aws

User Guide

Amazon Lookout for Equipment

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. Lookout for Equipment automatically creates the best model to learn your equipment's normal operating conditions. The model is optimized to find abnormal equipment behavior that occurred 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.

Lookout for Equipment is designed to monitor fixed and stationary industrial equipment that operates with limited variability in operating conditions. This includes rotating equipment such as pumps, compressors, motors, computer numerical control (CNC) machines, and turbines. It is also targeted for Process industries with applications such as heat exchangers, boilers, and inverters. Lookout for Equipment is a back-end analytics service, meant to supplement, and plug into, existing maintenance systems.

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 – 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 – Explains some basic Lookout for Equipment concepts and shows you how to get started with analyzing your data.

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 an anomaly detection 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 interpret the relationships between sensors, and to detect deviations from normal behavior because asset failures are rare and even the same failure type might have its own unique data pattern. Detected failures are preceded by behavior or conditions that fall out of the normal behavior of the equipment, and Lookout for Equipment is designed to look for those behaviors or conditions.

Additionally, if available, you can highlight abnormal equipment behavior using labels. 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 also schedule the frequency with which Amazon Lookout for Equipment monitors the asset.

Amazon Lookout for Equipment step by step

1. Set up your account.
2. Create your project.
3. Format your data.
4. Add your dataset to your project.
5. Review the dataset ingestion.
6. Train your model.
7. Evaluate your model.
8. Schedule inference.
9. Review your inference results.

You repeat the preceding steps for each asset that you want to monitor.

Setting up your AWS account

Before you can start with Lookout for Equipment, you must sign up for an AWS account if you don't already have one.

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.

Sign up for an AWS account

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

To sign up for an AWS account

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

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

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

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

Create a user with administrative access

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

Secure your AWS account root user

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

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

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

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

Create a user with administrative access

1. Enable IAM Identity Center.

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

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

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

Sign in as the user with administrative access

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

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

Assign access to additional users

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

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

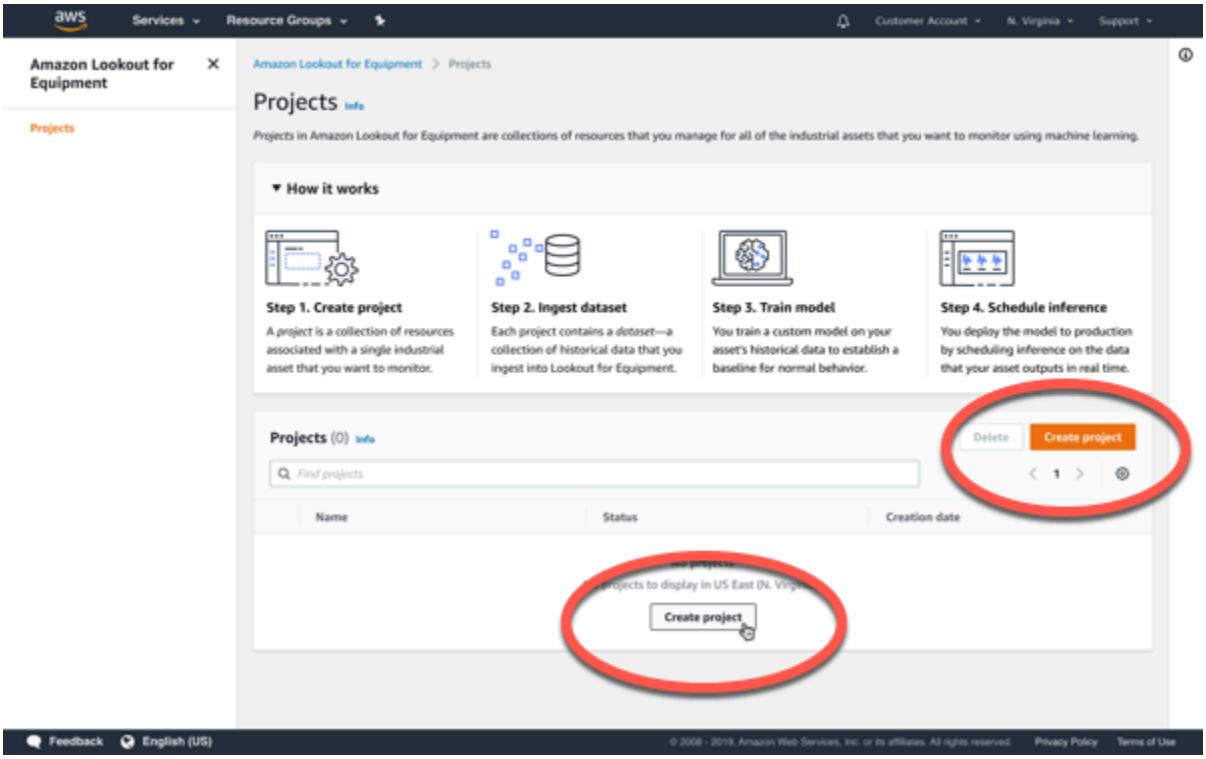
2. Assign users to a group, and then assign single sign-on access to the group.

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

Creating your project

After you set up your account, the next step is to create a project.

A project is a collection of resources associated with a single industrial asset that you want to monitor. Each project contains a dataset: a collection of historical data that you ingest into Lookout for Equipment.

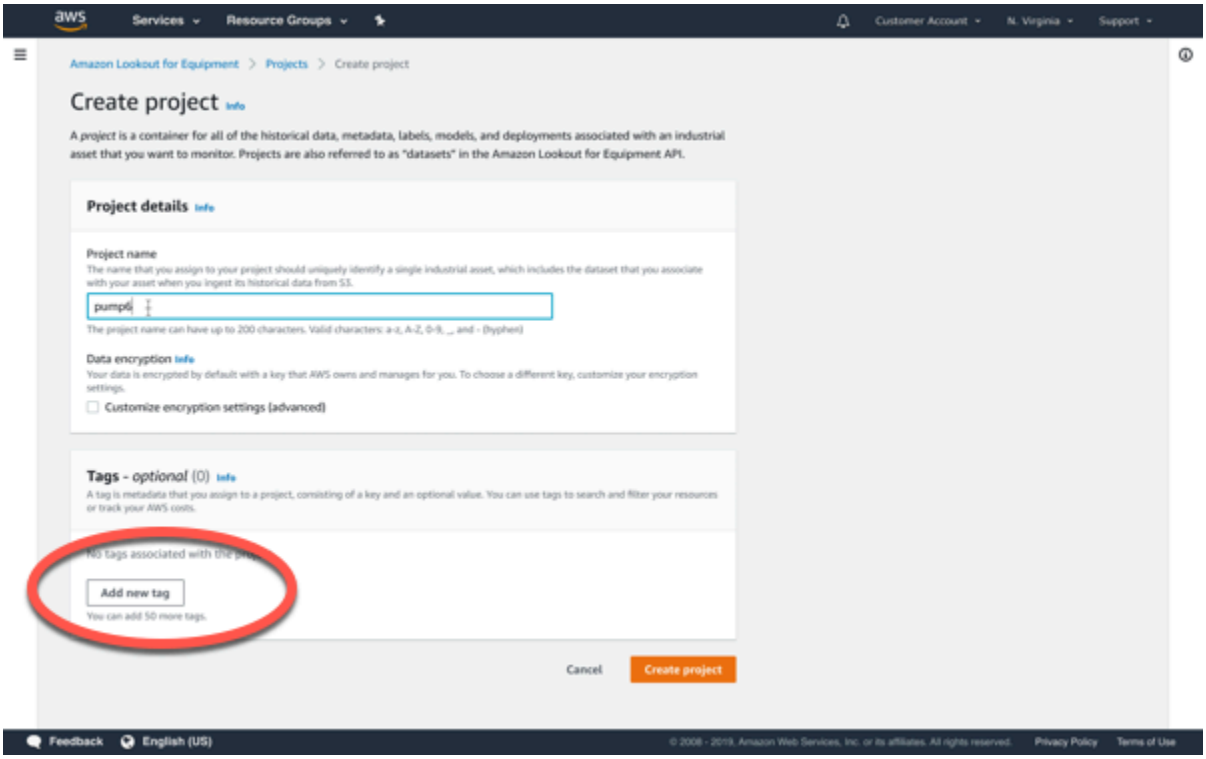


To create a project

1. Open Lookout for Equipment console.
2. Choose **Create project**.

Tagging your project

Optionally, you can assign tags to your project. Each tag is a label consisting of a user-defined key and value. Tags can help you associate your project with other resources in your AWS account. To learn more, see Tagging AWS resources.



Now that you've created your project, you'll need to check the formatting of your data. Then you'll need to organize your files before you upload them to Amazon S3.

Formatting your data

You've set up your account and created your project. Soon, you'll organize your data to help Lookout for Equipment determine an appropriate schema. But first, you must ensure that your data is formatted properly.

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:

- Storing all of the sensor data in one .csv file (recommended)
- Using one .csv file for each sensor

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.

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	Sens 1
2020/1/1 0:00	12

Timestamp	Sens 2
2020/1/1 0:05	31
2020/1/1 0:10	50
2020/1/1 0:15	9
2020/1/1 0:20	42

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

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

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	Sens 3
2020/1/1 0:00	34
2020/1/1 0:05	33

Timestamp	Sens 3
2020/1/1 0:10	35
2020/1/1 0:15	33
2020/1/1 0:20	34

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

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

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:

Category	Limit
minimum date range	**14 days**
maximum sensors per dataset	3000
maximum sensors per model	300
maximum length of a sensor name	200 characters
maximum size of each .csv file	5 GB

Category	Limit
minimum date range	**14 days**
maximum historical dataset size (combined .csv files)	50 GB
maximum files per historical dataset	1,000

- You can use the following delimiters for the data in the timestamp column: _ - (hyphen) and space
- 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
  - yyyy-MM-dd HH:mm
  - yyyy-MM-dd-HH:mm
  - yyyy/MM/dd'T'HH:mm:ss
  - yyyyMMdd'T'HH:mm
  - yyyyMMdd HH:mm
  - yyyyMMddHHmm
  - 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'T'HH:mm:ss
  - yyyy MM dd HH:mm:ss
  - yyyy MM dd'T'HH:mm

  - yyyy MM dd HH:mm
- The valid characters that you can use in the column names of the dataset are A-Z, a-z, 0-9, and . _ - (hyphen)

To learn more about the formats listed above, see the ISO 86021 standard.

Now that your data is formatted properly, it's time to organize your files.

Understanding the minimum date range

The minimum data range for a dataset in Lookout for Equipment is 14 days. However, there are situations in which you should include more than 14 days' worth of data.

The dataset that you submit should cover a period of time during which your asset functioned in all of its normal operating modes. This is necessary for Lookout for Equipment to recognize the difference between normal operation and anomalies.

If your dataset does not include examples of all of your asset's normal operating modes, then Lookout for Equipment may find more false positives. In other words, it may identify some of your operating modes, with which it is not familiar, as anomalies.

In such cases, you can help Lookout for Equipment accurately identify anomalies by labeling your data. For more information, see Understanding labeling.

Adding your dataset

> **Note**
>
> You can also add a dataset to your project or manage your dataset using the SDK.

You've created a project and you've uploaded your properly formatted data to Amazon S3. Now it's time to add the data to the project.

Setting your permissions

Lookout for Equipment requires permissions to access your data in Amazon S3, and to publish information about ingestion validation to CloudWatch Logs.

On the **Ingest dataset** page, under **Data source details,** under **IAM role**, select your preferred method of giving Lookout for Equipment the appropriate permissions.

- **Create an IAM role** is the default. If you select this option. Lookout for Equipment will create a role for you with the appropriate permissions.
- **Use an existing role**. If you have previously created an IAM role that you want to use with this dataset, you can select it here.
- **Enter a custom IAM role ARN**. This is another way to choose an existing role.

Logging your ingestion data

If you are creating a new role, you can check the box indicating that you want Lookout for Equipment to store log data in Amazon CloudWatch Logs.

You can also enable logging by modifying an existing role.

When you enable logging, Lookout for Equipment will record information about errors in the files submitted for ingestion. For example, the logs may help you identify duplicate timestamps, missing or invalid data, or rejected files.

For more information, see Viewing your ingestion history.

Choosing your schema

You have multiple options in how to structure your data in Amazon S3. Your choice of those options should be guided by one of two approaches.

Approach A: Your data is already organized in a particular way, and you prefer to keep it that way. In this case, choose the option below that best matches the way your data is currently organized.

Approach B: You haven't yet organized your data. In that case, examine the options below, and choose one that looks easier to implement. Then organize your data according to that option.

Before you proceed, be sure your data is formatted correctly.

> **Note**
>
> The following options assume that your files and folders have been organized by asset, which is what we recommend.
> However, organizing them by sensor, according to the same pattern, is also possible.

- **Option 1 (by filename):**
  - The name of the asset is the complete name of the CSV file.
  - All sensors from that asset are represented in that one CSV file.
  - The rest of the hierarchy of your Amazon S3 bucket doesn't affect the ingestion of data for this asset.
  - You can place multiple asset files into one folder.
  - There is one CSV file per asset.

  This is a good option if you have a small set of files, each named after a specific asset.
- **Option 2 (by part of filename):**
  - The name of the asset is part of the name of the CSV file. (Specifically, it's the part of the filename that precedes the delimiter.)
  - The rest of the hierarchy of your Amazon S3 bucket doesn't affect the ingestion of data for this asset.
  - There are multiple CSV files per asset.

This is a good option if you have to break up large files and give the smaller files similar names, such as pump1_january.csv and pump1_february.csv.

If you choose this option, then you must choose a delimiter. The delimiter indicates which character you are using, within the filename, to separate the name of the asset from the name of the sensor.

If applicable, select your delimiter from the dropdown menu at the bottom of the console window.

- **Option 3 (by folder name):**
  - The name of the asset is the complete name of the folder containing one or more CSV files.
  - The hierarchy in Amazon S3 is as follows:
    - Inside the Amazon S3 bucket is the folder you select when you specify the Amazon S3 location of your data source. Within that folder is a folder named after the asset.
    - Inside that folder are all the CSV files for that asset.
  - There can be multiple CSV files per asset.

  This is a good option if you have many files with long or inconsistent names, or a custom folder heirarchy that you want to retain.

Uploading your data to Amazon S3

You have organized the .csv files that contain your data. Now, the next step is to upload those files to Amazon S3.

Moving your data to Amazon S3 is a prerequisite to ingesting your data.

1. Open the Amazon S3 console.
2. Choose **Create bucket**
3. Under **Bucket name**, enter the name of your bucket. It might be useful to give your bucket the same name as your project, but that's optional.
4. Choose **Create bucket**
5. On the page with the list of buckets, choose your new bucket.
6. Choose **Create folder**.
7. Name your folder.

- If you chose to use one file for each asset, then the folder should be named after the facility.
- If you chose to use one file for each sensor, then the folder should be named after the asset.

8. Choose **Create folder**.
9. Choose the folder you created.
10. Choose one of the **Upload** buttons.
11. On the **Upload** page, choose **Add files**.
12. Add the appropriate files from your computer.
13. Choose **Upload**.
14. Return to the Lookout for Equipment console.
15. On the **Ingest dataset** page, under **Data source details**, indicate the location of the files you uploaded to Amazon S3.

So far, you've created your project, and (on this page) you've uploaded your well-organized data. Now it's time to integrate those steps by adding your uploaded data to your project.

Instructing Lookout for Equipment to ingest your data

You've set your permissions, chosen your schema, and (if applicable) chosen your delimiter. Now it is time for Lookout for Equipment to ingest your data.

1. Return to the **Ingest dataset** page.
2. Choose **Ingest dataset**.

You've ingested your data, but it's possible that there was an issue with the files, the sensors, or the ingestion job as a whole. To find out, you must now review data ingestion.

Reviewing data ingestion

Lookout for Equipment has ingested your data. Now it's time to make sure everything went according to plan.

> **Note**
>
> After ingestion, a red or green status bar will appear at the top of the console screen. Although a green status bar indicates success, there may still be issues with specific files or sensors. It is still necessary to review the data validation summary.

Topics

Next steps:

- If your entire job did not succeed, then a red bar has appeared at the top of the **Ingest dataset**page. In that case, it's time to review the job.
- If the job itself succeeded, but not every file was ingested, then you'll find yourself on the details page for your dataset, with an error message indicating that there was a problem ingesting certain files. In that case, it's time to check the files.
- If you did not receive any error messages regarding the ingestion job as a whole, or with issues with ingesting specific files, then it's time to look at your data's details by sensor.
- If you want to make changes to your dataset based on what you've learned so far, and then re-ingest it, skip to replacing your dataset.

Reviewing the job

Few datasets are perfectly formed. Missing or incorrectly formatted values are common. Therefore, it's not feasible to fail an ingestion job because of a single error.

Lookout for Equipment operates with a bias toward complete ingestion. In other words, when it encounters a problem in the ingested data, Lookout for Equipment attempts to fix that problem automatically. Then it alerts you to whatever issues it encountered, and lets you know what fixes it implemented.

If your entire job fails, consider the following possibilities:

1. The files are not .csv files, or they are corrupted, or they are unreadable for some other reason.
2. The files were not named or organized as explained under Adding your data.
3. The files contain no data, or 100% of the data they contain is not formatted in a way that Lookout for Equipment recognizes.

If your ingestion job fails, check the issues above and make the appropriate adjustments. When you're ready to try again, go back to Adding your dataset.

> **Important**
>
> This page is about troubleshooting the ingestion of *an entire job*. You can also read about why some specific files don't get ingested, and about evaluating the data from specific sensors.

Checking the logs

If you enabled CloudWatch Logs, then the logs may help you troubleshoot ingestion issues. The published logs may include the following error codes:

- COMPLETE_SENSOR_DATA_MISSING : A sensor has no valid data assosicated with it. The log contains the sensor name and the associated component name.
- DATA_MISSING_IN_COLUMN : Data associated with a sensor is invalid at a particular timestamp. Along with the sensor name and associated component name, the log contains details about the timestamp and the associated file path.
- UNSUPPORTED_DATE_FORMATS : A value in the timestamp column is invalid. The log contains details about the timestamp string, the path of the file, and the associated component name.
- INSUFFICIENT_SENSOR_DATA : A sensor is associated with less than 14 days of data. The log contains the sensor name, the component name, and the date range of data (in days) associated with the sensor.

- DUPLICATE_TIMESTAMPS : A value in the timestamp column of the data is a duplicate entry. The timestamp in question and the associated file path are part of the log.
- FILES_NOT_INGESTED : A file was not ingested during the ingestion workflow. The log contains details about the file's path.

