prefix": "string"
      }
   },
   "LastUpdatedTime": number,
   "ModelArn": "string",
   "ModelMetrics": "string",
   "ModelName": "string",
   "OffCondition": "string",
   "RoleArn": "string",
   "Schema": "string",
   "ServerSideKmsKeyId": "string",
   "Status": "string",
   "TrainingDataEndTime": number,
   "TrainingDataStartTime": number,
   "TrainingExecutionEndTime": number,
   "TrainingExecutionStartTime": number
```

```
}
```

Response Elements

If the action is successful, the service sends back an HTTP 200 response.

The following data is returned in JSON format by the service.

CreatedAt (p. 107)

Indicates the time and date at which the ML model was created.

Type: Timestamp

DataPreProcessingConfiguration (p. 107)

The configuration is the `TargetSamplingRate`, which is the sampling rate of the data after post processing by Amazon Lookout for Equipment. For example, if you provide data that has been collected at a 1 second level and you want the system to resample the data at a 1 minute rate before training, the `TargetSamplingRate` is 1 minute.

When providing a value for the `TargetSamplingRate`, you must attach the prefix "PT" to the rate you want. The value for a 1 second rate is therefore *PT1S*, the value for a 15 minute rate is *PT15M*, and the value for a 1 hour rate is *PT1H*

Type: DataPreProcessingConfiguration (p. 149) object

DatasetArn (p. 107)

The Amazon Resouce Name (ARN) of the dataset used to create the ML model being described.

Type: String

Length Constraints: Minimum length of 20. Maximum length of 2048.

Pattern: `arn:aws(-[^:]+)?:lookoutequipment:[a-zA-Z0-9\-]*:[0-9]{12}:dataset\/.+`

DatasetName (p. 107)

The name of the dataset being used by the ML being described.

Type: String

Length Constraints: Minimum length of 1. Maximum length of 200.

Pattern: `^[0-9a-zA-Z_-]{1,200}$`

EvaluationDataEndTime (p. 107)

Indicates the time reference in the dataset that was used to end the subset of evaluation data for the ML model.

Type: Timestamp

EvaluationDataStartTime (p. 107)

Indicates the time reference in the dataset that was used to begin the subset of evaluation data for the ML model.

Type: Timestamp

FailedReason (p. 107)

If the training of the ML model failed, this indicates the reason for that failure.

Type: String

Length Constraints: Minimum length of 1. Maximum length of 5000.

Pattern: `[\P{M}\p{M}]{1,5000}`

LabelsInputConfiguration (p. 107)

Specifies configuration information about the labels input, including its S3 location.

Type: LabelsInputConfiguration (p. 164) object

LastUpdatedTime (p. 107)

Indicates the last time the ML model was updated. The type of update is not specified.

Type: Timestamp

ModelArn (p. 107)

The Amazon Resource Name (ARN) of the ML model being described.

Type: String

Length Constraints: Minimum length of 20. Maximum length of 2048.

Pattern: `arn:aws(-[^:]+)?:lookoutequipment:[a-zA-Z0-9\-]*:[0-9]{12}:model\/.+`

ModelMetrics (p. 107)

The Model Metrics show an aggregated summary of the model's performance within the evaluation time range. This is the JSON content of the metrics created when evaluating the model.

Type: String

Length Constraints: Minimum length of 1. Maximum length of 50000.

ModelName (p. 107)

The name of the ML model being described.

Type: String

Length Constraints: Minimum length of 1. Maximum length of 200.

Pattern: `^[0-9a-zA-Z_-]{1,200}$`

OffCondition (p. 107)

Indicates that the asset associated with this sensor has been shut off. As long as this condition is met, Lookout for Equipment will not use data from this asset for training, evaluation, or inference.

Type: String

Length Constraints: Minimum length of 1. Maximum length of 2048.

RoleArn (p. 107)

The Amazon Resource Name (ARN) of a role with permission to access the data source for the ML model being described.

Type: String

Length Constraints: Minimum length of 20. Maximum length of 2048.

Pattern: `arn:aws(-[^:]+)?:iam::[0-9]{12}:role/.+`

Schema (p. 107)

A JSON description of the data that is in each time series dataset, including names, column names, and data types.

Type: String

Length Constraints: Minimum length of 1. Maximum length of 1000000.

ServerSideKmsKeyId (p. 107)

Provides the identifier of the AWS KMS key used to encrypt model data by Amazon Lookout for Equipment.

Type: String

Length Constraints: Minimum length of 1. Maximum length of 1024.

Pattern: `arn:aws[a-z\-]*:kms:[a-z0-9\-]*:\d{12}:[\w\-\/]+`

Status (p. 107)

Specifies the current status of the model being described. Status describes the status of the most recent action of the model.

Type: String

Valid Values: `IN_PROGRESS | SUCCESS | FAILED`

TrainingDataEndTime (p. 107)

Indicates the time reference in the dataset that was used to end the subset of training data for the ML model.

Type: Timestamp

TrainingDataStartTime (p. 107)

Indicates the time reference in the dataset that was used to begin the subset of training data for the ML model.

Type: Timestamp

TrainingExecutionEndTime (p. 107)

Indicates the time at which the training of the ML model was completed.

Type: Timestamp

TrainingExecutionStartTime (p. 107)

Indicates the time at which the training of the ML model began.

Type: Timestamp

Errors

AccessDeniedException

The request could not be completed because you do not have access to the resource.

HTTP Status Code: 400

InternalServerException

Processing of the request has failed because of an unknown error, exception or failure.

HTTP Status Code: 500

ResourceNotFoundException

The resource requested could not be found. Verify the resource ID and retry your request.

HTTP Status Code: 400

ThrottlingException

The request was denied due to request throttling.

HTTP Status Code: 400

ValidationException

The input fails to satisfy constraints specified by Amazon Lookout for Equipment or a related AWS service that's being utilized.

HTTP Status Code: 400

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS Command Line Interface
- AWS SDK for .NET
- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Java V2
- AWS SDK for JavaScript
- AWS SDK for PHP V3
- AWS SDK for Python
- AWS SDK for Ruby V3

ListDataIngestionJobs

Provides a list of all data ingestion jobs, including dataset name and ARN, S3 location of the input data, status, and so on.

Request Syntax

```
{
   "DatasetName": "string",
   "MaxResults": number,
   "NextToken": "string",
   "Status": "string"
}
```

Request Parameters

The request accepts the following data in JSON format.

DatasetName (p. 112)

The name of the dataset being used for the data ingestion job.

Type: String

Length Constraints: Minimum length of 1. Maximum length of 200.

Pattern: `^[0-9a-zA-Z_-]{1,200}$`

Required: No

MaxResults (p. 112)

Specifies the maximum number of data ingestion jobs to list.

Type: Integer

Valid Range: Minimum value of 1. Maximum value of 500.

Required: No

NextToken (p. 112)

An opaque pagination token indicating where to continue the listing of data ingestion jobs.

Type: String

Length Constraints: Maximum length of 8192.

Pattern: `\p{ASCII}{0,8192}`

Required: No

Status (p. 112)

Indicates the status of the data ingestion job.

Type: String

Valid Values: `IN_PROGRESS | SUCCESS | FAILED`

Required: No

Response Syntax