Checking the files

If Lookout for Equipment fails to ingest a particular file, consider the following possibilities:

- None of the sensors listed in the file have any data that can be ingested.
- The file is not a .csv file, or the file is corrupted, or the file cannot be read for some other reason.

To troubleshoot files that were not ingested:

1. From the **Job details** tab of the main console page for your dataset, note the names of any files that failed the ingestion process.
2. To address issues with file formatting, see Formatting your data.
3. To address issues with individual sensors, see Understanding sensor quality.
4. When you're ready to try again, see Replacing your dataset.

> **Important**
>
> This page is about troubleshooting the ingestion of *specific files*. You can also read about why the ingestion of an entire job can fail, and about evaluating the data from specific sensors.

Anticipating schema detection problems

The following circumstances will lead to the failure of an entire ingestion job:

- One or more column headers contain one or more invalid characters.

  A single invalid character in a single column in a single file is enough to fail an entire job involving multiple files.
- In a job consisting of a single file, that file has a formatting issue that prevents ingestion.

- In a job consisting of multiple files, every single file has a formatting issue that prevents ingestion.

The easiest way to prevent problems with file ingestion is to take the following precautions:

- Make sure your headers don't include any invalid characters, such as spaces.

  Valid characters are: 0-9, a-z, A-Z, and # $. \ - (hyphen) _ (underscore)
- Make sure that the timestamp column is the one furthest to the left in your CSV file.
- Make sure that you don't have any duplicated column headers.

Evaluating sensor grades

This is where you can dive deep and troubleshoot exactly why you're getting the error codes, and make some decisions about whether you want to remove some sensors from your dataset.

Even if your ingestion job succeeds as a whole, and all your individual files also ingest successfully, you may decide not to use all the data from your sensors.

For each sensor, Lookout for Equipment tallies up the number of issues that arise. Based on how many issues occur for each sensor, Lookout for Equipment issues that sensor a grade.

> **Important**
>
> This page is about evaluating the quality of the data coming from *specific sensors*. You can also read about why the ingestion of an entire job can fail, and about why the ingestion of a particular file can fail.

Sensor grades

- **High**

  No validation errors were detected in the data during ingestion. Data from sensors in this category is considered the most reliable for model training and evaluation.
- **Medium**

One or more potentially harmful validation errors were detected in the data during ingestion. Data from sensors in this category is considered less reliable for model training and evaluation.

- **Low**

  One or more significant validation errors were detected in the data during ingestion. There's a high probability that training a model on data from sensors in this category will result in poor model performance.

Individual sensor errors

Error	Explanation	Data quality	Action taken by Lookout for Equipment	Action recommended for customer
No data found	No data is present for this sensor.	Low	Cannot use data from this sensor	Do not use this sensor.
Insufficient data	Less than 14 days of data provided.	Low	Lookout for Equipment cannot use data from this sensor.	This sensor cannot be used.
Monotonic values detected	Data only goes up, only goes down, or remains virtually static.	Low	Lookout for equipment can use this sensor but there is a risk of high number of false positive alerts.	Review this sensor and update sensor if necessary. We recommend that you do not use monotonic sensors.
Large data gaps detected	Data has at least one gap longer than 30 days.	Medium	Lookout for Equipment will forward fill all the missing values.	Review missing values and update sensor if necessary. The data gaps

Error	Explanation	Data quality	Action taken by Lookout for Equipment	Action recommended for customer
				may cause false alerts.
Multiple operating modes detected	Data shifts between ranges.	Medium	Lookout for Equipment can use this sensor but there is a risk of high number of false positive alerts.	Multiple operating modes add variability. Ensure all normal modes of operation are present in both the training dataset and the evaluation dataset.
Missing values detected	Total number of missing values is above 10%	Medium	If used, the missing values will be forward filled.	Review the missing values and update the sensor if necessary.
Categorical values detected	This sensor has N=<10 distinct values.	Medium	Lookout for Equipment can use this sensor but there is a risk of high number of false positive alerts.	Review categorical values and update sensor if necessary. Categorical values may lead to a higher number of false positive alerts.

Error	Explanation	Data quality	Action taken by Lookout for Equipment	Action recommended for customer
Constant values detected	The value does not change over time.	Medium		This sensor can be used, but it is not likely to add value.
Non-numberical values detected	Non-numerical data is present in this sensor.	Medium	The unsupported data will be removed and treated as missing values, then forward filled.	Review the non numerical data and update sensor if necessary
Duplicate timestamps detected	There are two or more rows that have the exact same timestamp.	Medium	The last encountered data point will be ingested, and the remaining duplicates will be omitted.	Review the duplicate timestamps and update the sensor if necessary.

Choosing the best sensors for your project

Use this information to decide which sensors are right for your project.

A **high-grade sensor**, from the point of view of Lookout for Equipment, is a sensor that did not trigger any errors in the table above. However, just because it's eligible to contribute doesn't mean it should. For example, suppose that the sensor is not actually attached to the asset that you're trying to monitor.

Suppose that the sensor is attached, instead, to the leg of the table that the asset sits on. The sensor might collect data related to vibration or heat, and the data it collects may not trigger any of the errors in the table above. But that doesn't mean that the data is actually useful. The data the

sensor is collecting may not be relevant to the operation of your asset. Even if the data is revelant, another sensor, nearby but better positioned, may already be collecting the most useful data for that part of the asset. Just because the data from a particular sensor doesn't trigger any of the errors above, doesn't mean that it ought to be selected for your model.

A **medium-grade sensor** collects data that triggers at least one error from the table above. But that doesn't necessarily mean that you shouldn't use that sensor in your model. For example, your sensor may have been labeled as medium-grade because it duplicated a timestamp once over the course of 14 days.

Based on your knowledge of the asset and how the data was collected, you may decide that Lookout for Equipment's method of remediation (deleting all but the first record collected for duplicate timestamps) is appropriate and productive. On the other hand, after receiving the alert, you may review the data, find many duplicate timestamps, and decide that the duplications indicate a problem with how the data was collected. You may then decide not to use data from that sensor in your model.

Data from a **low-grade sensor** contains a problem that may interfere with the accuracy of your model. We recommend that you do not include sensors with low-grade data when building your model. However, you may still choose to do so.

Next Steps:

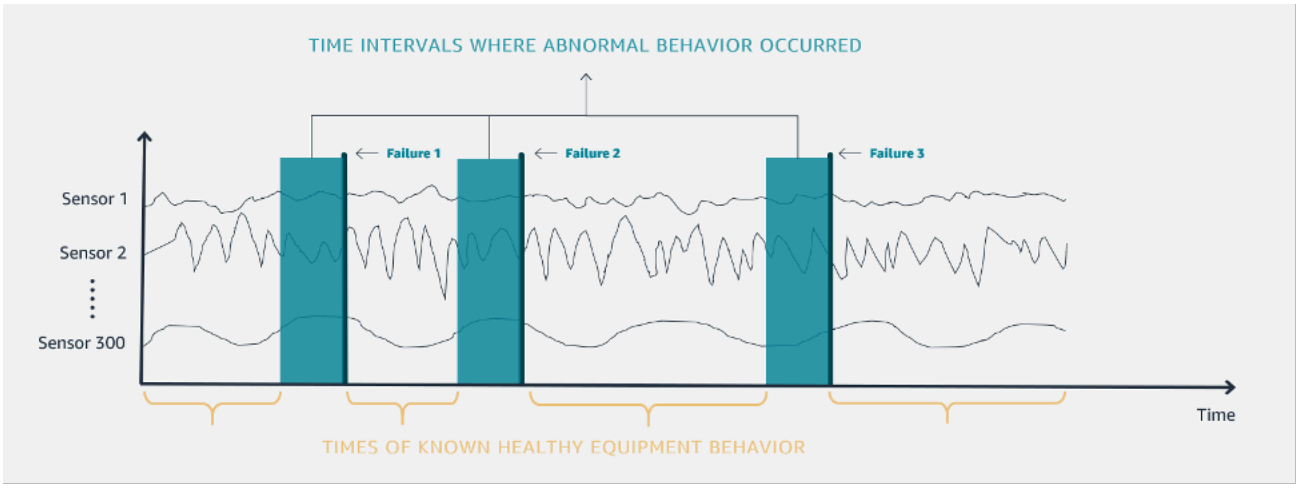
- If you've just chosen **Create model**, then it's time to Train your model.
- If you've changed your mind and decided to start over the data ingestion process, choose Replace your dataset.
- If this isn't the first time you've ingested a dataset with Lookout for Equipment, you may want to View your ingestion history.

Understanding labeling

Amazon Lookout for Equipment takes an input dataset, which it assumes is under normal operating conditions, and trains a model to detect deviations from this baseline, normal operation. However, if there are known periods of abnormal behavior in the input dataset, then that abnormal behavior can lead to less accurate models. To address this, we recommend that you use labels to identify the abnormal behavior in the input dataset and Lookout for Equipment can exclude that labeled data from model training. For example, if it is known that historical data for a machine contains data for planned or unplanned downtime states, you can use labels to identify and exclude the downtime state data from model training.

By using the labels as inputs to the model, Lookout for Equipment can use additional modeling techniques that can improve the accuracy of the model.

As an example, the following image shows the time intervals of known healthy equipment behavior and the time intervals of abnormal equipment behavior (that is, the width of the bars in the image).



In your labeling data, you define the abnormal time interval (bar width in image) from the actual failure point (for example, *Failure 1*). You provide the labeled data as a CSV file to model training. Each line of the CSV indicates the time intervals when your equipment did not function properly. For more information, see Labeling your data.

By consulting with Subject Matter Experts (SMEs) and understanding the various failure modes of the equipment you can provide a “lookahead” window indicating the amount of time the onset of the problem could have been detected.

You typically get information for labeling abnormal behavior data from two sources:

- Work orders which have been reported on the equipment. Work orders are notoriously subjective and inconsistent. That is why with Lookout for Equipment you only need to provide the approximate time of the failure and the approximate lookahead window in the labeling data you provide to Lookout for Equipment for model training.
- SMEs who work and maintain the equipment often have in depth knowledge about when machinery was in an erroneous state.

For information on how to apply labels when training a model, and the format of the label file, see Labeling your data.

Training your model

> **Note**
>
> You can also train your model with the SDK.

You've ingested your dataset, and you've reviewed any issues with the job, the files, or the sensors. You've also decided which sensors are providing the data that will be used to train your model. Now it's time to move forward with creating the model.

First, you'll specify the details of your model, such as its name, encryption settings, and tags.

Then, you'll configure your input data. During that process, you'll make decisions about the balance between your training dataset and your evaluation dataset, and whether or not to use data labels.

Topics

Specifying model details

> **Note**
>
> You can also view and train your model with the SDK.

This page describes the process of confirming your sensor inputs, naming your model, and choosing your tags.

1. On the **Dataset** page, under **Details by sensor**, use the checkboxes to select the sensor inputs that you want to include in your model.
2. Under **Model details**, enter a name for your model.
3. Optionally, customize your encryption settings. To learn more, see Data protection in Lookout for Equipment.

4. Optionally, associate tags with your model. To learn more, see Tagging AWS resources in the Amazon Web Services General Reference.
5. Choose **Next**.

Configuring your input data

Choosing your training and evaluation settings.

You can use Lookout for Equipment to train a model in one of the following ways:

- **Training set, no evaluation set, and no labels**

  The data you have ingested so far becomes, in its entirety, the entire basis for creating the model. Lookout for Equipment gets its concept of normal equipment behavior from the one set of data that has been ingested. All of the data uploaded during the ingestion phase becomes training data, no labeled data is used in the model training process. No data is designated for evaluating the model. Once the model has been created, its first use will be in production, on the real-time data streaming from your equipment.

  This setup requires the least amount of time and effort. But in the long run, a model set up this way may be less accurate than using one of the following methods.

- **Training set, evaluation set, and no labels**

  You divide the data you've uploaded so far (during the ingestion phase) into two parts: training data and evaluation data. Lookout for Equipment uses the training data to learn about normal behavior for your equipment. Then, Lookout for Equipment puts the model to the test on the evaluation data. You examine the model's performance on the evaluation data, and on that basis, you decide if the model is useful. You don't give Lookout for Equipment any direct indication of what you consider to be anomalous behavior for your equipment.

- **Training set, no evaluation set, and labels**

  You don't divide the ingested data into training data and evaluation data. It's all training data. But you do provide labeled data that indicates anomalous behavior.

- Training set, evaluation set, and labels

  You identify some of the ingested data as training data, and the rest of it as evaluation data. You also provide labeled data that indicates periods of anomalous behavior. This option may be

the most work to set up in the short term, but it may lead to a more accurate model in the long term.

Training, evaluating, and sampling

Now you'll need to decide how to split up your data between the training subset and the evaluation subset. The bigger the training set, the more data contributes to building your model. The bigger the evaluation set, the more chances you'll get to see how your model functions before you deploy it to production. A common breakdown is 80% training and 20% evaluation.

1. Choose the time range indicating your training data subset.
2. Choose the time range indicating your evaluation data subset.
3. Choose your sample rate. This is the rate at which the data will be sampled. A lower sample rate means that less data will be used, but the model will build faster. A higher sample rate means that more data will be used, but the model will take longer to build.
4. Enter your off-time indicators (optional).

   When your asset is off, Lookout for Equipment may interpret the absence of data as a behavioral anomaly (or as normal behavior). In order to prevent this, it's helpful to give Lookout for Equipment a clear indicator of whether or not your asset has been turned off. Choose one particular sensor whose status is indicative of whether your asset is active.

Now that you've configured your input data, the next step is to decide whether or not to use data labels.

If you already know that you do not want to label your data, you can skip ahead to ???.

Labeling your data

You've made a decision about your training and evaluation settings. If you decided to use labeled data, now is the time to upload it.

Lookout for Equipment takes labeling 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	Time 2
1	1/3/2 0:00
2	2/7/2 0:05
3	4/21/ 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.

To label your data

1. Create your labeled data.

   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.

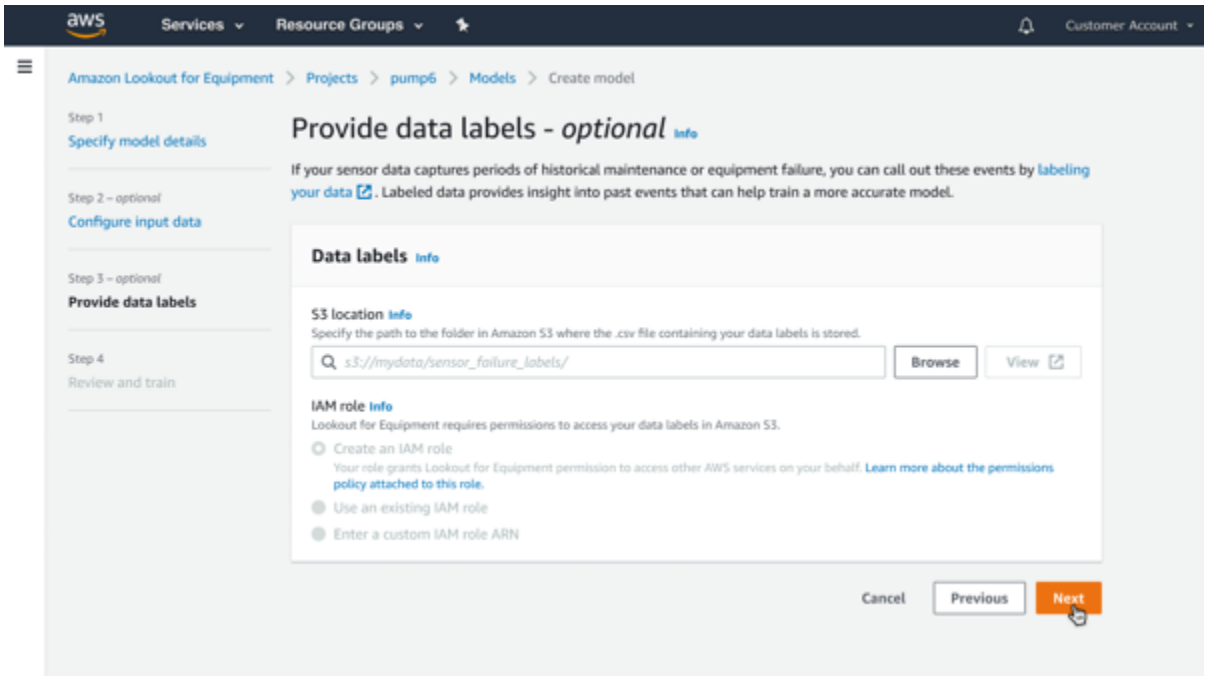
   The following example shows how your label data should appear as a .csv file.

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

2. Upload your data labels to Amazon S3. Here you'll follow the same procedure as in Uploading your data to Amazon S3.

   You can use the same Amazon S3 bucket or a different one. If you use the same one, it's a good practice to create a separate folder for your data labels.

3. In the Lookout for Equipment console, on the **Provide data labels** page, indicate the location of your data labels.



4. Choose your IAM role.

   This is the role that authorizes Lookout for Equipment to access the Amazon S3 bucket where your data labels are stored. If you're using the same bucket as before, you can choose the role that you already created. You can also select **Create an IAM role**, and the proper role will be created for you.

5. Choose **Next**.

6. Review your training settings and then train the model.

Starting the training process

The **Review and train** page gives you a chance to change some of your settings before you start training your model.

- To review model details such as the name, the encryption key, or the AWS tags, see Specifying model details.
- To review your input data configuration, which is where you (optionally) differentiated your training data from your evaluation data and set your sample rate and off time parameters, see Configuring your input data.
- To review data labels, see Labeling your data

When you're ready to train your model, choose **Train model**.

Evaluating your model

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.

Pointwise model diagnostics for an Amazon Lookout for Equipment model provides an evaluation of the model's performance at the individual events level. You can use the AWS SDK to get the pointwise model diagnostics for a model.

Topics

- Getting pointwise model diagnostics for a model (SDK)

Viewing the results for a model

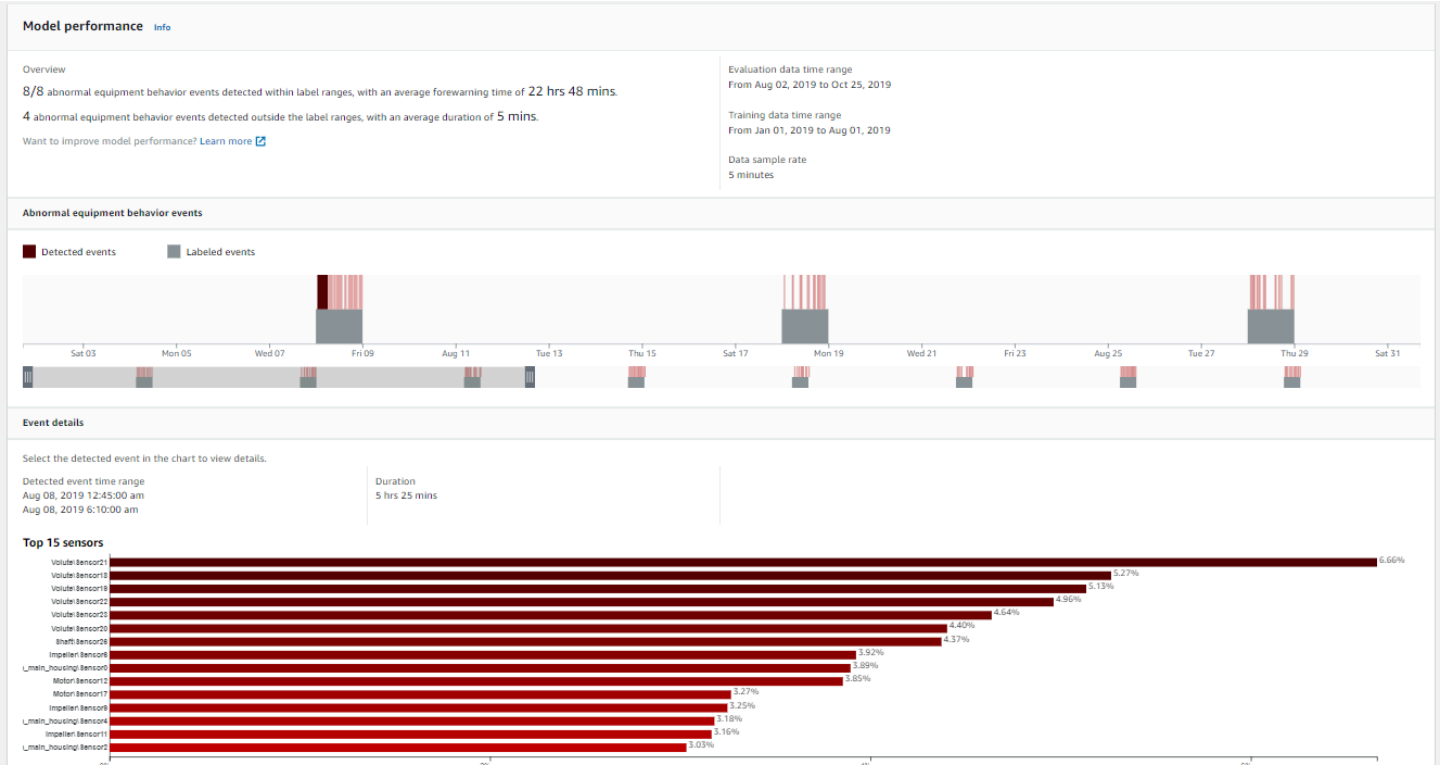
> **Note**
>
> You can get the evaluation results for a Lookout for Equipment model with the SDK.

To view the results for a model:

You can use this procedure to view model metrics in the console. To evaluate how the model performed, you must provide data labels. If you provide data labels, 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.



Getting pointwise model diagnostics for a model (SDK)

Pointwise model diagnostics for an Amazon Lookout for Equipment model is an evaluation of the model performance at the individual events. During training, Amazon Lookout for Equipment generates an anomaly score and sensor contribution diagnostics for each row in the input dataset. A higher anomaly score indicates a higher likelihood of an abnormal event.

You get pointwise diagnostics when you train a model with CreateModel. The period for which Lookout for Equipment generates model diagnostics depends on the following:

- If you specify the `EvaluationDataStartTime` and `EvaluationDataEndTime` request parameters, Lookout for Equipment generates model diagnostics for the period of time between `EvaluationDataStartTime` and `EvaluationDataEndTime`.
- If you supply `TrainingDataStartTime` and `TrainingDataEndTime`, but don't supply `EvaluationDataEndTime` and `EvaluationDataStartTime`, Lookout for Equipment generates model diagnostics for the period between `TrainingDataStartTime` and `TrainingDataEndTime`.
- If you don't specify an Evaluation or Training time range, Lookout for Equipment generates model diagnostics for the entire ingested data range in the input dataset.

If you want pointwise diagnostics for an existing model, use UpdateModel to provide the model diagnostics configuration. Lookout for Equipment then creates pointwise diagnostics for the entire retraining period.

For both `CreateModel` and `UpdateModel` operations, you need to specify the ModelDiagnosticsOutputConfiguration request parameter. The `S3OutputConfiguration` field specifies the Amazon S3 location where you want Lookout for Equipment to save the pointwise model diagnostics for the training period. If you don't specify `ModelDiagnosticsOutputConfiguration`, Lookout for Equipment doesn't create pointwise model diagnostics for the model.

If you update `ModelDiagnosticsOutputConfiguration` with `UpdateModel`, Lookout for Equipment only generates pointwise model diagnostics for future model versions.

You must specify an IAM role in the `RoleArn` request parameter with permission to access the Amazon S3 bucket that you reference in `ModelDiagnosticsOutputConfiguration`.

Amazon Lookout for Equipment creates pointwise model diagnostics for a model as a JSON format file. Lookout for Equipment stores the JSON file as a compressed file (`model_diagnostics_results.json.gz`) in the location you specify in `ModelDiagnosticsOutputConfiguration`.

The following is example JSON for a model evaluation.


```
{"timestamp": "2021-03-11T22:24:00.000000", "prediction": 0, "prediction_reason":
 "MACHINE_OFF"}
{"timestamp": "2021-03-11T22:25:00.000000", "prediction": 1, "prediction_reason":
 "ANOMALY_DETECTED", "anomaly_score": 0.72385, "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, "prediction_reason":
 "NO_ANOMALY_DETECTED", "anomaly_score": 0.41227, "diagnostics": [{"name":
 "component_5feceb66\\sensor0", "value": 0.03533}, {"name": "component_5feceb66\
\sensor1", "value": 0.24063}, {"name": "component_5feceb66\\sensor2", "value":
 0.06327}, {"name": "component_5feceb66\\sensor3", "value": 0.08303}, {"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}]}
{"timestamp": "2021-03-11T22:27:00.000000", "prediction": 0, "prediction_reason":
 "NO_ANOMALY_DETECTED", "anomaly_score": 0.10541, "diagnostics": [{"name":
 "component_5feceb66\\sensor0", "value": 0.02533}, {"name": "component_5feceb66\
\sensor1", "value": 0.34063}, {"name": "component_5feceb66\\sensor2", "value":
 0.07327}, {"name": "component_5feceb66\\sensor3", "value": 0.03303}, {"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}]}
{"timestamp": "2021-03-11T22:28:00.000000", "prediction": 0, "prediction_reason":
 "NO_ANOMALY_DETECTED", "anomaly_score": 0.24867, "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}]}
{"timestamp": "2021-03-11T22:29:00.000000", "prediction": 1, "prediction_reason":
 "ANOMALY_DETECTED", "anomaly_score": 0.79376, "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}]}
```


The JSON fields are as follows:

- **timestamp** – The date and time (in ISO 8601 format) that the event occurred.
- **prediction** – The prediction that the model made for the event. `0` for a normal event. 1 for an abnormal event.
- **prediction_reason** – The reason for the prediction. Valid values are `ANOMALY_DETECTED`, `NO_ANOMALY_DETECTED`, `MACHINE_OFF`.

- **anomaly_score** – The anomaly score for the event. `anomaly_score` is a float value (0-1) where higher values indicate a higher likelihood that the event is abnormal.
- **diagnostics** – diagnostics information for the event.

> **Note**
>
> The model evaluation JSON format is the same as the JSON file in which Lookout for Equipment returns inference results. For more information, see Reviewing inference results in a JSON file.

If you have previously created pointwise model diagnostics, you can get the Amazon S3 location of the model diagnostics files by calling the DescribeModel or DescribeModelVersion operations and checking the `ModelDiagnosticsOutputConfiguration` response field. If you have not previously created an evaluation for the model, the operations don't return the `ModelDiagnosticsOutputConfiguration` field.

Versioning your model

Machines' (assets') operating modes and health change over the course of their lifetimes. This is often referred to as *data drift*. Machines also go through expected, and unexpected, maintenance operations. Models developed for these machines must therefore be updated periodically to reflect these changes.

In the past, each training resulted in a separate model. When multiple models where related to the same asset, the only way to indicate that association was with the naming of the models (for example, pumpA_model1, pumpA_model2, and so forth), and you had to manage that association on your own.

With model versioning, you can store different versions of a model under the same model name, and then decide which model version you want to maintain as your active version. After you set your active version, Lookout for Equipment utilizes that version when it runs inference on your asset's sensor data.

Model versioning also helps maintain traceability and history of a given model, and its corresponding machine, over time.

Understanding model versioning

Currently, there are three ways to generate a model version:

- Training a model for the first time. In this case, as the parent model is created, so is a corresponding model version, which may be called Version 1.
- Importing a model from another account. In this case, if a model of the same name *does not* already exist in the target account, then the imported model becomes Version 1. If the imported model *does* already exist in the target account (and uses the same name), then the imported model gets the next available version number.
- Retraining a model. In this case, a new version is created. It has the same name as the parent model, but a version number incremented by 1. Note that the new version number will be 1 more than the *most recent* version number, regardless of which version is currently active.

The following APIs will help you work with, and understand, model versioning.

- ListModelVersions: all model versions for a given model, including the model version, model version ARN, and status. This list appears in the data type ModelVersionSummary.
- DescribeModelVersion: This API gives you relevant information (such as the data start and end times and the creation time). If the model fails, then this API will indicate why it failed.

Understanding model status

During the importation of a model, it will be in the state: `IMPORT_IN_PROGRESS`.

After you import a model, it will be in one of three states:

- `SUCCESS`
- `FAILED`
- `CANCELED`

Activating your model

This section describes how to set one of your model versions as the active model. The active model is the one that the inference scheduler uses during inferencing.

By default, in managed mode, the most recent version is active. However, if you are not satisfied with the most recent version, you can select a previous version.

Caveats to consider:

- If you try activating a model while inference is currently running, then that inference execution will continue to use the model that was active when inference began. Lookout for Equipment will pick up the newly activated model the next time you run inference.
- You cannot activate a model in the `FAILED` state.

The following API will help you in activating a model:

UpdateActiveModelVersion: This activates a particular model. You can only activate a model that is in the SUCCESS state.

Retraining your model

Understanding retraining

This section explains model retraining in the context of Lookout for Equipment.

Because machines operating modes and health change over time (leading to data drift), models developed for these machines should be updated periodically to reflect these changes. Retraining is the process of updating a machine learning model to take more recent information (that is, data and labels) about the machine into consideration. Retraining is the preferred method of addressing data drift.

When retraining a model Lookout for Equipment does not require you to run a new ingestion job. This is an important benefit, because you may have many assets running in your factory and setting up new ingestion jobs on thousands of machines could become an inconvenience.

> **Note**
>
> You may have been running inference on some models before AWS released the retraining feature for Lookout for Equipment. In that case, your inference data has not been collected and will not be available for retraining. In order to facilitate the retraining process, you should run a new ingestion job on those models.

By enabling retraining in Lookout for Equipment, you can schedule to have the system generate updated models on an ongoing basis without pausing your data-gathering process. Once a model is retrained it creates a new model version.

You may choose to manually control the activation of new models utilizing retraining metrics, or you may choose to allow Lookout for Equipment to activate your new models immediately utilizing managed mode, when appropriate.

Setting up your retraining scheduler

This section describes how set up your retraining scheduler.

The following APIs will help you manage your retraining scheduler:

- CreateRetrainingScheduler
- DescribeRetrainingScheduler
- ListRetrainingSchedulers
- StartRetrainingScheduler
- StopRetrainingScheduler
- DeleteRetrainingScheduler

When you set up a retraining schedule, there are two modes to be aware of for managing the selection of newly trained versions:

- In *manual mode*, the model is periodically retrained, but the new model versions are not activated until you indicate that it's time to activate them. This might be because you want to provide your own methodology, using metrics that describe the model, for determining if the newly trained version is better than the current version, or you might have a custom process to have extra testing done in a production environment which needs user sign-off.
- In *managed mode*, the model is periodically retrained, and then Lookout for Equipment automatically compares the metrics from the new version with the metrics version that is currently running. If Lookout for Equipment determines that the new version is more accurate than the current version, then Lookout for Equipment automatically activates the new version.

Both of these modes are set using the PromoteMode parameter in the CreateRetrainingScheduler API.

Understanding retraining data

This section explains how data is used for retraining, including the way that inference data is accumulated and stored.

When the inference scheduler is running, Lookout for Equipment accumulates and manages the inference data that it successfully processes. This allows Lookout for Equipment to use inference data as an input during retraining without the user having to manage providing the service updated data. Lookout for Equipment encrypts the stored data using either a customer-owned AWS KMS key configured as the model's ServerSideKmsKeyId, or, if there is no customer-owned AWS KMS key provided, then using a Lookout for Equipment-owned AWS KMS key.

Sensor data used for retraining comes from both a) the dataset associated with the model being retrained and b) the accumulated inference data for that model. Lookout for Equipment only uses the data from those two sources that falls within the LookbackWindow of the model's retraining scheduler. If, within that window, there is an overlap between the dataset and the accumulated inference data, the dataset takes priority.

During the retraining process, Lookout for Equipment also fetches labels from the location configured in the model's LabelsInputConfiguration.

Understanding retraining metrics

This section describes retraining metrics in the context of Lookout for Equipment.

If you are retraining in manual mode, then you may use these metrics to help you decide whether to activate a new model version.

The following table lists the model promotion criterion.

Old data has labels?	New data has labels?	Model promotion criterion	Metrics shown?
Yes	Yes	Select best model based on comparison metrics	Yes for both models
Yes	No	Select old model	No for both models
No	Yes	Select new model, if the new model meets the required quality threshold	Yes for new model only
No	No	Select new model	No for both models

Model metrics

The following `Model Metrics` are exposed in the DescribeModelVersion response. If a retrained model is the current active model version, then the same information is also returned in the DescribeModel response.

- **Recall:** The proportion of events that Lookout for Equipment correctly identified to the events that you labeled during the same period.

  For example, you may have labeled 10 events, but Lookout for Equipment only identified 9 of them. In this case, the recall is 90%.

- **Precision:** The proportion of true positives to total identified events.

  For example, if Lookout for Equipment identifies 10 events, but only 7 of those events correspond to events you labeled, then the precision is 70%.

- **MeanFractionalLeadTime:** A measurement of how quickly (relative to the length of the event), on average, Lookout for Equipment detects each event.

  For example, a typical event at your facility may last 10 hours. On average, it may take the model 3 hours to identify the event. In this case, the mean fractional lead time is 0.7.

- **AUC:** Area Under the ROC Curve (AUC) measures the ability of a machine learning model to predict a higher score for positive examples as compared to negative examples. A value between 0 and 1 that indicates how well your model is able to separate the categories in your dataset. A value of 1 indicates that it was able to separate the categories perfectly.

  For more information, see "A Visual Explanation of Receiver Operating Characteristic Curves and Area Under the Curve" at the *MLU Explain* website.

Model quality

If new data has labels, Lookout for Equipment uses the metrics to perform a quality assessment of the model. To get the quality assessment, check the `ModelQuality` field in the response from DescribeModel, DescribeModelVersion, ListModels, ListModelVersions, or CreateInferenceScheduler.

If Lookout for Equipment determines that the model quality is poor based on training metrics, the value is `POOR_QUALITY_DETECTED`. Otherwise, the value is `QUALITY_THRESHOLD_MET`. If the model is unlabeled, the model quality can't be assessed and the value of `ModelQuality`

is `CANNOT_DETERMINE_QUALITY`. In this situation, you can get a model quality assessment by adding labels to the input dataset and retraining the model.

If the previous model was labeled, Lookout for Equipment compares the metrics of each model on the new data to determine if the new model should be promoted. The quality assessment for the new model does not affect this comparison. If the previous model was unlabeled, Lookout for Equipment promotes the new model if the quality threshold is met.

For information about using labels with your models, see Understanding labeling.

For information about improving the quality of a model, see Best practices with Amazon Lookout for Equipment.

Importing your resources

You can copy existing Amazon Lookout for Equipment resources from one AWS account to another by using the import API operations. Additionally, we provide scripts that you can use to bulk import resources (datasets and models) from one AWS account to another.

Topics

Importing a model

Topics

Importing a model

This section describes how to copy existing Lookout for Equipment resources from one user account to another. For instance, as a user you might want to do this if you maintain different accounts for Development, QA, and Production pipelines to restrict user access at the various stages. Or, as an integrator you want to develop models in your user account and then provide them to your end users in their own AWS accounts. Importing is the mechanism allowing you to move Lookout for Equipment resources across these account boundaries.

In this guide, the term *resources* indicates the machine learning models that Lookout for Equipment generates, as well as the user datasets that you provide to train those models.

The following resources can be associated with a model version:

- the model version metadata
- the inference scheduler
- the training dataset
- the accumulated inference data
- the model performance metrics
- the retraining scheduler

The import resources APIs allow users to import the model version metadata, training datasets, accumulated inference data, and model metrics (if available). However, the inference scheduler and retraining schedulers are not copied over, and must be re-created in the target account.

In the context of performing an import, there is a *source account* and a *target account*. The API must be called from the target account, and it references information about the resources in the source account that you want to import.

In order for a target to be able to import resource from a source account, the source account must grant the appropriate permissions to the target account. See Controlling access to your model.

APIs related to importing

The following APIs will help you to import a model:

- ImportDataset: Imports the data that was used to train the original model.
- ImportModelVersion: Imports a model from another account. Use the attribute `SourceModelVersionArn` to indicate the version of the model that you want to import.

> **Note**
>
> If you plan to import both a model and the dataset that was used to create it, then you should first call ImportDataset, and then ImportModelVersion.

Whether or not you call both of these APIs depends on your use case. You may choose to import a model, but not the dataset that was used to create it. In that case, you would only call ImportModelVersion. You might do this because you already have a version of the same model in your account, and you are importing an improved version of the same model.

> **Note**
>
> If you plan to import both a model and the dataset that was used to create it, then you should first call ImportDataset, and then ImportModelVersion.

Importing a dataset

This section explains how to import your dataset using the Lookout for Equipment APIs.

For the purposes of this example, let us suppose that target account *2222222222* wants to import a dataset from source account *111111111111*.

> **Note**
>
> If the source account and the target account are the same, then you can skip the first two steps of this procedure.