```
{
   "DataIngestionJobSummaries": [
      {
         "DatasetArn": "string",
         "DatasetName": "string",
         "IngestionInputConfiguration": {
            "S3InputConfiguration": {
               "Bucket": "string",
               "Prefix": "string"
            }
         },
         "JobId": "string",
         "Status": "string"
      }
   ],
   "NextToken": "string"
}
```

Response Elements

If the action is successful, the service sends back an HTTP 200 response.

The following data is returned in JSON format by the service.

DataIngestionJobSummaries (p. 113)

Specifies information about the specific data ingestion job, including dataset name and status.

Type: Array of DataIngestionJobSummary (p. 147) objects

NextToken (p. 113)

An opaque pagination token indicating where to continue the listing of data ingestion jobs.

Type: String

Length Constraints: Maximum length of 8192.

Pattern: `\p{ASCII}{0,8192}`

Errors

AccessDeniedException

The request could not be completed because you do not have access to the resource.

HTTP Status Code: 400

InternalServerException

Processing of the request has failed because of an unknown error, exception or failure.

HTTP Status Code: 500

ThrottlingException

The request was denied due to request throttling.

HTTP Status Code: 400

ValidationException

The input fails to satisfy constraints specified by Amazon Lookout for Equipment or a related AWS service that's being utilized.

HTTP Status Code: 400

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS Command Line Interface
- AWS SDK for .NET
- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Java V2
- AWS SDK for JavaScript
- AWS SDK for PHP V3
- AWS SDK for Python
- AWS SDK for Ruby V3

ListDatasets

Lists all datasets currently available in your account, filtering on the dataset name.

Request Syntax

```
{
   "DatasetNameBeginsWith": "string",
   "MaxResults": number,
   "NextToken": "string"
}
```

Request Parameters

The request accepts the following data in JSON format.

DatasetNameBeginsWith (p. 115)

The beginning of the name of the datasets to be listed.

Type: String

Length Constraints: Minimum length of 1. Maximum length of 200.

Pattern: `^[0-9a-zA-Z_-]{1,200}$`

Required: No

MaxResults (p. 115)

Specifies the maximum number of datasets to list.

Type: Integer

Valid Range: Minimum value of 1. Maximum value of 500.

Required: No

NextToken (p. 115)

An opaque pagination token indicating where to continue the listing of datasets.

Type: String

Length Constraints: Maximum length of 8192.

Pattern: `\p{ASCII}{0,8192}`

Required: No

Response Syntax

```
{
   "DatasetSummaries": [
      {
         "CreatedAt": number,
         "DatasetArn": "string",
         "DatasetName": "string",
         "Status": "string"
```

```
        }
    ],
    "NextToken": "string"
}
```

Response Elements

If the action is successful, the service sends back an HTTP 200 response.

The following data is returned in JSON format by the service.

DatasetSummaries (p. 115)

Provides information about the specified dataset, including creation time, dataset ARN, and status.

Type: Array of DatasetSummary (p. 151) objects

NextToken (p. 115)

An opaque pagination token indicating where to continue the listing of datasets.

Type: String

Length Constraints: Maximum length of 8192.

Pattern: `\p{ASCII}{0,8192}`

Errors

AccessDeniedException

The request could not be completed because you do not have access to the resource.

HTTP Status Code: 400

InternalServerException

Processing of the request has failed because of an unknown error, exception or failure.

HTTP Status Code: 500

ThrottlingException

The request was denied due to request throttling.

HTTP Status Code: 400

ValidationException

The input fails to satisfy constraints specified by Amazon Lookout for Equipment or a related AWS service that's being utilized.

HTTP Status Code: 400

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS Command Line Interface
- AWS SDK for .NET

- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Java V2
- AWS SDK for JavaScript
- AWS SDK for PHP V3
- AWS SDK for Python
- AWS SDK for Ruby V3

ListInferenceExecutions

Lists all inference executions that have been performed by the specified inference scheduler.

Request Syntax

```
{
   "DataEndTimeBefore": number,
   "DataStartTimeAfter": number,
   "InferenceSchedulerName": "string",
   "MaxResults": number,
   "NextToken": "string",
   "Status": "string"
}
```

Request Parameters

The request accepts the following data in JSON format.

DataEndTimeBefore (p. 118)

The time reference in the inferenced dataset before which Amazon Lookout for Equipment stopped the inference execution.

Type: Timestamp

Required: No

DataStartTimeAfter (p. 118)

The time reference in the inferenced dataset after which Amazon Lookout for Equipment started the inference execution.

Type: Timestamp

Required: No

InferenceSchedulerName (p. 118)

The name of the inference scheduler for the inference execution listed.

Type: String

Length Constraints: Minimum length of 1. Maximum length of 200.

Pattern: `^[0-9a-zA-Z_-]{1,200}$`

Required: Yes

MaxResults (p. 118)

Specifies the maximum number of inference executions to list.

Type: Integer

Valid Range: Minimum value of 1. Maximum value of 500.

Required: No

NextToken (p. 118)

An opaque pagination token indicating where to continue the listing of inference executions.

Type: String

Length Constraints: Maximum length of 8192.

Pattern: `\p{ASCII}{0,8192}`

Required: No

Status (p. 118)

The status of the inference execution.

Type: String

Valid Values: `IN_PROGRESS | SUCCESS | FAILED`

Required: No

Response Syntax

```
{
   "InferenceExecutionSummaries": [
      {
         "CustomerResultObject": {
            "Bucket": "string",
            "Key": "string"
         },
         "DataEndTime": number,
         "DataInputConfiguration": {
            "InferenceInputNameConfiguration": {
               "ComponentTimestampDelimiter": "string",
               "TimestampFormat": "string"
            },
            "InputTimeZoneOffset": "string",
            "S3InputConfiguration": {
               "Bucket": "string",
               "Prefix": "string"
            }
         },
         "DataOutputConfiguration": {
            "KmsKeyId": "string",
            "S3OutputConfiguration": {
               "Bucket": "string",
               "Prefix": "string"
            }
         },
         "DataStartTime": number,
         "FailedReason": "string",
         "InferenceSchedulerArn": "string",
         "InferenceSchedulerName": "string",
         "ModelArn": "string",
         "ModelName": "string",
         "ScheduledStartTime": number,
         "Status": "string"
      }
   ],
   "NextToken": "string"
}
```

Response Elements

If the action is successful, the service sends back an HTTP 200 response.

The following data is returned in JSON format by the service.

InferenceExecutionSummaries (p. 119)

Provides an array of information about the individual inference executions returned from the `ListInferenceExecutions` operation, including model used, inference scheduler, data configuration, and so on.

Type: Array of InferenceExecutionSummary (p. 152) objects

NextToken (p. 119)

An opaque pagination token indicating where to continue the listing of inference executions.

Type: String

Length Constraints: Maximum length of 8192.

Pattern: `\p{ASCII}{0,8192}`

Errors

AccessDeniedException

The request could not be completed because you do not have access to the resource.

HTTP Status Code: 400

InternalServerException

Processing of the request has failed because of an unknown error, exception or failure.

HTTP Status Code: 500

ResourceNotFoundException

The resource requested could not be found. Verify the resource ID and retry your request.

HTTP Status Code: 400

ThrottlingException

The request was denied due to request throttling.

HTTP Status Code: 400

ValidationException

The input fails to satisfy constraints specified by Amazon Lookout for Equipment or a related AWS service that's being utilized.

HTTP Status Code: 400

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS Command Line Interface
- AWS SDK for .NET
- AWS SDK for C++
- AWS SDK for Go

- AWS SDK for Java V2
- AWS SDK for JavaScript
- AWS SDK for PHP V3
- AWS SDK for Python
- AWS SDK for Ruby V3

ListInferenceSchedulers

Retrieves a list of all inference schedulers currently available for your account.

Request Syntax

```
{
   "InferenceSchedulerNameBeginsWith": "string",
   "MaxResults": number,
   "ModelName": "string",
   "NextToken": "string"
}
```

Request Parameters

The request accepts the following data in JSON format.

InferenceSchedulerNameBeginsWith (p. 122)

The beginning of the name of the inference schedulers to be listed.

Type: String

Length Constraints: Minimum length of 1. Maximum length of 200.

Pattern: `^[0-9a-zA-Z_-]{1,200}$`

Required: No

MaxResults (p. 122)

Specifies the maximum number of inference schedulers to list.

Type: Integer

Valid Range: Minimum value of 1. Maximum value of 500.

Required: No

ModelName (p. 122)

The name of the ML model used by the inference scheduler to be listed.

Type: String

Length Constraints: Minimum length of 1. Maximum length of 200.

Pattern: `^[0-9a-zA-Z_-]{1,200}$`

Required: No

NextToken (p. 122)

An opaque pagination token indicating where to continue the listing of inference schedulers.

Type: String

Length Constraints: Maximum length of 8192.

Pattern: `\p{ASCII}{0,8192}`

Required: No

Response Syntax

```
{
   "InferenceSchedulerSummaries": [
      {
         "DataDelayOffsetInMinutes": number,
         "DataUploadFrequency": "string",
         "InferenceSchedulerArn": "string",
         "InferenceSchedulerName": "string",
         "ModelArn": "string",
         "ModelName": "string",
         "Status": "string"
      }
   ],
   "NextToken": "string"
}
```

Response Elements

If the action is successful, the service sends back an HTTP 200 response.

The following data is returned in JSON format by the service.

InferenceSchedulerSummaries (p. 123)

Provides information about the specified inference scheduler, including data upload frequency, model name and ARN, and status.

Type: Array of InferenceSchedulerSummary (p. 160) objects

NextToken (p. 123)

An opaque pagination token indicating where to continue the listing of inference schedulers.

Type: String

Length Constraints: Maximum length of 8192.

Pattern: `\p{ASCII}{0,8192}`

Errors

AccessDeniedException

The request could not be completed because you do not have access to the resource.

HTTP Status Code: 400

InternalServerException

Processing of the request has failed because of an unknown error, exception or failure.

HTTP Status Code: 500

ThrottlingException

The request was denied due to request throttling.

HTTP Status Code: 400

ValidationException

The input fails to satisfy constraints specified by Amazon Lookout for Equipment or a related AWS service that's being utilized.

HTTP Status Code: 400

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS Command Line Interface
- AWS SDK for .NET
- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Java V2
- AWS SDK for JavaScript
- AWS SDK for PHP V3
- AWS SDK for Python
- AWS SDK for Ruby V3

ListModels

Generates a list of all models in the account, including model name and ARN, dataset, and status.

Request Syntax

```
{
   "DatasetNameBeginsWith": "string",
   "MaxResults": number,
   "ModelNameBeginsWith": "string",
   "NextToken": "string",
   "Status": "string"
}
```

Request Parameters

The request accepts the following data in JSON format.

DatasetNameBeginsWith (p. 125)

The beginning of the name of the dataset of the ML models to be listed.

Type: String

Length Constraints: Minimum length of 1. Maximum length of 200.

Pattern: `^[0-9a-zA-Z_-]{1,200}$`

Required: No

MaxResults (p. 125)

Specifies the maximum number of ML models to list.

Type: Integer

Valid Range: Minimum value of 1. Maximum value of 500.

Required: No

ModelNameBeginsWith (p. 125)

The beginning of the name of the ML models being listed.

Type: String

Length Constraints: Minimum length of 1. Maximum length of 200.

Pattern: `^[0-9a-zA-Z_-]{1,200}$`

Required: No

NextToken (p. 125)

An opaque pagination token indicating where to continue the listing of ML models.

Type: String

Length Constraints: Maximum length of 8192.

Pattern: `\p{ASCII}{0,8192}`

Required: No

Status (p. 125)

The status of the ML model.

Type: String

Valid Values: `IN_PROGRESS | SUCCESS | FAILED`

Required: No

Response Syntax

```
{
   "ModelSummaries": [
      {
         "CreatedAt": number,
         "DatasetArn": "string",
         "DatasetName": "string",
         "ModelArn": "string",
         "ModelName": "string",
         "Status": "string"
      }
   ],
   "NextToken": "string"
}
```

Response Elements

If the action is successful, the service sends back an HTTP 200 response.

The following data is returned in JSON format by the service.

ModelSummaries (p. 126)

Provides information on the specified model, including created time, model and dataset ARNs, and status.

Type: Array of ModelSummary (p. 166) objects

NextToken (p. 126)

An opaque pagination token indicating where to continue the listing of ML models.

Type: String

Length Constraints: Maximum length of 8192.

Pattern: `\p{ASCII}{0,8192}`

Errors

AccessDeniedException

The request could not be completed because you do not have access to the resource.

HTTP Status Code: 400

InternalServerException

Processing of the request has failed because of an unknown error, exception or failure.

HTTP Status Code: 500

ThrottlingException

The request was denied due to request throttling.

HTTP Status Code: 400

ValidationException

The input fails to satisfy constraints specified by Amazon Lookout for Equipment or a related AWS service that's being utilized.

HTTP Status Code: 400

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS Command Line Interface
- AWS SDK for .NET
- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Java V2
- AWS SDK for JavaScript
- AWS SDK for PHP V3
- AWS SDK for Python
- AWS SDK for Ruby V3

ListTagsForResource

Lists all the tags for a specified resource, including key and value.

Request Syntax

```
{
   "ResourceArn": "string"
}
```

Request Parameters

The request accepts the following data in JSON format.

ResourceArn (p. 128)

The Amazon Resource Name (ARN) of the resource (such as the dataset or model) that is the focus of the `ListTagsForResource` operation.

Type: String

Length Constraints: Minimum length of 1. Maximum length of 1011.

Required: Yes

Response Syntax

```
{
   "Tags": [
      {
         "Key": "string",
         "Value": "string"
      }
   ]
}
```

Response Elements

If the action is successful, the service sends back an HTTP 200 response.

The following data is returned in JSON format by the service.

Tags (p. 128)

Any tags associated with the resource.

Type: Array of Tag (p. 169) objects

Array Members: Minimum number of 0 items. Maximum number of 200 items.

Errors

AccessDeniedException

The request could not be completed because you do not have access to the resource.

HTTP Status Code: 400

InternalServerException

Processing of the request has failed because of an unknown error, exception or failure.

HTTP Status Code: 500

ResourceNotFoundException

The resource requested could not be found. Verify the resource ID and retry your request.

HTTP Status Code: 400

ThrottlingException

The request was denied due to request throttling.

HTTP Status Code: 400

ValidationException

The input fails to satisfy constraints specified by Amazon Lookout for Equipment or a related AWS service that's being utilized.

HTTP Status Code: 400

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS Command Line Interface
- AWS SDK for .NET
- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Java V2
- AWS SDK for JavaScript
- AWS SDK for PHP V3
- AWS SDK for Python
- AWS SDK for Ruby V3

StartDataIngestionJob

Starts a data ingestion job. Amazon Lookout for Equipment returns the job status.

Request Syntax