1. The source account gives the target account permission to import the dataset *testDataset* with the following policy, using the PutResourcePolicy API.

```
{
  "Version": "2012-10-17",
  "Statement": {
    "Effect": "Allow",
    "Principal": {"AWS": "arn:aws:iam::2222222222:role/Admin"},
    "Action": [
      "lookoutequipment:ImportDataset"
    ],
    "Resource": "arn:aws:lookoutequipment:us-west-2:111111111111:dataset/
testDataset/00af0697-095b-433a-889c-9f4eed39db8b"
  }
}
```

2. Users of the source account may have used a AWS Key Management Service key to encrypt the original ingestion data. If that is the case, then the source account must give the target account permission to encrypt and decrypt the AWS KMS key.

For more information, see Authentication and access control for AWS Key Management Service in the *AWS Key Management Service Developer Guide*

3. The target account calls the ImportDataset API, supplying the dataset ARN (*arn:aws:lookoutequipment:us-west-2:111111111111:dataset/testDataset/00af0697-095b-433a-889c-9f4eed39db8b*). This action triggers the importation of the dataset.

> **Note**
>
> Labels associated with the source model will not be copied. Therefore, if labels are needed, the target account must explicitly provide them through the LabelsInputConfiguration parameter of the ImportModelVersion API.

Controlling access to your model

This section explains how a customer controls access to a model.

In order for a target to import resources from a source account, the source account must give permissions to the target account. These permissions are granted by applying resource policies to either the model, the model version, or the dataset resources.

Only the source account can apply, view or delete resource policies.

The following APIs will help you in controlling access to your model:

- PutResourcePolicy
- DescribeResourcePolicy
- DeleteResourcePolicy

Here is an example resource policy for setting the import permissions for a dataset:

```
{
  "Version": "2012-10-17",
  "Statement": {
    "Effect": "Allow",
    "Principal": {"AWS": "arn:aws:iam::2222222222:role/Admin"},
```

```
    "Action": [
      "lookoutequipment:ImportDataset"
    ],
    "Resource": "arn:aws:lookoutequipment:us-west-2:111111111111:dataset/
testDataset/00af0697-095b-433a-889c-9f4eed39db8b"
  }
}
```

This is an example policy for setting permissions for importing a specific model version:

```
{
  "Version": "2012-10-17",
  "Statement": {
    "Effect": "Allow",
    "Principal": {"AWS": "arn:aws:iam::2222222222:role/Admin"},
    "Action": [
      "lookoutequipment:ImportModelVersion"
    ],
    "Resource": "arn:aws:lookoutequipment:us-west-2:111111111111:model/
testModel/00af0697-095b-433a-889c-9f4eed39dbbc/model-version/1"
  }
}
```

This is an example policy to set the permissions to import all model versions (setting the permissions on a parent model):

```
{
  "Version": "2012-10-17",
  "Statement": {
    "Effect": "Allow",
    "Principal": {"AWS": "arn:aws:iam::2222222222:role/Admin"},
    "Action": [
      "lookoutequipment:ImportModelVersion"
    ],
    "Resource": "arn:aws:lookoutequipment:us-west-2:111111111111:model/
testModel/00af0697-095b-433a-889c-9f4eed39dbbc"
  }
}
```

By default, when you import a model version, you also accumulate inference data along with it. For information about changing that option, see Importing a model version with accumulated inference data.

> **Note**
>
> The policies above only support ImportDataset and ImportModelVersion. They cannot be used to give cross-account permissions to any other APIs associated with Lookout for Equipment.

What follows are explanations of several elements contained in the policies above.

- **Effect**: The effect can be `Allow` or `Deny`. By default, IAM users don't have permission to use resources and API actions, so all requests are denied. An explicit `Allow` overrides the default. An explicit `Deny` overrides any `Allows`.
- **Action**: The action is the specific Lookout for Equipment action for which you are granting or denying permission.
- **Resource**: The resource that's affected by the action.
- **Condition**: Conditions are optional. They can be used to control when your policy is in effect.

You may use the Lookout for Equipment ResourcePolicy APIs to control access to models, model versions, and datasets. For more information, see the API references for PutResourcePolicy and DeleteResourcePolicy.

Lookout for Equipment access control policies follow the same format as IAM policies. However, Lookout for Equipment policies will not appear in the IAM console, nor in the context of using IAM APIs. For more information, see Policies and permissions in IAM in the *IAM User Guide*.

Comparing access to model versions with access to parent models

When you give another account access to a model, you are giving that account access to *all* versions of that model.

When two policies exist, one for the model, and one for a version of that model, the more restrictive of the two policies applies.

If an account attempts to access a particular model or version, and no IAM policy exists for either the model itself or any version of that model, then access is not allowed.

For example, suppose you have a model called Pump_1. This model will serve as the parent model.

This model has two versions:

- Pump_1 version 1
- Pump_1 version 2

Now suppose that we set a policy *only* at the level of the parent model (Pump_1).

```
{
  "Version": "2012-10-17",
  "Statement": {
    "Effect": "Allow",
    "Principal": {"AWS": "arn:aws:iam::2222222222:role/Admin"},
    "Action": [
      "lookoutequipment:ImportModelVersion"
    ],
    "Resource": "arn:aws:lookoutequipment:us-west-2:111111111111:model/
Pump_1/00af0697-095b-433a-889c-9f4eed39dbbc"
  }
}
```

This policy indicates that all versions under model Pump_1 can be imported. No policies are specified at the level of the model *version*. Therefore, Lookout for Equipment will look at the permissions on the parent model level and apply them to all the versions.

Now, let us suppose that you also set a policy at the model version level. In this case, the model version will be Pump_1 Version 2.

```
{
  "Version": "2012-10-17",
  "Statement": {
    "Effect": "Deny",
    "Principal": {"AWS": "arn:aws:iam::2222222222:role/Admin"},
    "Action": [
      "lookoutequipment:ImportModelVersion"
    ],
    "Resource": "arn:aws:lookoutequipment:us-west-2:111111111111:model/
Pump_1/00af0697-095b-433a-889c-9f4eed39dbbc/model-version/2
  }
}
```

This policy indicates that Version 1 can be imported, but that Version 2 cannot be imported.

Lookout for Equipment looks at the permission at the model level and sees that it is set to **Allow**. Then, Lookout for Equipment will examine the permission for Version 2, and find that it is set to **Deny**.

Lookout for Equipment will then apply the more restrictive of the two permissions. Thus, Version 2 cannot be imported.

Finally, since there is no explicit permission on Version 1, Lookout for Equipment continues to apply the permission from the parent model (Allow). Therefore, Version 1 can be imported.

The table below illustrates the relationship between parent model permissions and model version permissions.

	Parent Model Permission -->	Allow	Deny	No Policy	
Model Version Permission | V					
Allow		Allow	Deny	Allow	
Deny		Deny	Deny	Deny	
No Policy		Allow	Deny	Deny	

Importing a model version with accumulated inference data

When you're importing a model version, you may want to also import the accumulated inference data along with it.

For example, if the retraining scheduler had the lookback window set to `P360D`, then the retraining execution would use data up to 360 days up to the current day of the retraining execution. If the inference data imported from the source account falls in that time period, then it would be used to retrain the model.

You can set three options with `InferenceDataImportStrategy` while calling the ImportModelVersion API:

- **NO_IMPORT:** No data with regard to inference will be imported
- **ADD_WHEN_EMPTY:** Only when the target model version has no inference data associated with it, then the inference data will be imported.

- **OVERWRITE:** Even if the target model version has some inference data associated with it, the inference data from the source account will overwrite it.

If nothing is set as input for `InferenceDataImportStrategy`, then the default setting is `NO_IMPORT`.

Before you can import a model version with the accumulated inference data, you must verify that the resource policy allows the importing of data related to the model version.

If you do not want to allow `ImportModelVersions` requests that import the inference data (that is, `InferenceDataImportStrategy` is set to `NO_IMPORT` in the request) then you should set the condition key `lookoutequipment:IsImportingData` to false on the resource policy of a model or model version that allows `ImportModelVersion` action.

If you want to allow `ImportModelVersions` requests with any `InferenceDataImportStrategy`, you don't need to additionally set `lookoutequipment:IsImportingData` on a resource policy of a model or model version that allows the `ImportModelVersion` action, because it is the default behavior when `lookoutequipment:IsImportingData` is not set.

It is unusual to only allow `ImportModelVersions` requests that import the inference data (that is `InferenceDataImportStrategy` is set to `ADD_WHEN_EMPTY` or `OVERWRITE` in the request), but if you have such a use case, you can explicitly set `lookoutequipment:IsImportingData` to true to achieve this permission control.

This is an example policy that will prevent the inference data from being imported:

```
{
  "Version": "2012-10-17",
  "Statement": {
    "Effect": "Allow",
    "Principal": {"AWS": "arn:aws:iam::2222222222:role/Admin"},
    "Action": [
      "lookoutequipment:ImportModelVersion"
    ],
    "Resource": "arn:aws:lookoutequipment:us-west-2:111111111111:model/
testModel/00af0697-095b-433a-889c-9f4eed39dbbc",
    "Condition": {
       "Bool": {
          "lookoutequipment:IsImportingData": "false"
```

```
        }
    }
  }
}
```

Bulk importing resources

You can import Amazon Lookout for Equipment resources (datasets and models) from a source AWS account to a target AWS account by using the ImportDataset (datasets) or ImportModelVersion (models) operations. If you need to import multiple resources, we recommend that you use the following scripts to bulk import resources.

- Resource CSV file script — Scans the source AWS account to get a list of all datasets and their respective active model versions. It then writes the list to an editable CSV file. You run the script in the source AWS account.
- Resource configuration script — Reads the CSV file generated by the Resource CSV file script and configures the resource policy for the target AWS account. The resource policy grants the target AWS account permissions to import resources from the CSV file. You run this script in the source AWS account.
- Bulk import script — Reads the CSV file that Resource CSV file script generates and calls `ImportDataset` on all datasets, and calls `ImportModelVersion` on the respective model versions. You run the script in the target AWS account, and after first running the Resource configuration script in the source AWS account.

Topics

Running the bulk import scripts

Although you can run the scripts in any environment that supports Python and boto3, we recommend that you run the scripts in an Amazon SageMaker notebook instance in Jupyter Lab. For more information, see https://jupyter.org/.

Topics

Creating the Amazon SageMaker notebook instances

Use the following procedure to create Amazon SageMaker notebook instances in the source AWS account and the target AWS account.

To create the Amazon SageMaker notebook instances

1. In the AWS account that you want to import resources from (source AWS account), open the Amazon SageMaker console and Create a Notebook Instance. For more information, JupyterLab versioning. Enter a name for the new notebook and use the default configurations.
2. Make sure that the IAM role that you use has following managed policy permissions:
   - AmazonSageMakerFullAccess.
   - AmazonLookoutEquipmentFullAccess. Alternatively, grant permissions to call the following Lookout for Equipment operations: `ListModels, DescribeModelVersion, PutResourcePolicy,importModelVersion,ImportDataset`.
3. In the target AWS account that you want to bulk import resources into, repeat steps 1 and 2.

Getting the resources from the source AWS account

Use the following procedures to get an editable CSV file of resources that you can import from a source AWS account and configure them for import into a target AWS account.

To get the resources from the source AWS account

1. In the source AWS account, open Jupyter Lab in the Amazon Sagemaker notebook instance that you created in step 1 of Creating the Amazon SageMaker notebook instances.
2. Copy each of the following scripts into separate cells within the notebook.
   - Resource CSV file script
   - Resource configuration script
3. Run the Resource CSV file script. The script prompts for the following:

- The AWS Region in which you want to run the script.
- The ID of the target AWS account to which you want to import the resources.

The script generates a CSV file (*import_input_file_{current_time}.csv*) that you use in the next step. If necessary you can make changes to the CSV before continuing. For more information, see Resource CSV file script

4. Run the Resource configuration script. The script prompts for the following information.

   - The AWS region in which you want to run the script.
   - Permission to update the existing policy, if the policy already exists for the source resource Amazon Resource Name (ARN).
   - The name and path of the csv file (*import_input_file_{current_time}.csv*) that you created in step 3.

   For more information, see Resource configuration script.

Importing the resources to the target AWS account

Use the following procedure to import the resources to the target AWS account.

To import the resources into the target AWS account

1. In the target AWS account, open Jupyter Lab in the Amazon SageMaker notebook instance that you created in step 3 of Creating the Amazon SageMaker notebook instances.
2. Copy the Bulk import script into a notebook cell.
3. Copy the file *import_input_file_{current_time}.csv* from the source AWS account to the target AWS account, in the same location where this script is located in the jupyter lab.
4. Run the Bulk import script. The script prompts for the following:

   - The AWS Region in which you want to run the script.
   - The name and path of the csv file (*import_input_file_{current_time}.csv*) that you copied in step 3.
5. After the script finishes, check the import results in the CSV file (*import_result_file_{current_time}.csv)* that the script creates. For more information, see Bulk import script.

Resource CSV file script

The script scans the source AWS account to get a list of active datasets and their respective active model versions. The script writes the list to a CSV file named *import_input_file_{current_time}.csv*. You use the CSV file as input to the next script (Resource configuration script).

The script populates the required fields and populates optional fields with None. If desired, you can supply your own values. Make sure you match datasets with the corresponding respective model version. You must not delete optional columns from the CSV file.

- **Current_model_name** — (Required) The current name of the model in the source AWS account.
- **New_model_name** — (Required) A name for the model in the target AWS account. By default the model name is the current model name. You can rename the model, if desired.
- **Current_dataset_name** — (Required) The current name of the active dataset in the source AWS account. This is the dataset name related to the model populated in Current_model_name field.
- **New_dataset_name** — (Required) The name for the imported dataset in the target AWS account. By default the dataset name is the value in `Current_dataset_name`. You can rename the dataset, if desired. If you only want to import the model and not import the dataset, use the existing active dataset name that's in the target AWS account. Additionally change the value of `Source_dataset_arn` to None.
- **Version(s)** — (Required) The total number of versions that the model has.
- **Version_to_import** — (Required) The model version which will be imported. By default the script populates `Version_to_import` with the active model version. You can specify a different model version, if desired.
- **Import?(Yes/No)** — (Required) Specifies if the script will import the dataset and model. By default the value is `Yes` If you don't want to import the dataset and model, change the value to No.
- **Target_account_id** — (Required) The ID of the target AWS account ID to which the script will import the resources. You enter this value when you run the script, but you can change the value as desired.
- **Source_dataset_arn** — (Required) The ARN of the dataset that will be imported. At the target AWS account in case If you don’t want to the import dataset and just want to perform import model, do the following:
  - Change the value of `Source_dataset_arn` to None.

- Change the value of `New_dataset_name` to the existing active dataset name, in the target AWS account.

- **Source_model_arn** — (Required) The ARN of the source model that the script will import.
- **Label_s3_bucket** — The name of the Amazon S3 bucket in the target AWS account where the label file exists. By default the script populates this value as None. We recommend that you leave this value unchanged, unless you want to use a different Amazon S3 bucket.
- **Label_s3_prefix** — The Amazon S3 bucket prefix path in the target AWS account where the label exists. By default the script populates this value as None. We recommend that you leave this value unchanged, unless you want to use a different Amazon S3 prefix.
- **Role_arn** — The ARN of the role that grants permission to read the label file at the target AWS account. By default the script populates this value as None. We recommend that you leave this value unchanged, unless you want to use a different role ARN.
- **kms_key_id** — The ID of the server-side AWS Key Management Service key. By default, the script populates this value as None. We recommend that you leave this value unchanged, unless you want to use a different server-side AWS KMS key ID.

Script

```
import boto3
import os
import csv
import time
import json
from botocore.config import Config
from datetime import datetime
import sys
import datetime


# By default these optional parameters are populated as None
label_s3_bucket = "None"
label_s3_prefix = "None"
kms_key_id = "None"
role_arn = "None"


def getTotalNumberOfModelVersions(model_name):
    total_length = 0
```

```
    try:
        response = lookoutequipment_client.list_model_versions(
            ModelName=model_name)
        total_length = len(response.get('ModelVersionSummaries'))
        next_token = response.get("NextToken")
        while next_token is not None:
            response = lookoutequipment_client.list_model_versions(
                ModelName=model_name, NextToken=next_token)
            next_token += len(response.get('ModelVersionSummaries'))
        return total_length
    except Exception as e:
        print("Exception thrown while listing models for model name:", model_name)


config = Config(connect_timeout=30, read_timeout=30,
                retries={'max_attempts': 3})
region_name = input(
    "Please enter the region to run the script('us-east-1', 'ap-northeast-2', 'eu-
west-1'): ")

lookoutequipment_client = boto3.client(
    service_name='lookoutequipment',
    region_name=region_name,
    config=config,
    endpoint_url='https://lookoutequipment.{region_name}.amazonaws.com'.format(
        region_name=region_name),
)


response = lookoutequipment_client.list_models()
target_account = None
current_time = datetime.datetime.now()
formatted_time = current_time.strftime("%Y-%m-%d_%H-%M-%S")
file_name = f"import_input_file_{formatted_time}.csv"
target_account = input("Please enter the target account id: ")
if len(target_account) != 12:
    print("Target account id is not valid hence terminating the script execution..")
    sys.exit()
with open(file_name, "a") as f:

 f.write("Current_model_name,New_model_name,Current_dataset_name,New_dataset_name,Version(s),Ve
(yes/
no),Target_account_id,Source_dataset_arn,Source_model_arn,Label_s3_bucket,Label_s3_prefix,Role_
 + '\n')
```

```
for model in response.get('ModelSummaries'):
    with open(file_name, "a") as f:
        f.write(model.get('ModelName') + "," + model.get('ModelName') +
 "," + model.get('DatasetName') + "," + model.get('DatasetName') + "," +
 str(getTotalNumberOfModelVersions(model.get('ModelName'))) + "," + str(model.get(
            'ActiveModelVersion')) + "," + "yes" + "," + target_account + "," +
 model.get('DatasetArn') + "," + model.get('ModelArn') + "," + label_s3_bucket + "," +
 label_s3_prefix + "," + role_arn + "," + kms_key_id + '\n')
next_token = response.get("NextToken")
while next_token is not None:
    response = lookoutequipment_client.list_models(NextToken=next_token)
    for model in response.get('ModelSummaries'):
        with open(file_name, "a") as f:
            f.write(model.get('ModelName') + "," + model.get('ModelName')
 + "," + model.get('DatasetName') + "," + model.get('DatasetName') + "," +
 str(getTotalNumberOfModelVersions(model.get('ModelName'))) + "," + str(model.get(
                'ActiveModelVersion')) + "," + "yes" + "," + target_account + "," +
 model.get('DatasetArn') + "," + model.get('ModelArn') + "," + label_s3_bucket + "," +
 label_s3_prefix + "," + role_arn + "," + kms_key_id + '\n')
    next_token = response.get("NextToken")

print("All the active models have been scanned and written to a file:", file_name)
```

Resource configuration script

This script configures the resource policies to let the target AWS account bulk import the resources. By using the CSV file (*import_input_file_{current_time}.csv*) that the Resource CSV file script creates, the script configures the resource policy for each dataset and model version ARN. The script updates existing resource policies for source datasets and model version ARNs to grant permissions to the target AWS account, along with any existing conditions.