```
{
   "ClientToken": "string",
   "DatasetName": "string",
   "IngestionInputConfiguration": {
      "S3InputConfiguration": {
         "Bucket": "string",
         "Prefix": "string"
      }
   },
   "RoleArn": "string"
}
```

Request Parameters

The request accepts the following data in JSON format.

ClientToken (p. 130)

A unique identifier for the request. If you do not set the client request token, Amazon Lookout for Equipment generates one.

Type: String

Length Constraints: Minimum length of 1. Maximum length of 256.

Pattern: `\p{ASCII}{1,256}`

Required: Yes

DatasetName (p. 130)

The name of the dataset being used by the data ingestion job.

Type: String

Length Constraints: Minimum length of 1. Maximum length of 200.

Pattern: `^[0-9a-zA-Z_-]{1,200}$`

Required: Yes

IngestionInputConfiguration (p. 130)

Specifies information for the input data for the data ingestion job, including dataset S3 location.

Type: IngestionInputConfiguration (p. 162) object

Required: Yes

RoleArn (p. 130)

The Amazon Resource Name (ARN) of a role with permission to access the data source for the data ingestion job.

Type: String

Length Constraints: Minimum length of 20. Maximum length of 2048.

Pattern: `arn:aws(-[^:]+)?:iam::[0-9]{12}:role/.+`

Required: Yes

Response Syntax

```
{
   "JobId": "string",
   "Status": "string"
}
```

Response Elements

If the action is successful, the service sends back an HTTP 200 response.

The following data is returned in JSON format by the service.

JobId (p. 131)

Indicates the job ID of the data ingestion job.

Type: String

Length Constraints: Maximum length of 32.

Pattern: `[A-Fa-f0-9]{0,32}`

Status (p. 131)

Indicates the status of the `StartDataIngestionJob` operation.

Type: String

Valid Values: `IN_PROGRESS | SUCCESS | FAILED`

Errors

AccessDeniedException

The request could not be completed because you do not have access to the resource.

HTTP Status Code: 400

ConflictException

The request could not be completed due to a conflict with the current state of the target resource.

HTTP Status Code: 400

InternalServerException

Processing of the request has failed because of an unknown error, exception or failure.

HTTP Status Code: 500

ResourceNotFoundException

The resource requested could not be found. Verify the resource ID and retry your request.

HTTP Status Code: 400

ServiceQuotaExceededException

Resource limitations have been exceeded.

HTTP Status Code: 400

ThrottlingException

The request was denied due to request throttling.

HTTP Status Code: 400

ValidationException

The input fails to satisfy constraints specified by Amazon Lookout for Equipment or a related AWS service that's being utilized.

HTTP Status Code: 400

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS Command Line Interface
- AWS SDK for .NET
- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Java V2
- AWS SDK for JavaScript
- AWS SDK for PHP V3
- AWS SDK for Python
- AWS SDK for Ruby V3

StartInferenceScheduler

Starts an inference scheduler.

Request Syntax

```
{
   "InferenceSchedulerName": "string"
}
```

Request Parameters

The request accepts the following data in JSON format.

InferenceSchedulerName (p. 133)

The name of the inference scheduler to be started.

Type: String

Length Constraints: Minimum length of 1. Maximum length of 200.

Pattern: `^[0-9a-zA-Z_-]{1,200}$`

Required: Yes

Response Syntax

```
{
   "InferenceSchedulerArn": "string",
   "InferenceSchedulerName": "string",
   "ModelArn": "string",
   "ModelName": "string",
   "Status": "string"
}
```

Response Elements

If the action is successful, the service sends back an HTTP 200 response.

The following data is returned in JSON format by the service.

InferenceSchedulerArn (p. 133)

The Amazon Resource Name (ARN) of the inference scheduler being started.

Type: String

Length Constraints: Minimum length of 20. Maximum length of 2048.

Pattern: `arn:aws(-[^:]+)?:lookoutequipment:[a-zA-Z0-9\-]*:[0-9]{12}:inference-scheduler\/.+`

InferenceSchedulerName (p. 133)

The name of the inference scheduler being started.

Type: String

Length Constraints: Minimum length of 1. Maximum length of 200.

Pattern: `^[0-9a-zA-Z_-]{1,200}$`

ModelArn (p. 133)

The Amazon Resource Name (ARN) of the ML model being used by the inference scheduler.

Type: String

Length Constraints: Minimum length of 20. Maximum length of 2048.

Pattern: `arn:aws(-[^:]+)?:lookoutequipment:[a-zA-Z0-9\-]*:[0-9]{12}:model\/.+`

ModelName (p. 133)

The name of the ML model being used by the inference scheduler.

Type: String

Length Constraints: Minimum length of 1. Maximum length of 200.

Pattern: `^[0-9a-zA-Z_-]{1,200}$`

Status (p. 133)

Indicates the status of the inference scheduler.

Type: String

Valid Values: `PENDING | RUNNING | STOPPING | STOPPED`

Errors

AccessDeniedException

The request could not be completed because you do not have access to the resource.

HTTP Status Code: 400

ConflictException

The request could not be completed due to a conflict with the current state of the target resource.

HTTP Status Code: 400

InternalServerException

Processing of the request has failed because of an unknown error, exception or failure.

HTTP Status Code: 500

ResourceNotFoundException

The resource requested could not be found. Verify the resource ID and retry your request.

HTTP Status Code: 400

ThrottlingException

The request was denied due to request throttling.

HTTP Status Code: 400

ValidationException

The input fails to satisfy constraints specified by Amazon Lookout for Equipment or a related AWS service that's being utilized.

HTTP Status Code: 400

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS Command Line Interface
- AWS SDK for .NET
- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Java V2
- AWS SDK for JavaScript
- AWS SDK for PHP V3
- AWS SDK for Python
- AWS SDK for Ruby V3

StopInferenceScheduler

Stops an inference scheduler.

Request Syntax

```
{
   "InferenceSchedulerName": "string"
}
```

Request Parameters

The request accepts the following data in JSON format.

InferenceSchedulerName (p. 136)

The name of the inference scheduler to be stopped.

Type: String

Length Constraints: Minimum length of 1. Maximum length of 200.

Pattern: `^[0-9a-zA-Z_-]{1,200}$`

Required: Yes

Response Syntax

```
{
   "InferenceSchedulerArn": "string",
   "InferenceSchedulerName": "string",
   "ModelArn": "string",
   "ModelName": "string",
   "Status": "string"
}
```

Response Elements

If the action is successful, the service sends back an HTTP 200 response.

The following data is returned in JSON format by the service.

InferenceSchedulerArn (p. 136)

The Amazon Resource Name (ARN) of the inference schedule being stopped.

Type: String

Length Constraints: Minimum length of 20. Maximum length of 2048.

Pattern: `arn:aws(-[^:]+)?:lookoutequipment:[a-zA-Z0-9\-]*:[0-9]{12}:inference-scheduler\/.+`

InferenceSchedulerName (p. 136)

The name of the inference scheduler being stopped.

Type: String

Length Constraints: Minimum length of 1. Maximum length of 200.

Pattern: `^[0-9a-zA-Z_-]{1,200}$`

ModelArn (p. 136)

The Amazon Resource Name (ARN) of the ML model used by the inference scheduler being stopped.

Type: String

Length Constraints: Minimum length of 20. Maximum length of 2048.

Pattern: `arn:aws(-[^:]+)?:lookoutequipment:[a-zA-Z0-9\-]*:[0-9]{12}:model\/.+`

ModelName (p. 136)

The name of the ML model used by the inference scheduler being stopped.

Type: String

Length Constraints: Minimum length of 1. Maximum length of 200.

Pattern: `^[0-9a-zA-Z_-]{1,200}$`

Status (p. 136)

Indicates the status of the inference scheduler.

Type: String

Valid Values: `PENDING | RUNNING | STOPPING | STOPPED`

Errors

AccessDeniedException

The request could not be completed because you do not have access to the resource.

HTTP Status Code: 400

ConflictException

The request could not be completed due to a conflict with the current state of the target resource.

HTTP Status Code: 400

InternalServerException

Processing of the request has failed because of an unknown error, exception or failure.

HTTP Status Code: 500

ResourceNotFoundException

The resource requested could not be found. Verify the resource ID and retry your request.

HTTP Status Code: 400

ThrottlingException

The request was denied due to request throttling.

HTTP Status Code: 400

ValidationException

The input fails to satisfy constraints specified by Amazon Lookout for Equipment or a related AWS service that's being utilized.

HTTP Status Code: 400

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS Command Line Interface
- AWS SDK for .NET
- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Java V2
- AWS SDK for JavaScript
- AWS SDK for PHP V3
- AWS SDK for Python
- AWS SDK for Ruby V3

TagResource

Associates a given tag to a resource in your account. A tag is a key-value pair which can be added to an Amazon Lookout for Equipment resource as metadata. Tags can be used for organizing your resources as well as helping you to search and filter by tag. Multiple tags can be added to a resource, either when you create it, or later. Up to 50 tags can be associated with each resource.

Request Syntax

```
{
   "ResourceArn": "string",
   "Tags": [
      {
         "Key": "string",
         "Value": "string"
      }
   ]
}
```

Request Parameters

The request accepts the following data in JSON format.

ResourceArn (p. 139)

The Amazon Resource Name (ARN) of the specific resource to which the tag should be associated.

Type: String

Length Constraints: Minimum length of 1. Maximum length of 1011.

Required: Yes

Tags (p. 139)

The tag or tags to be associated with a specific resource. Both the tag key and value are specified.

Type: Array of Tag (p. 169) objects

Array Members: Minimum number of 0 items. Maximum number of 200 items.

Required: Yes

Response Elements

If the action is successful, the service sends back an HTTP 200 response with an empty HTTP body.

Errors

AccessDeniedException

The request could not be completed because you do not have access to the resource.

HTTP Status Code: 400

InternalServerException

Processing of the request has failed because of an unknown error, exception or failure.

HTTP Status Code: 500

ResourceNotFoundException

The resource requested could not be found. Verify the resource ID and retry your request.

HTTP Status Code: 400

ServiceQuotaExceededException

Resource limitations have been exceeded.

HTTP Status Code: 400

ThrottlingException

The request was denied due to request throttling.

HTTP Status Code: 400

ValidationException

The input fails to satisfy constraints specified by Amazon Lookout for Equipment or a related AWS service that's being utilized.

HTTP Status Code: 400

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS Command Line Interface
- AWS SDK for .NET
- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Java V2
- AWS SDK for JavaScript
- AWS SDK for PHP V3
- AWS SDK for Python
- AWS SDK for Ruby V3

UntagResource

Removes a specific tag from a given resource. The tag is specified by its key.

Request Syntax

```
{
   "ResourceArn": "string",
   "TagKeys": [ "string" ]
}
```

Request Parameters

The request accepts the following data in JSON format.

ResourceArn (p. 141)

The Amazon Resource Name (ARN) of the resource to which the tag is currently associated.

Type: String

Length Constraints: Minimum length of 1. Maximum length of 1011.

Required: Yes

TagKeys (p. 141)

Specifies the key of the tag to be removed from a specified resource.

Type: Array of strings

Array Members: Minimum number of 0 items. Maximum number of 200 items.

Length Constraints: Minimum length of 1. Maximum length of 128.

Pattern: `^(?!aws:)[a-zA-Z+-=._:/]+$`

Required: Yes

Response Elements

If the action is successful, the service sends back an HTTP 200 response with an empty HTTP body.

Errors

AccessDeniedException

The request could not be completed because you do not have access to the resource.

HTTP Status Code: 400

InternalServerException

Processing of the request has failed because of an unknown error, exception or failure.

HTTP Status Code: 500

ResourceNotFoundException

The resource requested could not be found. Verify the resource ID and retry your request.

HTTP Status Code: 400

ThrottlingException

The request was denied due to request throttling.

HTTP Status Code: 400

ValidationException

The input fails to satisfy constraints specified by Amazon Lookout for Equipment or a related AWS service that's being utilized.

HTTP Status Code: 400

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS Command Line Interface
- AWS SDK for .NET
- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Java V2
- AWS SDK for JavaScript
- AWS SDK for PHP V3
- AWS SDK for Python
- AWS SDK for Ruby V3

UpdateInferenceScheduler

Updates an inference scheduler.

Request Syntax