After running this script, you can bulk import resources to the target AWS account by running the Bulk import script.

Script

```
import boto3
import os
import csv
import time
import json
from botocore.config import Config
```

```
from datetime import datetime
import sys
import datetime


# By default these optional parameters are populated as None
label_s3_bucket = "None"
label_s3_prefix = "None"
kms_key_id = "None"
role_arn = "None"


def getTotalNumberOfModelVersions(model_name):
    total_length = 0
    try:
        response = lookoutequipment_client.list_model_versions(
            ModelName=model_name)
        total_length = len(response.get('ModelVersionSummaries'))
        next_token = response.get("NextToken")
        while next_token is not None:
            response = lookoutequipment_client.list_model_versions(
                ModelName=model_name, NextToken=next_token)
            next_token += len(response.get('ModelVersionSummaries'))
        return total_length
    except Exception as e:
        print("Exception thrown while listing models for model name:", model_name)


config = Config(connect_timeout=30, read_timeout=30,
                retries={'max_attempts': 3})
region_name = input(
    "Please enter the region to run the script('us-east-1', 'ap-northeast-2', 'eu-
west-1'): ")

lookoutequipment_client = boto3.client(
    service_name='lookoutequipment',
    region_name=region_name,
    config=config,
    endpoint_url='https://lookoutequipment.{region_name}.amazonaws.com'.format(
        region_name=region_name),
)


response = lookoutequipment_client.list_models()
```

```
target_account = None
current_time = datetime.datetime.now()
formatted_time = current_time.strftime("%Y-%m-%d_%H-%M-%S")
file_name = f"import_input_file_{formatted_time}.csv"
target_account = input("Please enter the target account id: ")
if len(target_account) != 12:
    print("Target account id is not valid hence terminating the script execution..")
    sys.exit()
with open(file_name, "a") as f:

 f.write("Current_model_name,New_model_name,Current_dataset_name,New_dataset_name,Version(s),Ve
(yes/
no),Target_account_id,Source_dataset_arn,Source_model_arn,Label_s3_bucket,Label_s3_prefix,Role_
 + '\n')
for model in response.get('ModelSummaries'):
    with open(file_name, "a") as f:
        f.write(model.get('ModelName') + "," + model.get('ModelName') +
 "," + model.get('DatasetName') + "," + model.get('DatasetName') + "," +
 str(getTotalNumberOfModelVersions(model.get('ModelName'))) + "," + str(model.get(
            'ActiveModelVersion')) + "," + "yes" + "," + target_account + "," +
 model.get('DatasetArn') + "," + model.get('ModelArn') + "," + label_s3_bucket + "," +
 label_s3_prefix + "," + role_arn + "," + kms_key_id + '\n')
next_token = response.get("NextToken")
while next_token is not None:
    response = lookoutequipment_client.list_models(NextToken=next_token)
    for model in response.get('ModelSummaries'):
        with open(file_name, "a") as f:
            f.write(model.get('ModelName') + "," + model.get('ModelName')
 + "," + model.get('DatasetName') + "," + model.get('DatasetName') + "," +
 str(getTotalNumberOfModelVersions(model.get('ModelName'))) + "," + str(model.get(
                'ActiveModelVersion')) + "," + "yes" + "," + target_account + "," +
 model.get('DatasetArn') + "," + model.get('ModelArn') + "," + label_s3_bucket + "," +
 label_s3_prefix + "," + role_arn + "," + kms_key_id + '\n')
    next_token = response.get("NextToken")

print("All the active models have been scanned and written to a file:", file_name)
```

Bulk import script

This script scans the CSV file that the Resource CSV file script creates. For each row the script calls `ImportDataset` on the source dataset ARN. After the dataset import successfully finishes, the script then calls `ImportModelVersion` on the dataset's respective model version. If desired, you can call `ImportModelVersion` on an existing active dataset by populating the existing dataset

name in the columns `Current_dataset_name` and `New_dataset_name`. You must also set the `Source_dataset_arn` value to None.

The script outputs an import results CSV file (*import_result_file_{current_time}.csv*) that lists the following:

- **Source_resource_arn** — The ARN of the source dataset or source model.
- **Is_import_successful?** — Yes, if the resource import was successful. Otherwise, No.
- **type** — The type of the dataset (`dataset` or `model_version`).
- **Source_resource_name** — The name of the source resource.
- **New resource_name** — The new name for the resource in the target AWS account.
- **Version_to_import** — The model version in the source AWS account that was identified for import.
- **Failed_reason** — If the value of `Is_import_successful` is No, provides a reason for the failure.

Script

```
import boto3
import os
import csv
import time
import string
import random
import json
from botocore.config import Config
from datetime import datetime
import sys
import datetime


def activate_model_version(model_name, version, model_version_arn):
    try:
        response = lookoutequipment_client.update_active_model_version(
            ModelName=model_name, ModelVersion=version)
        print("Activated the model version: {} for the copied model:{}:".format(
            version, model_name))
    except Exception as e:
        print("Error while activating the model version:", e)
```

```
        with open(final_result_file, "a") as f:
            f.write(f"{model_version_arn},No,{e}\n")


config = Config(connect_timeout=30, read_timeout=30,
                retries={'max_attempts': 3})
region_name = input(
    "Please enter the region to run the script('us-east-1', 'ap-northeast-2', 'eu-
west-1'): ")

lookoutequipment_client = boto3.client(
    service_name='lookoutequipment',
    region_name=region_name,
    config=config,
    endpoint_url=f'https://lookoutequipment.{region_name}.amazonaws.com'
)

labels_configuration = {
    'S3InputConfiguration': {
        'Bucket': 'my-bucket',
        'Prefix': 'path/to/label_files/'
    }
}


source_input_file = input(
    "Please enter the source file name to start the import: ")
current_time = datetime.datetime.now()
formatted_time = current_time.strftime("%Y-%m-%d_%H-%M-%S")
final_result_file = f"import_result_file_{formatted_time}.csv"

with open(final_result_file, "a") as f:

 f.write("Source_resource_arn,Is_import_successful?,Type,Source_resource_name,New_resource_name
 + '\n')
with open(source_input_file) as csvfile:
    csvReader = csv.reader(csvfile, delimiter=',')
    for row in csvReader:
        client_token = ''.join(random.choices(
            string.ascii_lowercase + string.digits, k=10))
        if len(row) < 14 or len(row) > 14:
            print(
                "Skipping this Row as it doesn't match the format:
 Current_model_name,New_model_name,Current_dataset_Name,New_dataset_name,Version(s),Version_to_
```

```
(yes/
no),Target_account_id,Source_dataset_arn,Source_model_arn,Label_s3_bucket,Label_s3_prefix,Role_
            continue
        if row[6].lower() == "no":
            print(f"skipping import for model {row[9]}")
            with open(final_result_file, "a") as f:
                f.write(
                    f"{row[9]},No,skipped import as the input file says 'no' for import
\n")
            continue
        if row[0] == "Current_model_name" and row[1] == "New_model_name":
            continue
        is_dataset_import_success = True
        # import dataset logic
        if 'dataset' in row[8]:
            is_dataset_import_success = False
            print("Triggering import for dataset:", row[8])
            datasetnamefinal = None
            if row[3] == "None":
                datasetnamefinal = row[8].split(":")[5].split("/")[1]
            else:
                datasetnamefinal = row[3]
            import_status = None

            request = {
                'SourceDatasetArn': row[8],
                'DatasetName': datasetnamefinal,
                'ClientToken': client_token
            }

            if row[13] != "None":
                request['ServerSideKmsKeyId'] = row[13]

            try:
                response = lookoutequipment_client.import_dataset(**request)
                print("Latest response for the import dataset is:", response)
                import_status = response.get("Status")
                if import_status == "SUCCESS":
                    is_dataset_import_success = True
            except Exception as e:
                print("Error while importing a dataset:", e)
                with open(final_result_file, "a") as f:
                    f.write(
                        f"{row[8]},No,dataset,{row[2]},{row[3]},None,{e}\n")
```

```
                continue

            timeout_seconds = 900  # 15 minutes in seconds
            start_time = time.time()
            print("Latest import_status for dataset is:", import_status)
            while import_status != "SUCCESS" and is_dataset_import_success != True:
                response = lookoutequipment_client.import_dataset(**request)
                print("Latest response for the import dataset is:", response)
                import_status = response.get("Status")
                if import_status == "SUCCESS":
                    is_dataset_import_success = True
                    print("Import dataset completed for arn:", row[8])
                    with open(final_result_file, "a") as f:
                        f.write(
                            f"{row[8]},Yes,dataset,{row[2]},{row[3]},None,\n")
                if import_status == "FAILED":
                    print("import dataset has failed hence skipping the import model")
                    with open(final_result_file, "a") as f:
                        f.write(
                            f"{row[8]},No,dataset,{row[2]},{row[3]},None,check
ingestion job {response.get('JobId')} failure reason\n")
                    continue
                elapsed_time = time.time() - start_time
                if elapsed_time >= timeout_seconds:
                    print("Timeout reached. Exiting..")
                    is_dataset_import_success = False
                    with open(final_result_file, "a") as f:
                        f.write(
                            f"{row[8]},No,dataset,{row[2]},{row[3]},None,Timed out
checking the success status for import\n")
                    continue

                time.sleep(15)

        # import model logic
        if 'model' in row[9] and is_dataset_import_success:
            is_model_import_success = False
            model_version_arn = row[9] + "/model-version/" + row[5]
            print("Triggering import for model version:", model_version_arn)
            new_model_name = row[1]
            request = {
                'SourceModelVersionArn': model_version_arn,
                'DatasetName': datasetnamefinal,
                'ModelName': new_model_name,
```

```
                'ClientToken': client_token
            }

            if row[13] != "None":
                request['ServerSideKmsKeyId'] = row[13]

            if row[12] != "None":
                request['RoleArn'] = row[12]

            if row[10] != "None" and row[11] != "None":
                # populate label bucket and prefix if provided
                labels_configuration['S3InputConfiguration']['Bucket'] = row[10]
                labels_configuration['S3InputConfiguration']['Prefix'] = row[11]
                request['LabelsInputConfiguration'] = labels_configuration

            import_status = None

            try:
                response = lookoutequipment_client.import_model_version(
                    **request)
                print("Latest response for the import model is:", response)
                import_status = response.get("Status")
                if import_status == "SUCCESS":
                    is_model_import_success = True
            except Exception as e:
                print("Error while importing the model:", e)
                with open(final_result_file, "a") as f:
                    f.write(
                        f"{model_version_arn},No,model_version,{row[0]},{row[1]},
{row[5]},{e}\n")
                continue

            timeout_seconds = 900  # 15 minutes in seconds
            start_time = time.time()
            while import_status != "SUCCESS" and is_model_import_success != True:
                response = lookoutequipment_client.import_model_version(
                    **request)
                import_status = response.get("Status")
                print("Latest response for the import model is:", response)
                if import_status == "SUCCESS":
                    is_model_import_success = True
                    activate_model_version(response.get("ModelName"), response.get(
                        "ModelVersion"), model_version_arn)
                    with open(final_result_file, "a") as f:
```

```
                        f.write(
                            f"{model_version_arn},Yes,model_version,{row[0]},{row[1]},
{row[5]},None\n")
                if import_status == "FAILED":
                    print("Import model failed for arn:", model_version_arn)
                    with open(final_result_file, "a") as f:
                        f.write(
                            f"{model_version_arn},No,model_version,{row[0]},{row[1]},
{row[5]},check model version arn {response.get('ModelVersionArn')} details to know the
 failure reason\n")
                    continue

                elapsed_time = time.time() - start_time
                if elapsed_time >= timeout_seconds:
                    print("Timeout reached. Exiting..")
                    with open(final_result_file, "a") as f:
                        f.write(
                            f"{model_version_arn},No,Timed out checking the success
 status for import\n")
                    continue

                time.sleep(15)

            print("Import model completed for arn:", model_version_arn)
print(
    f"Import for all the dataset/models in the input file is completed, Check the
 results file {final_result_file} for details")
```

Scheduling inference

> **Note**
>
> You can also schedule inference with the AWS SDK for Python (Boto).

Starting inference

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.

To schedule inference, you specify the model, the schedule, the Amazon 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 Amazon S3 bucket.
8. For **S3 location** under **Output data**, specify the Amazon 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**.

Managing inference schedules

Stopping inference

This section explains how to halt the inference process.

1. From the AWS console, under Lookout for Equipment, from the left nav, choose **Inference schedules**.
2. If necessary, choose the **Active schedules** tab.
3. Select the schedule that you want to stop.
4. Choose **Stop**.
5. Choose **Stop schedule**.
6. Your stopped schedule will appear on the **Inactive schedules** tab.

Resuming inference

This section explains how to resume a stopped inference schedule.

1. From the AWS console, under Lookout for Equipment, from the left nav, choose **Inference schedules**.
2. If necessary, choose the **Inactive schedules** tab.
3. Choose **Set as active**.
4. Your stopped schedule will appear on the **Active schedules** tab.

Editing an active schedule

1. From the AWS console, under Lookout for Equipment, from the left nav, choose **Inference schedules**.
2. If necessary, choose the **Active schedules** tab.
3. Select the schedule that you want to edit.
4. Choose **Edit**.
5. On the pop-up window, choose **edit**.

> **Note**
>
> After you finish editing an inference schedule, the schedule returns to the activation status that it was in before you started editing.
> A schedule that was active before editing will return to active status after editing.

Editing an inactive schedule

1. From the AWS console, under Lookout for Equipment, from the left nav, choose **Inference schedules**.
2. If necessary, choose the **Inactive schedules** tab.
3. Select the schedule that you want to edit.
4. Choose **Edit**.
5. On the pop-up window, choose **edit**.

> **Note**
>
> After you finish editing an inference schedule, the schedule returns to the activation status that it was in before you started editing.
> A schedule that was inactive before editing will remain inactive after editing. To re-activate it, you must select the schedule on the **Inactive schedules** page and choose **Set as active**.

Delete an active schedule

1. From the AWS console, under Lookout for Equipment, from the left nav, choose **Inference schedules**.
2. If necessary, choose the **Active schedules** tab.
3. Select the schedule that you want to delete.
4. Choose **Delete**.
5. In the pop-up window, choose **Stop** to indicate that you are going to stop the schedule before deleting it.
6. In the pop-up window, enter *`delete`* in the text field.

7. In the pop-up window, choose **delete**.

Delete an inactive schedule

1. From the AWS console, under Lookout for Equipment, from the left nav, choose **Inference schedules**.
2. If necessary, choose the **Inactive schedules** tab.
3. Select the schedule that you want to delete.
4. Choose **Delete**.
5. In the pop-up window, enter *delete* in the text field.
6. In the pop-up window, choose **delete**.

Understanding the inference process

When you're planning your use of Lookout for Equipment, it may be useful to understand exactly what happens at each step of the inference process.

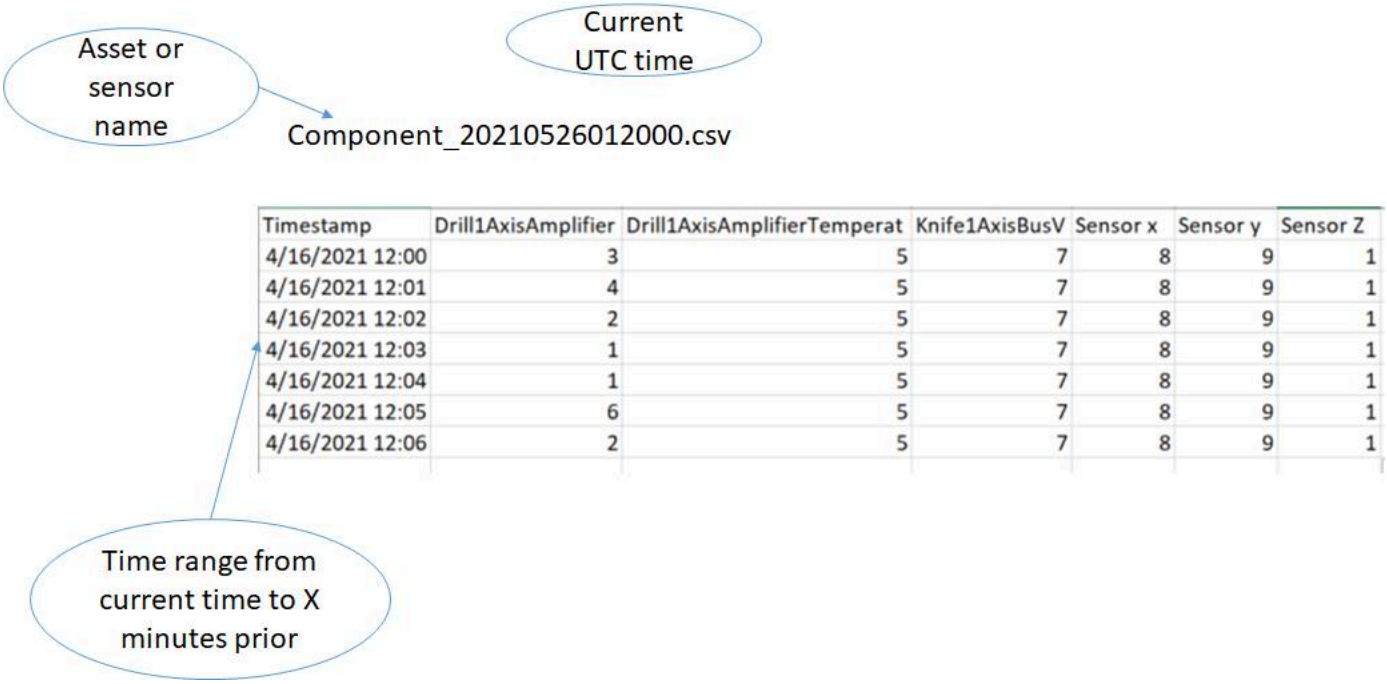
Understanding inference scheduling windows

- When you schedule inference, you may set your data upload frequency time to any of the following values, in minutes: 5, 10, 15, 30, 60.
- Lookout for Equipment then calculates the base number of segments per hour by dividing 60 by the length of your segments.
- You may also set an offset window in increments of minutes, from 0 to 60.
- At the beginning of each segment, Lookout for Equipment waits for the offset window to close before running inference.
- At the top of the hour, the process begins again.