```
{
   "DataDelayOffsetInMinutes": number,
   "DataInputConfiguration": {
      "InferenceInputNameConfiguration": {
         "ComponentTimestampDelimiter": "string",
         "TimestampFormat": "string"
      },
      "InputTimeZoneOffset": "string",
      "S3InputConfiguration": {
         "Bucket": "string",
         "Prefix": "string"
      }
   },
   "DataOutputConfiguration": {
      "KmsKeyId": "string",
      "S3OutputConfiguration": {
         "Bucket": "string",
         "Prefix": "string"
      }
   },
   "DataUploadFrequency": "string",
   "InferenceSchedulerName": "string",
   "RoleArn": "string"
}
```

Request Parameters

The request accepts the following data in JSON format.

DataDelayOffsetInMinutes (p. 143)

A period of time (in minutes) by which inference on the data is delayed after the data starts. For instance, if you select an offset delay time of five minutes, inference will not begin on the data until the first data measurement after the five minute mark. For example, if five minutes is selected, the inference scheduler will wake up at the configured frequency with the additional five minute delay time to check the customer S3 bucket. The customer can upload data at the same frequency and they don't need to stop and restart the scheduler when uploading new data.

Type: Long

Valid Range: Minimum value of 0. Maximum value of 60.

Required: No

DataInputConfiguration (p. 143)

Specifies information for the input data for the inference scheduler, including delimiter, format, and dataset location.

Type: InferenceInputConfiguration (p. 155) object

Required: No

DataOutputConfiguration (p. 143)

Specifies information for the output results from the inference scheduler, including the output S3 location.

Type: InferenceOutputConfiguration (p. 157) object

Required: No

DataUploadFrequency (p. 143)

How often data is uploaded to the source S3 bucket for the input data. The value chosen is the length of time between data uploads. For instance, if you select 5 minutes, Amazon Lookout for Equipment will upload the real-time data to the source bucket once every 5 minutes. This frequency also determines how often Amazon Lookout for Equipment starts a scheduled inference on your data. In this example, it starts once every 5 minutes.

Type: String

Valid Values: `PT5M | PT10M | PT15M | PT30M | PT1H`

Required: No

InferenceSchedulerName (p. 143)

The name of the inference scheduler to be updated.

Type: String

Length Constraints: Minimum length of 1. Maximum length of 200.

Pattern: `^[0-9a-zA-Z_-]{1,200}$`

Required: Yes

RoleArn (p. 143)

The Amazon Resource Name (ARN) of a role with permission to access the data source for the inference scheduler.

Type: String

Length Constraints: Minimum length of 20. Maximum length of 2048.

Pattern: `arn:aws(-[^:]+)?:iam::[0-9]{12}:role/.+`

Required: No

Response Elements

If the action is successful, the service sends back an HTTP 200 response with an empty HTTP body.

Errors

AccessDeniedException

The request could not be completed because you do not have access to the resource.

HTTP Status Code: 400

ConflictException

The request could not be completed due to a conflict with the current state of the target resource.

HTTP Status Code: 400

InternalServerException

Processing of the request has failed because of an unknown error, exception or failure.

HTTP Status Code: 500

ResourceNotFoundException

The resource requested could not be found. Verify the resource ID and retry your request.

HTTP Status Code: 400

ThrottlingException

The request was denied due to request throttling.

HTTP Status Code: 400

ValidationException

The input fails to satisfy constraints specified by Amazon Lookout for Equipment or a related AWS service that's being utilized.

HTTP Status Code: 400

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS Command Line Interface
- AWS SDK for .NET
- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Java V2
- AWS SDK for JavaScript
- AWS SDK for PHP V3
- AWS SDK for Python
- AWS SDK for Ruby V3

Data Types

The following data types are supported:

DataIngestionJobSummary

Provides information about a specified data ingestion job, including dataset information, data ingestion configuration, and status.

Contents

DatasetArn

The Amazon Resource Name (ARN) of the dataset used in the data ingestion job.

Type: String

Length Constraints: Minimum length of 20. Maximum length of 2048.

Pattern: `arn:aws(-[^:]+)?:lookoutequipment:[a-zA-Z0-9\-]*:[0-9]{12}:dataset\/.+`

Required: No

DatasetName

The name of the dataset used for the data ingestion job.

Type: String

Length Constraints: Minimum length of 1. Maximum length of 200.

Pattern: `^[0-9a-zA-Z_-]{1,200}$`

Required: No

IngestionInputConfiguration

Specifies information for the input data for the data inference job, including data S3 location parameters.

Type: IngestionInputConfiguration (p. 162) object

Required: No

JobId

Indicates the job ID of the data ingestion job.

Type: String

Length Constraints: Maximum length of 32.

Pattern: `[A-Fa-f0-9]{0,32}`

Required: No

Status

Indicates the status of the data ingestion job.

Type: String

Valid Values: `IN_PROGRESS | SUCCESS | FAILED`

Required: No

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Java V2
- AWS SDK for Ruby V3

DataPreProcessingConfiguration

The configuration is the `TargetSamplingRate`, which is the sampling rate of the data after post processing by Amazon Lookout for Equipment. For example, if you provide data that has been collected at a 1 second level and you want the system to resample the data at a 1 minute rate before training, the `TargetSamplingRate` is 1 minute.

When providing a value for the `TargetSamplingRate`, you must attach the prefix "PT" to the rate you want. The value for a 1 second rate is therefore *PT1S*, the value for a 15 minute rate is *PT15M*, and the value for a 1 hour rate is *PT1H*

Contents

TargetSamplingRate

The sampling rate of the data after post processing by Amazon Lookout for Equipment. For example, if you provide data that has been collected at a 1 second level and you want the system to resample the data at a 1 minute rate before training, the `TargetSamplingRate` is 1 minute.

When providing a value for the `TargetSamplingRate`, you must attach the prefix "PT" to the rate you want. The value for a 1 second rate is therefore *PT1S*, the value for a 15 minute rate is *PT15M*, and the value for a 1 hour rate is *PT1H*

Type: String

Valid Values: `PT1S | PT5S | PT10S | PT15S | PT30S | PT1M | PT5M | PT10M | PT15M | PT30M | PT1H`

Required: No

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Java V2
- AWS SDK for Ruby V3

DatasetSchema

Provides information about the data schema used with the given dataset.

Contents

InlineDataSchema

Type: String

Length Constraints: Minimum length of 1. Maximum length of 1000000.

Required: No

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Java V2
- AWS SDK for Ruby V3

DatasetSummary

Contains information about the specific data set, including name, ARN, and status.

Contents

CreatedAt

The time at which the dataset was created in Amazon Lookout for Equipment.

Type: Timestamp

Required: No

DatasetArn

The Amazon Resource Name (ARN) of the specified dataset.

Type: String

Length Constraints: Minimum length of 20. Maximum length of 2048.

Pattern: `arn:aws(-[^:]+)?:lookoutequipment:[a-zA-Z0-9\-]*:[0-9]{12}:dataset\/.+`

Required: No

DatasetName

The name of the dataset.

Type: String

Length Constraints: Minimum length of 1. Maximum length of 200.

Pattern: `^[0-9a-zA-Z_-]{1,200}$`

Required: No

Status

Indicates the status of the dataset.

Type: String

Valid Values: `CREATED | INGESTION_IN_PROGRESS | ACTIVE`

Required: No

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Java V2
- AWS SDK for Ruby V3

InferenceExecutionSummary

Contains information about the specific inference execution, including input and output data configuration, inference scheduling information, status, and so on.

Contents

CustomerResultObject

Type: S3Object (p. 168) object

Required: No

DataEndTime

Indicates the time reference in the dataset at which the inference execution stopped.

Type: Timestamp

Required: No

DataInputConfiguration

Specifies configuration information for the input data for the inference scheduler, including delimiter, format, and dataset location.

Type: InferenceInputConfiguration (p. 155) object

Required: No

DataOutputConfiguration

Specifies configuration information for the output results from for the inference execution, including the output S3 location.

Type: InferenceOutputConfiguration (p. 157) object

Required: No

DataStartTime

Indicates the time reference in the dataset at which the inference execution began.

Type: Timestamp

Required: No

FailedReason

Specifies the reason for failure when an inference execution has failed.

Type: String

Length Constraints: Minimum length of 1. Maximum length of 5000.

Pattern: `[\P{M}\p{M}]{1,5000}`

Required: No

InferenceSchedulerArn

The Amazon Resource Name (ARN) of the inference scheduler being used for the inference execution.

Type: String

Length Constraints: Minimum length of 20. Maximum length of 2048.

Pattern: `arn:aws(-[^:]+)?:lookoutequipment:[a-zA-Z0-9\-]*:[0-9]{12}:inference-scheduler\/.+`

Required: No

InferenceSchedulerName

The name of the inference scheduler being used for the inference execution.

Type: String

Length Constraints: Minimum length of 1. Maximum length of 200.

Pattern: `^[0-9a-zA-Z_-]{1,200}$`

Required: No

ModelArn

The Amazon Resource Name (ARN) of the ML model used for the inference execution.

Type: String

Length Constraints: Minimum length of 20. Maximum length of 2048.

Pattern: `arn:aws(-[^:]+)?:lookoutequipment:[a-zA-Z0-9\-]*:[0-9]{12}:model\/.+`

Required: No

ModelName

The name of the ML model being used for the inference execution.

Type: String

Length Constraints: Minimum length of 1. Maximum length of 200.

Pattern: `^[0-9a-zA-Z_-]{1,200}$`

Required: No

ScheduledStartTime

Indicates the start time at which the inference scheduler began the specific inference execution.

Type: Timestamp

Required: No

Status

Indicates the status of the inference execution.

Type: String

Valid Values: `IN_PROGRESS | SUCCESS | FAILED`

Required: No

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Java V2
- AWS SDK for Ruby V3

InferenceInputConfiguration

Specifies configuration information for the input data for the inference, including S3 location of input data..

Contents

InferenceInputNameConfiguration

Specifies configuration information for the input data for the inference, including timestamp format and delimiter.

Type: InferenceInputNameConfiguration (p. 156) object

Required: No

InputTimeZoneOffset

Indicates the difference between your time zone and Greenwich Mean Time (GMT).

Type: String

Pattern: `^(\+|\-)[0-9]{2}\:[0-9]{2}$`

Required: No

S3InputConfiguration

Specifies configuration information for the input data for the inference, including S3 location of input data..

Type: InferenceS3InputConfiguration (p. 158) object

Required: No

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Java V2
- AWS SDK for Ruby V3

InferenceInputNameConfiguration

Specifies configuration information for the input data for the inference, including timestamp format and delimiter.

Contents

ComponentTimestampDelimiter

Indicates the delimiter character used between items in the data.

Type: String

Length Constraints: Minimum length of 0. Maximum length of 1.

Pattern: `^(\-|\_|\s)?$`

Required: No

TimestampFormat

The format of the timestamp, whether Epoch time, or standard, with or without hyphens (-).

Type: String

Pattern: `^EPOCH|yyyy-MM-dd-HH-mm-ss|yyyyMMddHHmmss$`

Required: No

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Java V2
- AWS SDK for Ruby V3

InferenceOutputConfiguration

Specifies configuration information for the output results from for the inference, including KMS key ID and output S3 location.

Contents

KmsKeyId

The ID number for the AWS KMS key used to encrypt the inference output.

Type: String

Length Constraints: Minimum length of 1. Maximum length of 2048.

Pattern: `^[A-Za-z0-9][A-Za-z0-9:_/+=,@.-]{0,2048}$`

Required: No

S3OutputConfiguration

Specifies configuration information for the output results from for the inference, output S3 location.

Type: InferenceS3OutputConfiguration (p. 159) object

Required: Yes

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Java V2
- AWS SDK for Ruby V3

InferenceS3InputConfiguration

Specifies configuration information for the input data for the inference, including input data S3 location.

Contents

Bucket

The bucket containing the input dataset for the inference.

Type: String

Length Constraints: Minimum length of 3. Maximum length of 63.

Pattern: `^[a-z0-9][\.\-a-z0-9]{1,61}[a-z0-9]$`

Required: Yes

Prefix

The prefix for the S3 bucket used for the input data for the inference.

Type: String

Length Constraints: Minimum length of 0. Maximum length of 1024.

Pattern: `(^$)|([\P{M}\p{M}]{1,1023}/$)`

Required: No

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Java V2
- AWS SDK for Ruby V3

InferenceS3OutputConfiguration

Specifies configuration information for the output results from the inference, including output S3 location.

Contents

Bucket

The bucket containing the output results from the inference

Type: String

Length Constraints: Minimum length of 3. Maximum length of 63.

Pattern: `^[a-z0-9][\.\-a-z0-9]{1,61}[a-z0-9]$`

Required: Yes

Prefix

The prefix for the S3 bucket used for the output results from the inference.

Type: String

Length Constraints: Minimum length of 0. Maximum length of 1024.

Pattern: `(^$)|([\P{M}\p{M}]{1,1023}/$)`

Required: No

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Java V2
- AWS SDK for Ruby V3

InferenceSchedulerSummary

Contains information about the specific inference scheduler, including data delay offset, model name and ARN, status, and so on.

Contents

DataDelayOffsetInMinutes

A period of time (in minutes) by which inference on the data is delayed after the data starts. For instance, if an offset delay time of five minutes was selected, inference will not begin on the data until the first data measurement after the five minute mark. For example, if five minutes is selected, the inference scheduler will wake up at the configured frequency with the additional five minute delay time to check the customer S3 bucket. The customer can upload data at the same frequency and they don't need to stop and restart the scheduler when uploading new data.

Type: Long

Valid Range: Minimum value of 0. Maximum value of 60.

Required: No

DataUploadFrequency

How often data is uploaded to the source S3 bucket for the input data. This value is the length of time between data uploads. For instance, if you select 5 minutes, Amazon Lookout for Equipment will upload the real-time data to the source bucket once every 5 minutes. This frequency also determines how often Amazon Lookout for Equipment starts a scheduled inference on your data. In this example, it starts once every 5 minutes.

Type: String

Valid Values: `PT5M | PT10M | PT15M | PT30M | PT1H`

Required: No

InferenceSchedulerArn

The Amazon Resource Name (ARN) of the inference scheduler.

Type: String

Length Constraints: Minimum length of 20. Maximum length of 2048.

Pattern: `arn:aws(-[^:]+)?:lookoutequipment:[a-zA-Z0-9\-]*:[0-9]{12}:inference-scheduler\/.+`

Required: No

InferenceSchedulerName

The name of the inference scheduler.

Type: String

Length Constraints: Minimum length of 1. Maximum length of 200.

Pattern: `^[0-9a-zA-Z_-]{1,200}$`

Required: No

ModelArn

The Amazon Resource Name (ARN) of the ML model used by the inference scheduler.

Type: String

Length Constraints: Minimum length of 20. Maximum length of 2048.

Pattern: `arn:aws(-[^:]+)?:lookoutequipment:[a-zA-Z0-9\-]*:[0-9]{12}:model\/.+`

Required: No

ModelName

The name of the ML model used for the inference scheduler.

Type: String

Length Constraints: Minimum length of 1. Maximum length of 200.

Pattern: `^[0-9a-zA-Z_-]{1,200}$`

Required: No

Status

Indicates the status of the inference scheduler.

Type: String

Valid Values: `PENDING | RUNNING | STOPPING | STOPPED`

Required: No

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Java V2
- AWS SDK for Ruby V3

IngestionInputConfiguration

Specifies configuration information for the input data for the data ingestion job, including input data S3 location.

Contents

S3InputConfiguration

The location information for the S3 bucket used for input data for the data ingestion.

Type: IngestionS3InputConfiguration (p. 163) object

Required: Yes

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Java V2
- AWS SDK for Ruby V3

IngestionS3InputConfiguration

Specifies S3 configuration information for the input data for the data ingestion job.

Contents

Bucket

The name of the S3 bucket used for the input data for the data ingestion.

Type: String

Length Constraints: Minimum length of 3. Maximum length of 63.

Pattern: `^[a-z0-9][\.\-a-z0-9]{1,61}[a-z0-9]$`

Required: Yes

Prefix

The prefix for the S3 location being used for the input data for the data ingestion.

Type: String

Length Constraints: Minimum length of 0. Maximum length of 1024.

Pattern: `(^$)|([\P{M}\p{M}]{1,1023}/$)`

Required: No

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Java V2
- AWS SDK for Ruby V3

LabelsInputConfiguration

Contains the configuration information for the S3 location being used to hold label data.

Contents

S3InputConfiguration

Contains location information for the S3 location being used for label data.

Type: LabelsS3InputConfiguration (p. 165) object

Required: Yes

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Java V2
- AWS SDK for Ruby V3

LabelsS3InputConfiguration

The location information (prefix and bucket name) for the s3 location being used for label data.

Contents

Bucket

The name of the S3 bucket holding the label data.

Type: String

Length Constraints: Minimum length of 3. Maximum length of 63.

Pattern: `^[a-z0-9][\.\-a-z0-9]{1,61}[a-z0-9]$`

Required: Yes

Prefix