Inference interval	Inferences per hour	First inference after 09:00 (with no offset)	First inference after 09:00 (with a 5-minute offset)
5	12	09:05	09:10

Inference interval	Inferences per hour	First inference after 09:00 (with no offset)	First inference after 09:00 (with a 5-minute offset)
10	6	09:10	09:15
15	4	09:15	09:20
30	2	09:30	09:35
60	1	10:00	10:05

The inference process



1. Lookout for Equipment looks for the component name (which can be the name of an asset or a sensor, depending on how your data was ingested).
2. Once the component name is found in the file name, Lookout for Equipment looks at the time stamp in the CSV file name.
3. The timestamp in the file name must be within the range of time that your scheduler is running. For example, if the scheduler is running every 5 minutes, then at 9:05, Lookout for Equipment

will look for any files that have a timestamp from 9:00 to 9:05. Any files with timestamps outside this range will be ignored for the inference run.

4. Lookout for Equipment automatically ingests the files with the right component name, and within the right time range.
5. Lookout for Equipment opens the CSV file and runs inference on any rows in the CSV file with timestamps that fit within the scheduler window. For example, if the scheduler is running every 5 minutes, and the current time is 9:05, then Lookout for Equipment will grab any files with the timestamp in the file name from 9:00 to 9:05, and will then run inference on any rows in the CSV with timestamps between 9:00 to 9:05.
6. The inference results are placed into your designated output bucket in a JSON file.
7. The steps above are repeated in perpetuity until the scheduler is turned off.

Reviewing inference results

After you've scheduled inference, you are able to see how your equipment is operating.

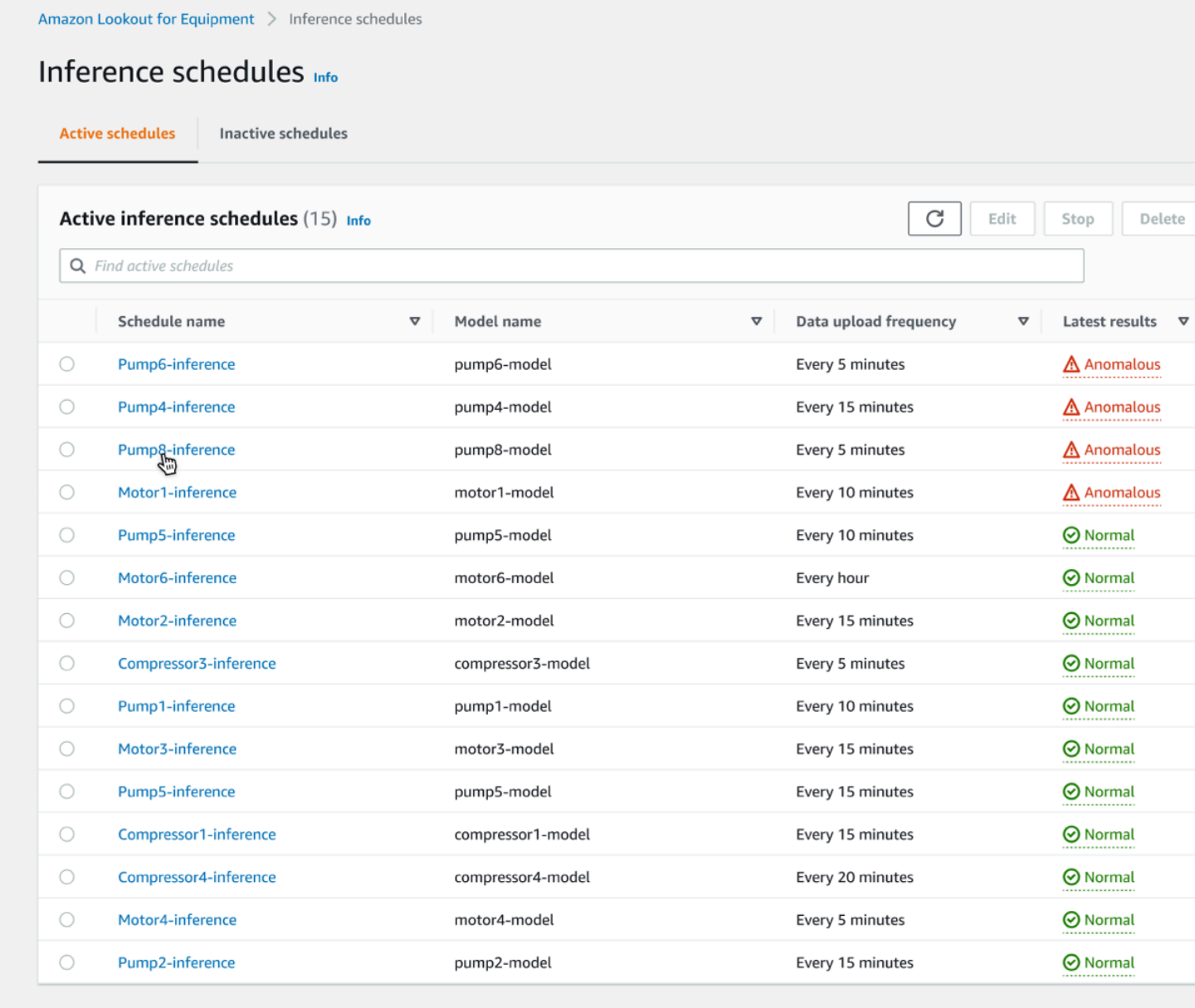
Topics

Reviewing inference results in the console

Using the main inference schedules page

On the inference schedules main page you'll find your list of inference schedules, both active and inactive (on different tabs). For each schedule, you'll find the model name, data upload frequency, and latest results.

In this context, *latest results* means the results from the most recent inference run.

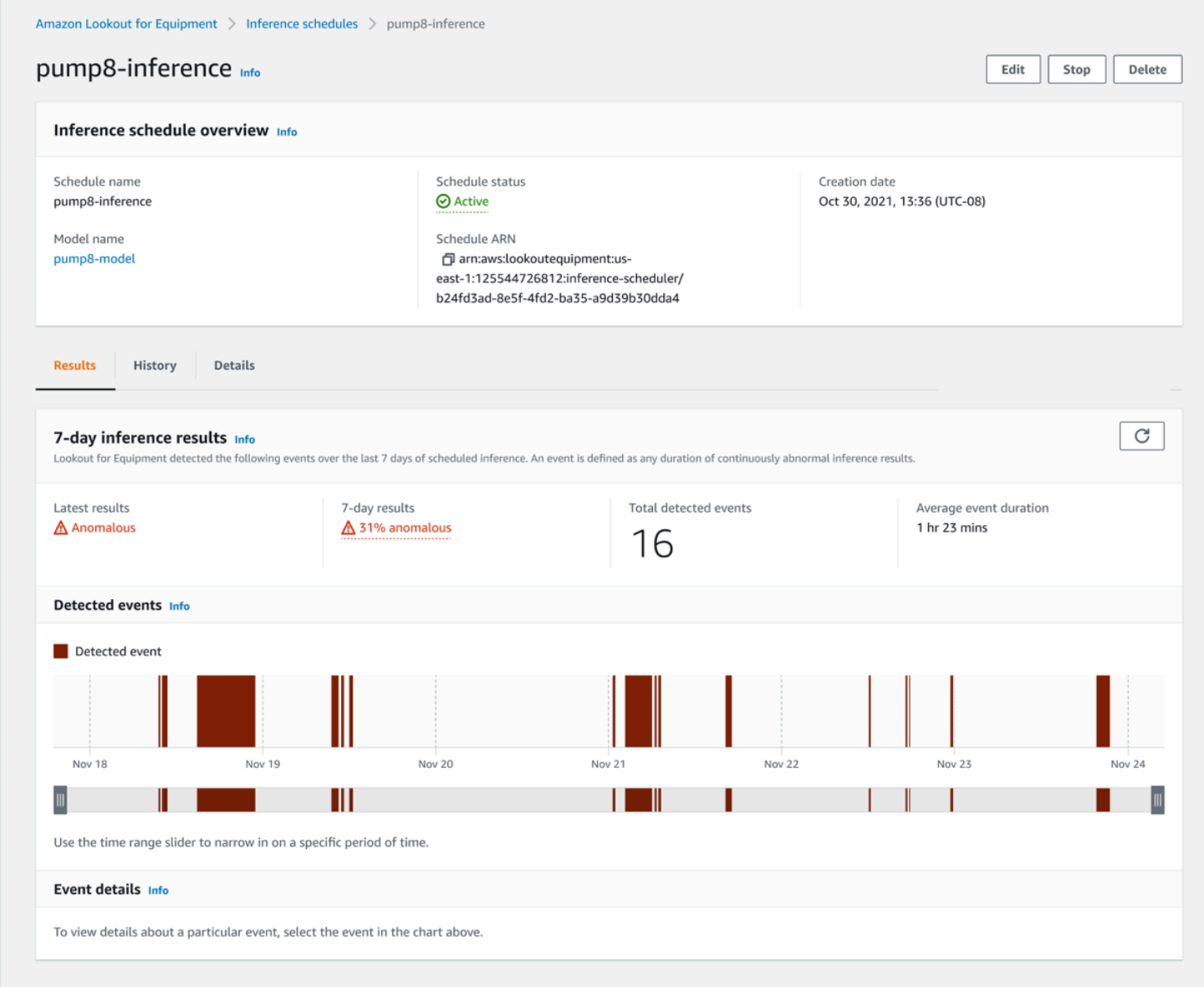


To edit, delete, stop, or restart a schedule, see Managing inference schedules.

Using the inference schedule detail page

On the inference schedule detail page you'll find details about the anomalous behavior of your assets, as presented in the context of a particular inference schedule.

You'll also find metadata about the schedule itself.



At the top of the results tab are the 7-day inference results. These results provide information about anomalous behavior that occurred over the past week.

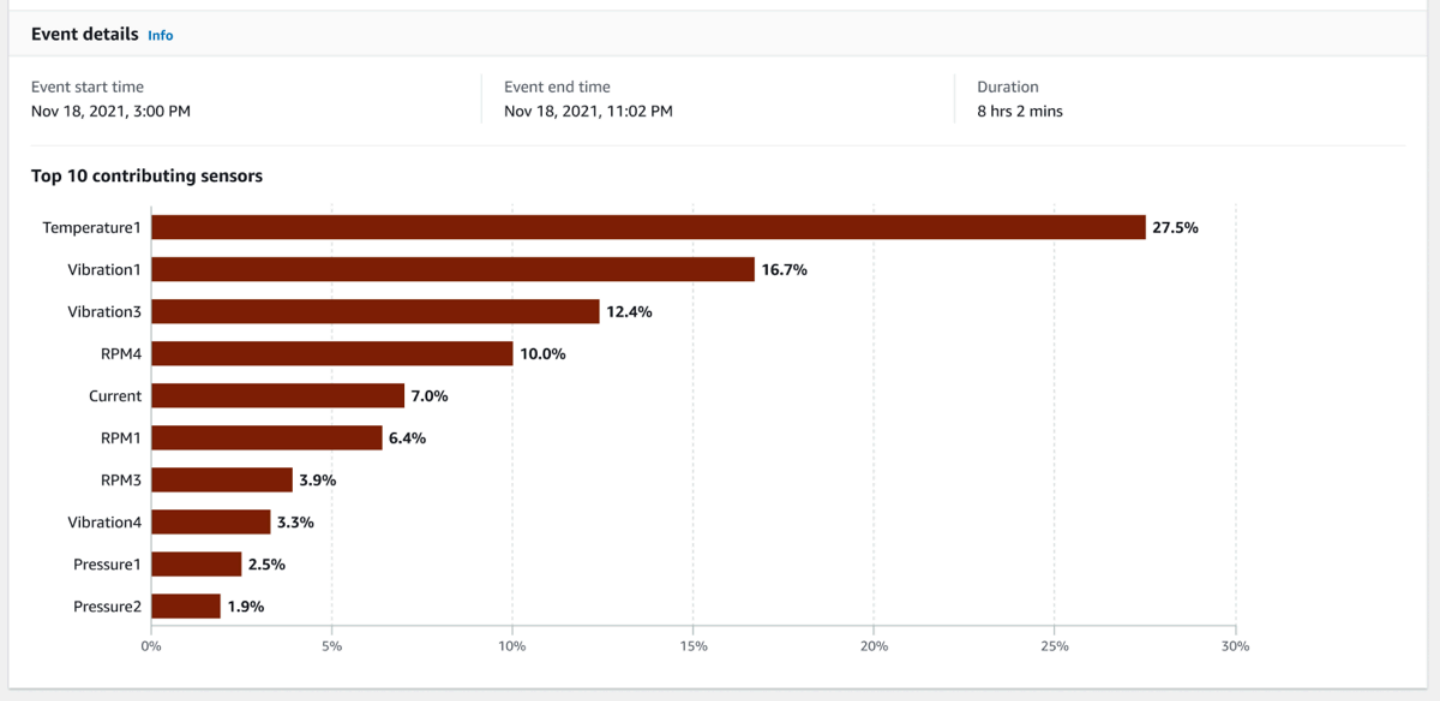
Latest results refers to results from the latest inference run.

7-day results indicates the percentage of hours during the last seven days, during which an anomaly was detected.

Use the slider to zoom in on a particular event (red bar).

Click on a particular event (red bar) to view details about it.

After you click on a particular event, the **Event details** tab indicates which sensors contributed the most to that event.



> **Note**
>
> Lookout for Equipment only records events that last longer than 5 minutes.

Reviewing inference results in a JSON file

The JSON file containing the inference results is stored in the Amazon Simple Storage Service (Amazon S3) bucket that you've specified.