The prefix for the S3 bucket used for the label data.

Type: String

Length Constraints: Minimum length of 0. Maximum length of 1024.

Pattern: `(^$)|([\P{M}\p{M}]{1,1023}/$)`

Required: No

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Java V2
- AWS SDK for Ruby V3

ModelSummary

Provides information about the specified ML model, including dataset and model names and ARNs, as well as status.

Contents

CreatedAt

The time at which the specific model was created.

Type: Timestamp

Required: No

DatasetArn

The Amazon Resource Name (ARN) of the dataset used to create the model.

Type: String

Length Constraints: Minimum length of 20. Maximum length of 2048.

Pattern: `arn:aws(-[^:]+)?:lookoutequipment:[a-zA-Z0-9\-]*:[0-9]{12}:dataset\/.+`

Required: No

DatasetName

The name of the dataset being used for the ML model.

Type: String

Length Constraints: Minimum length of 1. Maximum length of 200.

Pattern: `^[0-9a-zA-Z_-]{1,200}$`

Required: No

ModelArn

The Amazon Resource Name (ARN) of the ML model.

Type: String

Length Constraints: Minimum length of 20. Maximum length of 2048.

Pattern: `arn:aws(-[^:]+)?:lookoutequipment:[a-zA-Z0-9\-]*:[0-9]{12}:model\/.+`

Required: No

ModelName

The name of the ML model.

Type: String

Length Constraints: Minimum length of 1. Maximum length of 200.

Pattern: `^[0-9a-zA-Z_-]{1,200}$`

Required: No

Status

Indicates the status of the ML model.

Type: String

Valid Values: `IN_PROGRESS | SUCCESS | FAILED`

Required: No

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Java V2
- AWS SDK for Ruby V3

S3Object

Contains information about an S3 bucket.

Contents

Bucket

The name of the specific S3 bucket.

Type: String

Length Constraints: Minimum length of 3. Maximum length of 63.

Pattern: `^[a-z0-9][\.\-a-z0-9]{1,61}[a-z0-9]$`

Required: Yes

Key

The AWS Key Management Service (AWS KMS) key being used to encrypt the S3 object. Without this key, data in the bucket is not accessible.

Type: String

Length Constraints: Minimum length of 1. Maximum length of 1024.

Pattern: `[\P{M}\p{M}]{1,1024}[^/]$`

Required: Yes

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Java V2
- AWS SDK for Ruby V3

Tag

A tag is a key-value pair that can be added to a resource as metadata.

Contents

Key

The key for the specified tag.

Type: String

Length Constraints: Minimum length of 1. Maximum length of 128.

Pattern: `^(?!aws:)[a-zA-Z+-=._:/]+$`

Required: Yes

Value

The value for the specified tag.

Type: String

Length Constraints: Minimum length of 0. Maximum length of 256.

Pattern: `[\s\w+-=\.:/@]*`

Required: Yes

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Java V2
- AWS SDK for Ruby V3

Common Errors

This section lists the errors common to the API actions of all AWS services. For errors specific to an API action for this service, see the topic for that API action.

AccessDeniedException

You do not have sufficient access to perform this action.

HTTP Status Code: 400

IncompleteSignature

The request signature does not conform to AWS standards.

HTTP Status Code: 400

InternalFailure

The request processing has failed because of an unknown error, exception or failure.

HTTP Status Code: 500

InvalidAction

The action or operation requested is invalid. Verify that the action is typed correctly.

HTTP Status Code: 400

InvalidClientTokenId

The X.509 certificate or AWS access key ID provided does not exist in our records.

HTTP Status Code: 403

InvalidParameterCombination

Parameters that must not be used together were used together.

HTTP Status Code: 400

InvalidParameterValue

An invalid or out-of-range value was supplied for the input parameter.

HTTP Status Code: 400

InvalidQueryParameter

The AWS query string is malformed or does not adhere to AWS standards.

HTTP Status Code: 400

MalformedQueryString

The query string contains a syntax error.

HTTP Status Code: 404

MissingAction

The request is missing an action or a required parameter.

HTTP Status Code: 400

MissingAuthenticationToken

The request must contain either a valid (registered) AWS access key ID or X.509 certificate.

HTTP Status Code: 403

MissingParameter

A required parameter for the specified action is not supplied.

HTTP Status Code: 400

NotAuthorized

You do not have permission to perform this action.

HTTP Status Code: 400

OptInRequired

The AWS access key ID needs a subscription for the service.

HTTP Status Code: 403

RequestExpired

The request reached the service more than 15 minutes after the date stamp on the request or more than 15 minutes after the request expiration date (such as for pre-signed URLs), or the date stamp on the request is more than 15 minutes in the future.

HTTP Status Code: 400

ServiceUnavailable

The request has failed due to a temporary failure of the server.

HTTP Status Code: 503

ThrottlingException

The request was denied due to request throttling.

HTTP Status Code: 400

ValidationError

The input fails to satisfy the constraints specified by an AWS service.

HTTP Status Code: 400

Common Parameters

The following list contains the parameters that all actions use for signing Signature Version 4 requests with a query string. Any action-specific parameters are listed in the topic for that action. For more information about Signature Version 4, see Signature Version 4 Signing Process in the *Amazon Web Services General Reference*.

Action

The action to be performed.

Type: string

Required: Yes

Version

The API version that the request is written for, expressed in the format YYYY-MM-DD.

Type: string

Required: Yes

X-Amz-Algorithm

The hash algorithm that you used to create the request signature.

Condition: Specify this parameter when you include authentication information in a query string instead of in the HTTP authorization header.

Type: string

Valid Values: `AWS4-HMAC-SHA256`

Required: Conditional

X-Amz-Credential

The credential scope value, which is a string that includes your access key, the date, the region you are targeting, the service you are requesting, and a termination string ("aws4_request"). The value is expressed in the following format: *access_key*/*YYYYMMDD*/*region*/*service*/aws4_request.

For more information, see Task 2: Create a String to Sign for Signature Version 4 in the *Amazon Web Services General Reference*.

Condition: Specify this parameter when you include authentication information in a query string instead of in the HTTP authorization header.

Type: string

Required: Conditional

X-Amz-Date

The date that is used to create the signature. The format must be ISO 8601 basic format (YYYYMMDD'T'HHMMSS'Z'). For example, the following date time is a valid X-Amz-Date value: `20120325T120000Z`.

Condition: X-Amz-Date is optional for all requests; it can be used to override the date used for signing requests. If the Date header is specified in the ISO 8601 basic format, X-Amz-Date is not required. When X-Amz-Date is used, it always overrides the value of the Date header. For more information, see Handling Dates in Signature Version 4 in the *Amazon Web Services General Reference*.

Type: string

Required: Conditional

X-Amz-Security-Token

The temporary security token that was obtained through a call to AWS Security Token Service (AWS STS). For a list of services that support temporary security credentials from AWS Security Token Service, go to AWS Services That Work with IAM in the *IAM User Guide*.

Condition: If you're using temporary security credentials from the AWS Security Token Service, you must include the security token.

Type: string

Required: Conditional

X-Amz-Signature

Specifies the hex-encoded signature that was calculated from the string to sign and the derived signing key.

Condition: Specify this parameter when you include authentication information in a query string instead of in the HTTP authorization header.

Type: string

Required: Conditional

X-Amz-SignedHeaders

Specifies all the HTTP headers that were included as part of the canonical request. For more information about specifying signed headers, see Task 1: Create a Canonical Request For Signature Version 4 in the *Amazon Web Services General Reference*.

Condition: Specify this parameter when you include authentication information in a query string instead of in the HTTP authorization header.

Type: string

Required: Conditional

Document history for the Amazon Lookout for Equipment User Guide

The following table describes the documentation for this release of Lookout for Equipment.

- **API version:** latest
- **Latest documentation update:** November 19, 2021

Change	Description	Date
Update to grant retirement policy	AWS LookoutForEquipment removed the retire grant from the managed policy as the service will be using retiring grant principal to retire the grants. You dont need to provide the retire grant permissions in the managed policy.	November 19, 2021
General availability	This version supports the generally available release of Amazon Lookout for Equipment.	April 8, 2021
New guide and service	This version supports the preview release of Amazon Lookout for Equipment.	December 1, 2020