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 shows example JSON output.

```
{"timestamp": "2021-03-11T22:24:00.000000", "prediction": 0, "prediction_reason":
 "MACHINE_OFF"}
{"timestamp": "2021-03-11T22:25:00.000000", "prediction": 1, "prediction_reason":
 "ANOMALY_DETECTED", "anomaly_score": 0.72385, "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, "prediction_reason":
 "NO_ANOMALY_DETECTED", "anomaly_score": 0.41227, "diagnostics": [{"name":
 "component_5feceb66\\sensor0", "value": 0.03533}, {"name": "component_5feceb66\
\sensor1", "value": 0.24063}, {"name": "component_5feceb66\\sensor2", "value":
 0.06327}, {"name": "component_5feceb66\\sensor3", "value": 0.08303}, {"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}]}
{"timestamp": "2021-03-11T22:27:00.000000", "prediction": 0, "prediction_reason":
 "NO_ANOMALY_DETECTED", "anomaly_score": 0.10541, "diagnostics": [{"name":
 "component_5feceb66\\sensor0", "value": 0.02533}, {"name": "component_5feceb66\
\sensor1", "value": 0.34063}, {"name": "component_5feceb66\\sensor2", "value":
 0.07327}, {"name": "component_5feceb66\\sensor3", "value": 0.03303}, {"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}]}
{"timestamp": "2021-03-11T22:28:00.000000", "prediction": 0, "prediction_reason":
 "NO_ANOMALY_DETECTED", "anomaly_score": 0.24867, "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}]}
{"timestamp": "2021-03-11T22:29:00.000000", "prediction": 1, "prediction_reason":
 "ANOMALY_DETECTED", "anomaly_score": 0.79376, "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.

If the value of `prediction_reason` isn't `MACHINE_OFF`, Amazon Lookout for Equipment returns an object that contains a diagnostics list, regardless of the value of `prediction`. 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.

The anomaly score is a value between 0 and 1 that indicates the intensity of the anomaly.

The prediction reason can be ANOMALY_DETECTED, NO_ANOMALY_DETECTED or MACHINE_OFF.

Viewing your ingestion history

To view your ingestion history:

1. Go to the main page for your dataset. (Amazon Lookout for Equipment -> Projects -> [asset name] -> Dataset
2. Select the **Ingestion history** tab.

For each ingestion job that succeeded, you may also view the associated logs in Amazon CloudWatch. The log group for your Lookout for Equipment logs will be `/aws/lookoutequipment/ingestion`. The logstream name will be the ingestion job ID.

For more information, see Publishing information about ingestion validation to Amazon CloudWatch Logs.

Replacing your dataset

Replacing your dataset allows you to change the data without re-creating the project from the beginning.

You may want to do this after reviewing the ingestion of your dataset, and addressing problems with the job, files, or sensors.

> **Note**
>
> If you want to change your schema, then you must start over with a new project.

The **Replace dataset** page is similar to the **Ingest dataset** page that you visited earlier in the workflow. The main difference is that the **Replace dataset** page does not ask you for information about how you named your .csv files. When you replace a dataset, Lookout for Equipment re-uses the schema detection information that you entered before.

To replace your dataset:

- From the **Dataset details** screen, choose **Replace dataset**.
- On the **Replace dataset** page, indicate the location of your data on Amazon S3 and choose your IAM role.
- Choose **Start ingestion**.

After the procedure above, you'll review your dataset ingestion once again.

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 (that is, labels)** – 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 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.

For information about using labels, see Understanding labeling and Labeling your data.

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

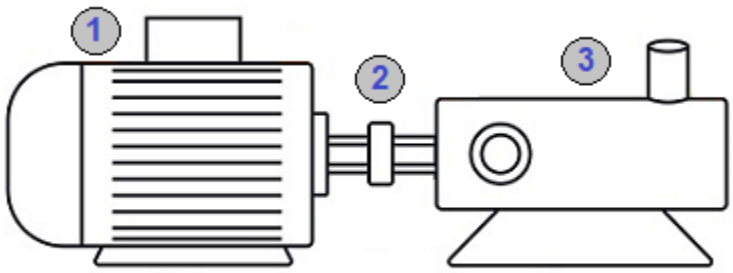
Lookout for Equipment identifies patterns in the dataset that help to detect critical issues, but it's the responsibility of a technician or subject matter expert (SME) to diagnose the problem and take corrective action, if needed. To ensure that you are getting the right output, we highly recommend 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.

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.

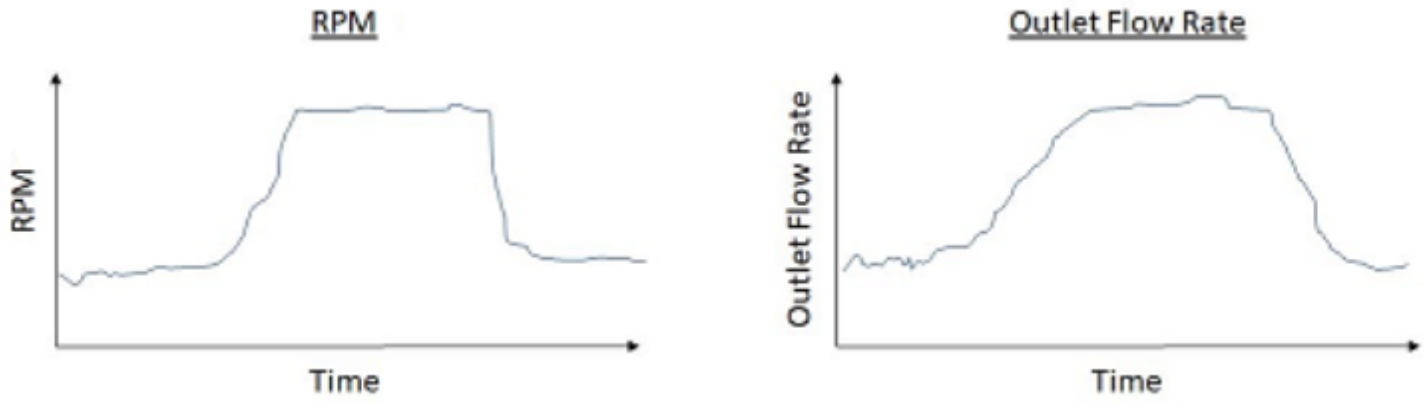
As an example of using Lookout for Equipment on data from a high-level machine , let's look at a fluid pump.



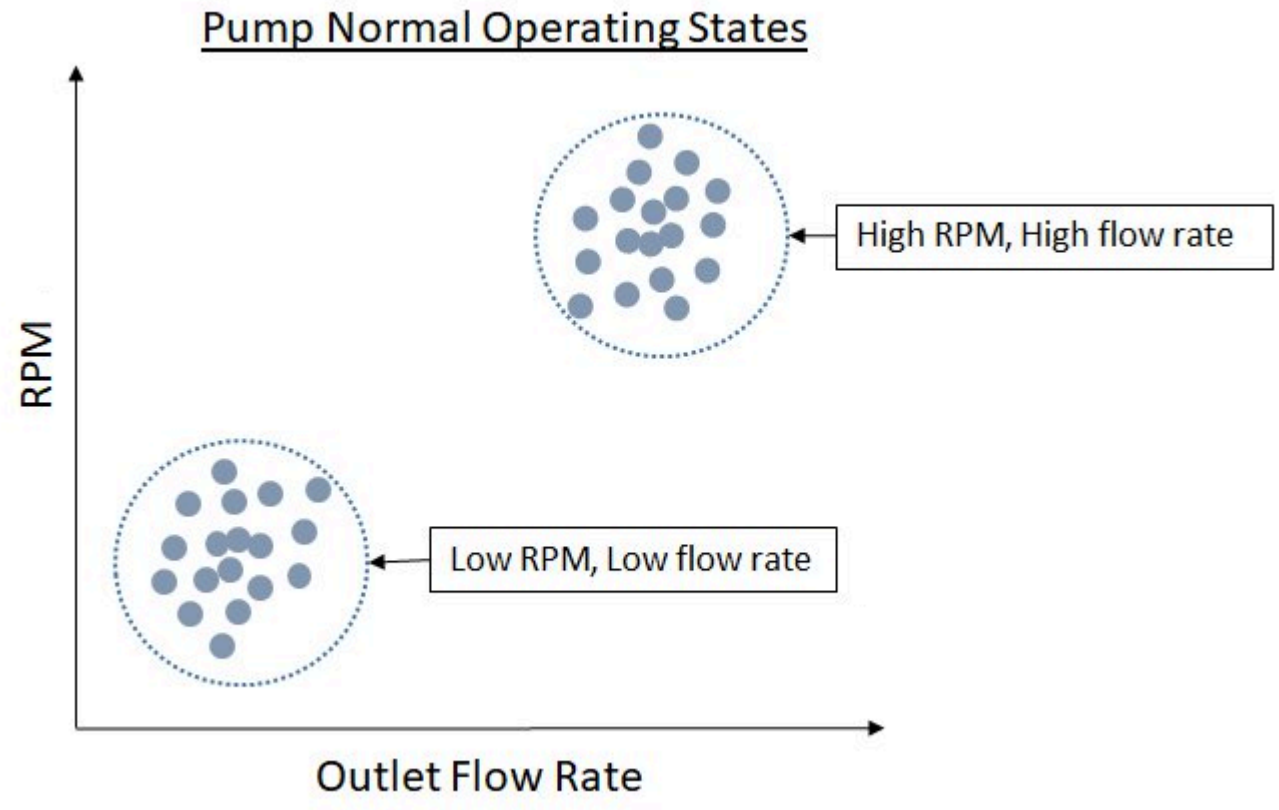
In a simplified form, this fluid pump consists of three major components and their sensors. Note that this is just an example and not a complete list of features and components of such a pump.

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.



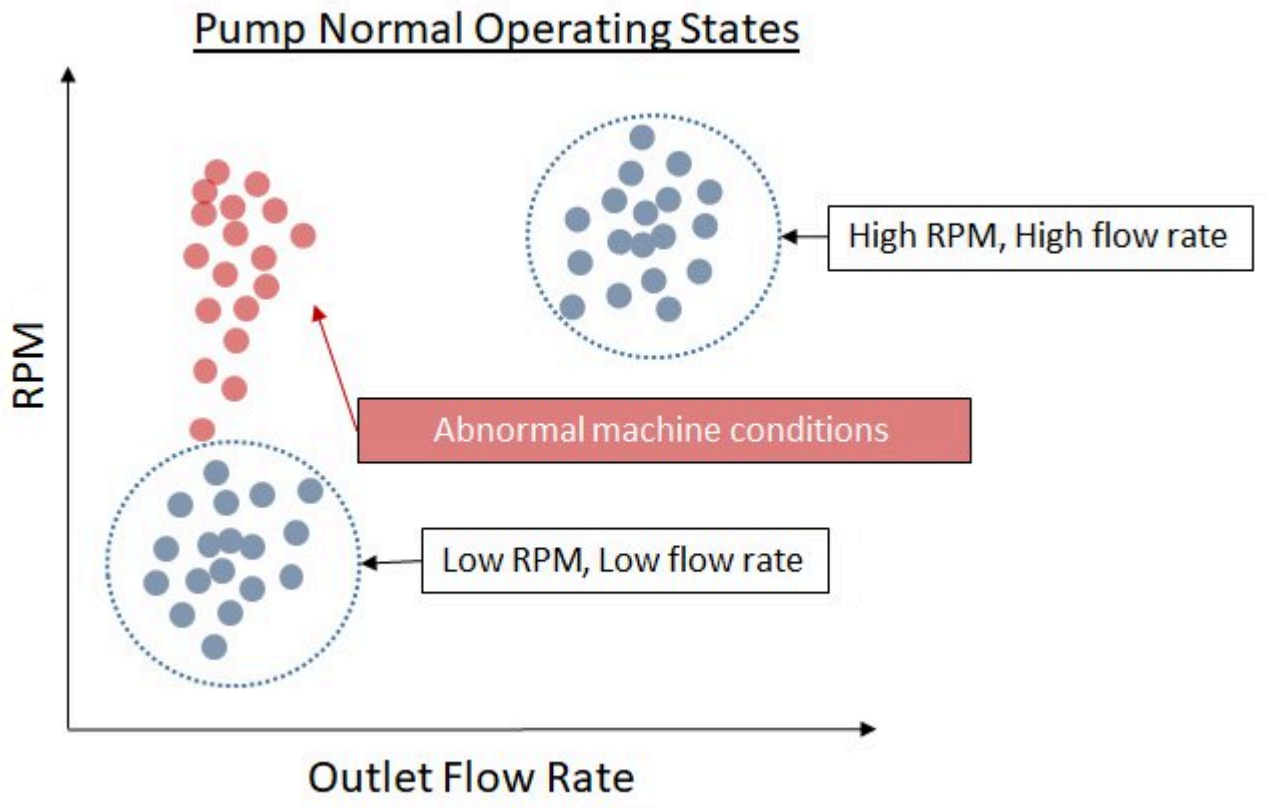
These graphs show the expected relationship between RPM and flow rate: as the pump rotates faster, the fluid flows faster. The graphs show two operating modes: one with low RPM and a low flow rate, and a second mode with high RPM and high flow rate. In this case, Amazon Lookout for Equipment wants to 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

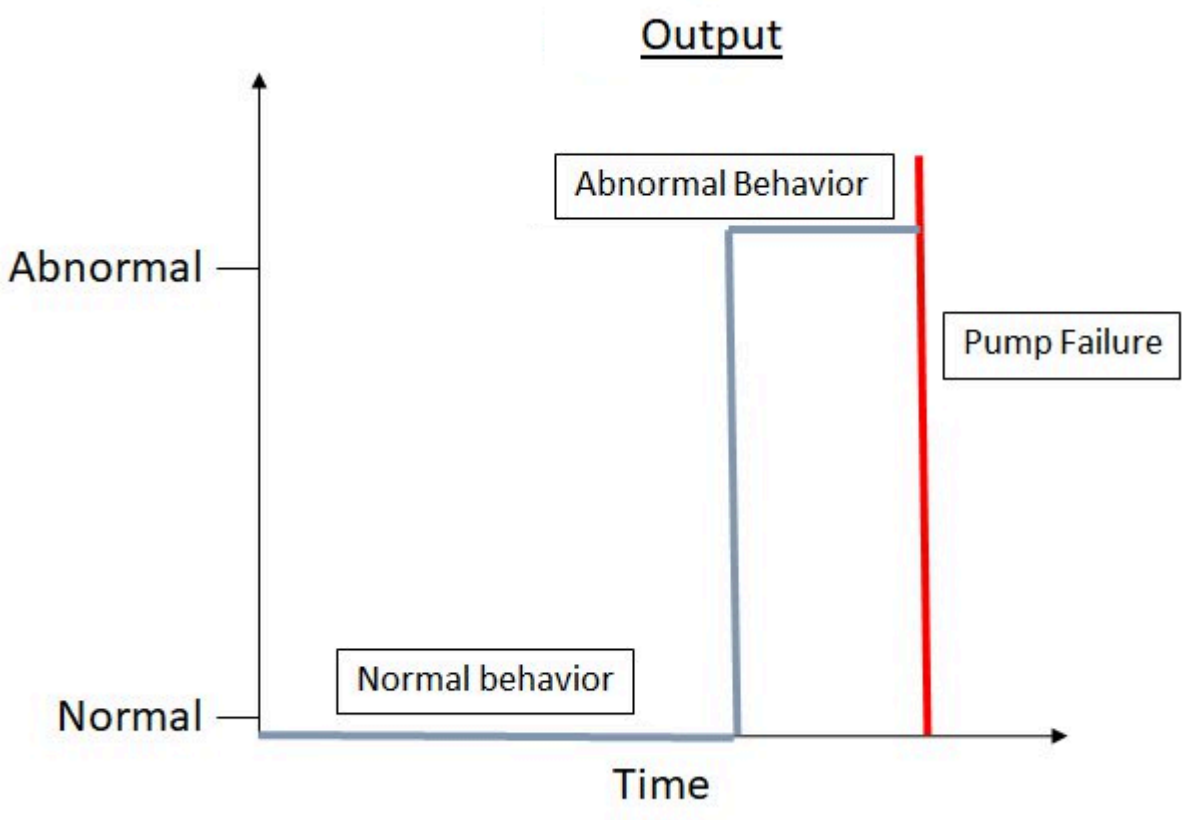
data will continue to fall into one of these two operating modes. However, if the pump starts to have problems, this relationship might not hold true.

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.

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 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
Training and evaluation	
Maximum number of models per account	15
Maximum number of pending models per account	5

Description	Quota
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
Minimum timespan of training data	14 days
Inference	
Maximum number of inference schedulers per model	1
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

Description	Quota
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
Labels	
Maximum number of label group per account	15
Maximum number of labels per label group	3000

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.
- 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.

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 service users should access. You must then submit requests to your IAM administrator to change the permissions of your service users. Review the information on this page to understand the basic concepts of IAM. To learn more about how your company can use IAM with Amazon Lookout for Equipment, see AWS Identity and Access Management for Amazon Lookout for Equipment.

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.

Authenticating with identities

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

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

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

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

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

AWS account root user

When you create an AWS account, you begin with one sign-in identity that has complete access to all AWS services and resources in the account. This identity is called the AWS account *root user* and is accessed by signing in with the email address and password that you used to create the account. We strongly recommend that you don't use the root user for your everyday tasks. Safeguard your

root user credentials and use them to perform the tasks that only the root user can perform. For the complete list of tasks that require you to sign in as the root user, see Tasks that require root user credentials in the *IAM User Guide*.

Federated identity

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

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

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

IAM users and groups

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

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

Users are different from roles. A user is uniquely associated with one person or application, but a role is intended to be assumable by anyone who needs it. Users have permanent long-term credentials, but roles provide temporary credentials. To learn more, see 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:

- **Federated user access** – To assign permissions to a federated identity, you create a role and define permissions for the role. When a federated identity authenticates, the identity is associated with the role and is granted the permissions that are defined by the role. For information about roles for federation, see Creating a role for a third-party Identity Provider in the *IAM User Guide*. If you use IAM Identity Center, you configure a permission set. To control what your identities can access after they authenticate, IAM Identity Center correlates the permission set to a role in IAM. For information about permissions sets, see Permission sets in the *AWS IAM Identity Center User Guide*.
- **Temporary IAM user permissions** – An IAM user or role can assume an IAM role to temporarily take on different permissions for a specific task.
- **Cross-account access** – You can use an IAM role to allow someone (a trusted principal) in a different account to access resources in your account. Roles are the primary way to grant cross-account access. However, with some AWS services, you can attach a policy directly to a resource (instead of using a role as a proxy). To learn the difference between roles and resource-based policies for cross-account access, see Cross account resource access in IAM in the *IAM User Guide*.
- **Cross-service access** – Some AWS services use features in other AWS services. For example, when you make a call in a service, it's common for that service to run applications in Amazon EC2 or store objects in Amazon S3. A service might do this using the calling principal's permissions, using a service role, or using a service-linked role.
  - **Forward access sessions (FAS)** – When you use an IAM user or role to perform actions in AWS, you are considered a principal. When you use some services, you might perform an action that then initiates another action in a different service. FAS uses the permissions of the principal calling an AWS service, combined with the requesting AWS service to make requests to downstream services. FAS requests are only made when a service receives a request that requires interactions with other AWS services or resources to complete. In this case, you must have permissions to perform both actions. For policy details when making FAS requests, see Forward access sessions.

- **Service role** – A service role is an IAM role that a service assumes to perform actions on your behalf. An IAM administrator can create, modify, and delete a service role from within IAM. For more information, see 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 AWS account and are owned by the service. An IAM administrator can view, but not edit the permissions for service-linked roles.

- **Applications running on Amazon EC2** – You can use an IAM role to manage temporary credentials for applications that are running on an EC2 instance and making AWS CLI or AWS API requests. This is preferable to storing access keys within the EC2 instance. To assign an AWS role to an EC2 instance and make it available to all of its applications, you create an instance profile that is attached to the instance. An instance profile contains the role and enables programs that are running on the EC2 instance to get temporary credentials. For more information, see 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 AWS identities or resources. A policy is an object in AWS that, when associated with an identity or resource, defines their permissions. AWS evaluates these policies when a principal (user, root user, or role session) makes a request. Permissions in the policies determine whether the request is allowed or denied. Most policies are stored in AWS as JSON documents. For more information about the structure and contents of JSON policy documents, see Overview of JSON policies in the *IAM User Guide*.

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

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

IAM policies define permissions for an action regardless of the method that you use to perform the operation. For example, suppose that you have a policy that allows the `iam:GetRole` action. A

user with that policy can get role information from the AWS Management Console, the AWS CLI, or the AWS API.

Identity-based policies

Identity-based policies are JSON permissions policy documents that you can attach to an identity, such as an IAM user, group of users, or role. These policies control what actions users and roles can perform, on which resources, and under what conditions. To learn how to create an identity-based policy, see 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 an entity's identity-based policies and its permissions boundaries. Resource-based policies that specify the user or role in the `Principal` field are not limited by the permissions boundary. An explicit deny in any of these policies overrides the allow. For more information about permissions boundaries, see Permissions boundaries for IAM entities in the *IAM User Guide*.
- **Service control policies (SCPs)** – SCPs are JSON policies that specify the maximum permissions for an organization or organizational unit (OU) in AWS Organizations. AWS Organizations is a service for grouping and centrally managing multiple AWS accounts that your business owns. If you enable all features in an organization, then you can apply service control policies (SCPs) to any or all of your accounts. The SCP limits permissions for entities in member accounts, including each AWS account root user. For more information about Organizations and SCPs, see 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.

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.

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

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

Service roles for 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, users and roles don't have permission to create or modify Amazon Lookout for Equipment resources. They also can't perform tasks by using the AWS Management Console, AWS Command Line Interface (AWS CLI), or AWS API. To grant users permission to perform actions on the resources that they need, an IAM administrator can create IAM policies. The administrator can then add the IAM policies to roles, and users can assume the roles.

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

For details about actions and resource types defined by Lookout for Equipment, including the format of the ARNs for each of the resource types, see in the *Service Authorization Reference*.

Topics

- Policy best practices
- Using the Lookout for Equipment console
- Allow users to view their own permissions

- Accessing a single Lookout for Equipment dataset
- Publishing information about ingestion validation to Amazon CloudWatch Logs
- Tag-based policy examples

Policy best practices

Identity-based policies 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 with AWS managed policies and move toward least-privilege permissions** – To get started granting permissions to your users and workloads, use the *AWS managed policies* that grant permissions for many common use cases. They are available in your AWS account. We recommend that you reduce permissions further by defining AWS customer managed policies that are specific to your use cases. For more information, see AWS managed policies or AWS managed policies for job functions in the *IAM User Guide*.
- **Apply least-privilege permissions** – When you set permissions with IAM policies, grant only the permissions required to perform a task. You do this by defining the actions that can be taken on specific resources under specific conditions, also known as *least-privilege permissions*. For more information about using IAM to apply permissions, see Policies and permissions in IAM in the *IAM User Guide*.
- **Use conditions in IAM policies to further restrict access** – You can add a condition to your policies to limit access to actions and resources. For example, you can write a policy condition to specify that all requests must be sent using SSL. You can also use conditions to grant access to service actions if they are used through a specific AWS service, such as AWS CloudFormation. For more information, see IAM JSON policy elements: Condition in the *IAM User Guide*.
- **Use IAM Access Analyzer to validate your IAM policies to ensure secure and functional permissions** – IAM Access Analyzer validates new and existing policies so that the policies adhere to the IAM policy language (JSON) and IAM best practices. IAM Access Analyzer provides more than 100 policy checks and actionable recommendations to help you author secure and functional policies. For more information, see IAM Access Analyzer policy validation in the *IAM User Guide*.
- **Require multi-factor authentication (MFA)** – If you have a scenario that requires IAM users or a root user in your AWS account, turn on MFA for additional security. To require MFA when API operations are called, add MFA conditions to your policies. For more information, see Configuring MFA-protected API access in the *IAM User Guide*.

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

Using the 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 (users or roles) with that policy.

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

To ensure that users and roles can still use the 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}*"
        }
    ]
}
```

Publishing information about ingestion validation to Amazon CloudWatch Logs

To enable Amazon CloudWatch Logs, do one of the following:

- When creating a new role, check the **Enable CloudWatch Logs** box on the console. For more information see Logging your ingestion data.
- Attach the following permissions to the `dataAccessRoleArn` submitted during ingestion:

```
{
  "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "logs:CreateLogStream",
                "logs:CreateLogGroup",
                "logs:PutLogEvents",
                "logs:DeleteLogStream"
            ],
            "Resource": [
                "arn:aws:logs:{{region}}:{{account-id}}:log-group:/aws/
lookoutequipment/ingestion:*"
            ]
        },
        {
            "Effect": "Allow",
            "Action": [
                "logs:DescribeLogGroups"
            ],
            "Resource": [
                "arn:aws:logs:{{region}}:{{account-id}}:log-group:*"
            ]
        }
    ]
}
```

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:Describe*",
                "lookoutequipment:List*"
            ],
```

```
            "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
AmazonLookoutEquipmentReadOnlyAccess	Lookout for Equipment modified the policy to allow	November 4, 2022

Change	Description	Date
	all Describe actions and all List actions.	
AmazonLookoutEquipmentReadOnlyAccess – Update to an existing policy	Lookout for Equipment modified the policy so as to allow all list and describe APIs.	October 26, 2022
AmazonLookoutEquipmentReadOnlyAccess – Update to an existing policy	Lookout for Equipment modified the policy so as to enable you to list sensor statistics.	June 22, 2022
AmazonLookoutEquipmentFullAccess – 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
AmazonLookoutEquipmentFullAccess – Update to an existing policy	Lookout for Equipment modified the policy so as to only apply the kms:ViaService condition to DescribeKey and CreateGrant.	Oct 29, 2021
AmazonLookoutEquipmentReadOnlyAccess – New policy	Lookout for Equipment added a new policy to allow read only access for all Lookout for Equipment resources.	May 05, 2021

Change	Description	Date
AmazonLookoutEquipmentFullAccess – 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
- I am not authorized to perform iam:PassRole
- I want to allow people outside of my AWS account to access my Lookout for Equipment resources

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

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

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

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

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

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

I am not authorized to perform iam:PassRole

If you receive an error that you're not authorized to perform the `iam:PassRole` action, your policies must be updated to allow you to pass a role to 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 that are granted by a service role. Mary does not have permissions to pass the role to the service.

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

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

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

I want to allow people outside of my AWS account to access my 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.
- To learn how to provide access to your resources across AWS accounts that you own, see Providing access to an IAM user in another AWS account that you own in the *IAM User Guide*.
- To learn how to provide access to your resources to third-party AWS accounts, see Providing access to AWS accounts owned by third parties in the *IAM User Guide*.
- To learn how to provide access through identity federation, see Providing access to externally authenticated users (identity federation) in the *IAM User Guide*.
- To learn the difference between using roles and resource-based policies for cross-account access, see Cross account resource access in IAM in the *IAM User Guide*.

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.

Monitoring Amazon Lookout for Equipment

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

- *Amazon CloudWatch* monitors your AWS resources and the applications you run on AWS in real time. You can collect and track metrics, create customized dashboards, and set alarms that notify you or take actions when a specified metric reaches a threshold that you specify. For example, you can have CloudWatch track CPU usage or other metrics of your Amazon EC2 instances and automatically launch new instances when needed. For more information, see the Amazon CloudWatch User Guide.
- *Amazon CloudWatch Logs* enables you to monitor, store, and access your log files from Amazon EC2 instances, CloudTrail, and other sources. CloudWatch Logs can monitor information in the log files and notify you when certain thresholds are met. You can also archive your log data in highly durable storage. For more information, see the Amazon CloudWatch Logs User Guide.
- *AWS CloudTrail* captures API calls and related events made by or on behalf of your AWS account and delivers the log files to an Amazon S3 bucket that you specify. You can identify which users and accounts called AWS, the source IP address from which the calls were made, and when the calls occurred. For more information, see the AWS CloudTrail User Guide.

Amazon EventBridge is a serverless event bus service that makes it easy to connect your applications with data from a variety of sources. EventBridge delivers a stream of real-time data from your own applications, Software-as-a-Service (SaaS) applications, and AWS services and routes that data to targets such as Lambda. This enables you to monitor events that happen in services, and build event-driven architectures. For more information, see the Amazon EventBridge User Guide.

Monitoring Lookout for Equipment with Amazon CloudWatch

You can monitor 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.

The Lookout for Equipment service reports the following metrics in the `AWS/lookoutequipment` namespace.

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 an invalid value for the inference. ModelName: The name of the model. InferenceSchedulerName: Name of the inference scheduler

The following dimensions are supported for the Lookout for Equipment metrics.

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.

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

Python SDK examples

Creating a schema from multiple .csv files

> **Note**
>
> If you use the console to ingest your data, Lookout for Equipment can detect your schema for you, according to the way you organize your 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.

In the following 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.

For example, `"ComponentName": "Sensor2"` could acccess `s3://DOC-EXAMPLE-BUCKET/`*`AssetName`*`/`*`Sensor2`*`/`*`Sensor2`*`.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 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
    signature_version = 'v4'
)
```

```
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"
                }
            ]
        }
    ]
}
```

Creating a schema from a single .csv file

> **Note**
>
> If you use the console to ingest your data, Lookout for Equipment can detect your schema for you, according to the way you organize your files.

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

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 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
    signature_version = 'v4'
)

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)
```

Adding a dataset to your project

> **Note**
>
> You can also add a dataset to your project using the console.

Use the following AWS SDK for Python (Boto3) example code to tell Lookout for Equipment to ingest your dataset.

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


config = Config(
region_name = 'Region' #Choose a valid AWS Region
 signature_version = 'v4'
 )

 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=====")
```

Viewing a model

> **Note**
>
> You can also view and evaluate a model in the console.

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 your model 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")
```

Managing your datasets

> **Note**
>
> You can also manage your datasets in the console.

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 must first run the example code in either Creating a schema from a single .csv file or Creating a schema from multiple .csv files .

```
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")
```

Labeling your data

The following example code uses the AWS SDK for Python (Boto3) to provide labels for your 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)

LABEL_GROUP_NAME = "[Replace it with your label group name]"
create_label_group_request = {
    "LabelGroupName": LABEL_GROUP_NAME,
    "FaultCodes": ["[Your Fault Code1]", "[Your Fault Code2]"]
}

# CREATE A LABEL GROUP
create_label_group_response =
 lookoutequipment_client.create_label_group(**create_label_group_request)
pp = pprint.PrettyPrinter(depth=4)
print("=====Create Label Group Response=====\n")
pp.pprint(create_label_group_response)
print("\n=====End of Response=====")
```

```
# CREATE A LABEL
# You can create more labels as anomalies are identified before and after creating a
 model.
create_label_request = {
    "LabelGroupName": LABEL_GROUP_NAME,
    "Rating": "ANOMALY", # ANOMALY, NORMAL or NEUTRAL
    "StartTime": datetime(2022, 1, 1, 0, 0, 0),
    "EndTime": datetime(2022, 1, 1, 1, 0, 0),
    "FaultCode": "[Your Fault Code1]", # Must be defined in label group fault codes.
 Optional.
    "Equipment": "[replace with your equipment identifier]", # Optional
    "Notes": "[replace with your notes]", # Optional
}

create_label_response = lookoutequipment_client.create_label(**create_label_request)
print("=====Create Label Response=====\n")
pp.pprint(create_label_response)
print("\n=====End of Response=====")
```

Managing your labels

You can use code like the following example code to manage your labels.

```
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)

LABEL_GROUP_NAME = "[Replace it with your label group name]"

# LIST LABEL GROUPS
list_label_groups_request = {}
```

```
list_label_groups_response =
 lookoutequipment_client.list_label_groups(**list_label_groups_request)
pp = pprint.PrettyPrinter(depth=4)
print("=====List Label Groups Response=====\n")
pp.pprint(list_label_groups_response)
print("\n=====End of Response=====")


# DESCRIBE A LABEL GROUP
describe_label_group_request = {"LabelGroupName": LABEL_GROUP_NAME}
describe_label_group_response =
 lookoutequipment_client.describe_label_group(**describe_label_group_request)
print("=====Describe Label Group Response=====\n")
pp.pprint(describe_label_group_response)
print("\n=====End of Response=====")


# UPDATE A LABEL GROUP
update_label_group_request = {"LabelGroupName": LABEL_GROUP_NAME, "FaultCodes": ["[Your
 Fault Code1]", "[Your Fault Code3]"]}
update_label_group_response =
 lookoutequipment_client.update_label_group(**update_label_group_request)
print("=====Update Label Group Response=====\n")
pp.pprint(update_label_group_response)
print("\n=====End of Response=====")


# LIST LABELS
list_labels = {
    "LabelGroupName": LABEL_GROUP_NAME,
}

list_labels_response = lookoutequipment_client.list_labels(**list_labels)
print("=====Create Label Response=====\n")
pp.pprint(list_labels_response)
print("\n=====End of Response=====")


# DESCRIBE A LABEL
describe_label_request = {
    "LabelGroupName": LABEL_GROUP_NAME,
    "LabelId": "[Replace with Label Id]",
}
```

```
describe_label_response =
 lookoutequipment_client.describe_label(**describe_label_request)
print("=====Describe Label Response=====\n")
pp.pprint(describe_label_response)
print("\n=====End of Response=====")


# DELETE A LABEL
delete_label_request = {
    "LabelGroupName": LABEL_GROUP_NAME,
    "LabelId": "[Replace with Label Id]",
}

delete_label_response = lookoutequipment_client.delete_label(**delete_label_request)
print("=====Delete Label Response=====\n")
pp.pprint(delete_label_response)
print("\n=====End of Response=====")


# DELETE A LABEL GROUP
delete_label_group_request = {
    "LabelGroupName": LABEL_GROUP_NAME
}

delete_label_group_response =
 lookoutequipment_client.delete_label_group(**delete_label_group_request)
print("=====Delete Label Group Response=====\n")
pp.pprint(delete_label_group_response)
print("\n=====End of Response=====")
```

Training a model

> **Note**
>
> You can also train a model with the console.

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 in label group (See Labeling APIs example)
LABEL_GROUP_NAME = 'your-label-group'
# If you have label data in S3, 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_GROUP_NAME is not None:
    labels_input_config = dict()
    labels_input_config['LabelGroupName'] = LABEL_GROUP_NAME
    create_model_request['LabelsInputConfiguration'] = labels_input_config
elif 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=====")
```

Schedule inference

> **Note**
>
> You can also schedule inference with the console.

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=====")
```

API Reference

This section contains the API Reference documentation.

Actions

The following actions are supported:

- CreateDataset
- CreateInferenceScheduler
- CreateLabel
- CreateLabelGroup
- CreateModel
- CreateRetrainingScheduler
- DeleteDataset
- DeleteInferenceScheduler
- DeleteLabel
- DeleteLabelGroup
- DeleteModel
- DeleteResourcePolicy
- DeleteRetrainingScheduler
- DescribeDataIngestionJob
- DescribeDataset
- DescribeInferenceScheduler
- DescribeLabel
- DescribeLabelGroup
- DescribeModel
- DescribeModelVersion
- DescribeResourcePolicy
- DescribeRetrainingScheduler
- ImportDataset

- ImportModelVersion
- ListDataIngestionJobs
- ListDatasets
- ListInferenceEvents
- ListInferenceExecutions
- ListInferenceSchedulers
- ListLabelGroups
- ListLabels
- ListModels
- ListModelVersions
- ListRetrainingSchedulers
- ListSensorStatistics
- ListTagsForResource
- PutResourcePolicy
- StartDataIngestionJob
- StartInferenceScheduler
- StartRetrainingScheduler
- StopInferenceScheduler
- StopRetrainingScheduler
- TagResource
- UntagResource
- UpdateActiveModelVersion
- UpdateInferenceScheduler
- UpdateLabelGroup
- UpdateModel
- UpdateRetrainingScheduler

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. For example, 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

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

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

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

Type: DatasetSchema object

Required: No

ServerSideKmsKeyId

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

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

Type: Array of Tag 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

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\/[0-9a-zA-Z_-]{1,200}\/.+`

DatasetName

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

Indicates the status of the `CreateDataset` operation.

Type: String

Valid Values: `CREATED | INGESTION_IN_PROGRESS | ACTIVE | IMPORT_IN_PROGRESS`

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 v2
- AWS SDK for Java V2

- AWS SDK for JavaScript V3
- 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

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

The interval (in minutes) of planned delay at the start of each inference segment. For example, if inference is set to run every ten minutes, the delay is set to five minutes and the time is 09:08. The inference scheduler will wake up at the configured interval (which, without a delay configured, would be 09:10) plus the additional five minute delay time (so 09:15) to check your Amazon S3 bucket. The delay provides a buffer for you to upload data at the same frequency, so that you don't have to stop and restart the scheduler when uploading new data.

For more information, see Understanding the inference process.

Type: Long

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

Required: No

DataInputConfiguration

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

Type: InferenceInputConfiguration object

Required: Yes

DataOutputConfiguration

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

Type: InferenceOutputConfiguration object

Required: Yes

DataUploadFrequency

How often data is uploaded to the source Amazon 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 runs inference on your data.

For more information, see Understanding the inference process.

Type: String

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

Required: Yes

InferenceSchedulerName

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

The name of the previously trained machine learning 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

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

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

Any tags associated with the inference scheduler.

Type: Array of Tag objects

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

Required: No

Response Syntax

```
{
   "InferenceSchedulerArn": "string",
```

```
   "InferenceSchedulerName": "string",
   "ModelQuality": "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

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

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}$`

ModelQuality

Provides a quality assessment for a model that uses labels. If Lookout for Equipment determines that the model quality is poor based on training metrics, the value is `POOR_QUALITY_DETECTED`. Otherwise, the value is `QUALITY_THRESHOLD_MET`.

If the model is unlabeled, the model quality can't be assessed and the value of `ModelQuality` is `CANNOT_DETERMINE_QUALITY`. In this situation, you can get a model quality assessment by adding labels to the input dataset and retraining the model.

For information about using labels with your models, see Understanding labeling.

For information about improving the quality of a model, see Best practices with Amazon Lookout for Equipment.

Type: String

Valid Values: `QUALITY_THRESHOLD_MET | CANNOT_DETERMINE_QUALITY | POOR_QUALITY_DETECTED`

Status

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 v2
- AWS SDK for Java V2
- AWS SDK for JavaScript V3
- AWS SDK for PHP V3
- AWS SDK for Python
- AWS SDK for Ruby V3

CreateLabel

Creates a label for an event.

Request Syntax

```
{
   "ClientToken": "string",
   "EndTime": number,
   "Equipment": "string",
   "FaultCode": "string",
   "LabelGroupName": "string",
   "Notes": "string",
   "Rating": "string",
   "StartTime": number
}
```

Request Parameters

The request accepts the following data in JSON format.

ClientToken

A unique identifier for the request to create a label. If you do not set the client request token, Lookout for Equipment generates one.

Type: String

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

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

Required: Yes

EndTime

The end time of the labeled event.

Type: Timestamp

Required: Yes

Equipment

Indicates that a label pertains to a particular piece of equipment.

Data in this field will be retained for service usage. Follow best practices for the security of your data.

Type: String

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

Pattern: `[\P{M}\p{M}]{1,200}`

Required: No

FaultCode

Provides additional information about the label. The fault code must be defined in the FaultCodes attribute of the label group.

Data in this field will be retained for service usage. Follow best practices for the security of your data.

Type: String

Length Constraints: Minimum length of 1. Maximum length of 100.

Pattern: `[\P{M}\p{M}]{1,100}`

Required: No

LabelGroupName

The name of a group of labels.

Data in this field will be retained for service usage. Follow best practices for the security of your data.

Type: String

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

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

Required: Yes

Notes

Metadata providing additional information about the label.

Data in this field will be retained for service usage. Follow best practices for the security of your data.

Type: String

Length Constraints: Minimum length of 1. Maximum length of 2560.

Pattern: `[\P{M}\p{M}]{1,2560}`

Required: No

Rating

Indicates whether a labeled event represents an anomaly.

Type: String

Valid Values: `ANOMALY | NO_ANOMALY | NEUTRAL`

Required: Yes

StartTime

The start time of the labeled event.

Type: Timestamp

Required: Yes

Response Syntax

```
{
   "LabelId": "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.

LabelId

The ID of the label that you have created.

Type: String

Length Constraints: Maximum length of 32.

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

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 v2
- AWS SDK for Java V2
- AWS SDK for JavaScript V3
- AWS SDK for PHP V3
- AWS SDK for Python
- AWS SDK for Ruby V3

CreateLabelGroup

Creates a group of labels.

Request Syntax

```
{
   "ClientToken": "string",
   "FaultCodes": [ "string" ],
   "LabelGroupName": "string",
   "Tags": [
      {
         "Key": "string",
         "Value": "string"
      }
   ]
}
```

Request Parameters

The request accepts the following data in JSON format.

ClientToken

A unique identifier for the request to create a label group. If you do not set the client request token, Lookout for Equipment generates one.

Type: String

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

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

Required: Yes

FaultCodes

The acceptable fault codes (indicating the type of anomaly associated with the label) that can be used with this label group.

Data in this field will be retained for service usage. Follow best practices for the security of your data.

Type: Array of strings

Array Members: Minimum number of 0 items. Maximum number of 50 items.

Length Constraints: Minimum length of 1. Maximum length of 100.

Pattern: `[\P{M}\p{M}]{1,100}`

Required: No

LabelGroupName

Names a group of labels.

Data in this field will be retained for service usage. Follow best practices for the security of your data.

Type: String

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

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

Required: Yes

Tags

Tags that provide metadata about the label group you are creating.

Data in this field will be retained for service usage. Follow best practices for the security of your data.

Type: Array of Tag objects

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

Required: No

Response Syntax

```
{
   "LabelGroupArn": "string",
   "LabelGroupName": "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.

LabelGroupArn

The Amazon Resource Name (ARN) of the label group that you have created.

Type: String

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

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

LabelGroupName

The name of the label group that you have created. Data in this field will be retained for service usage. Follow best practices for the security of your data.

Type: String

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

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

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 v2
- AWS SDK for Java V2
- AWS SDK for JavaScript V3
- AWS SDK for PHP V3
- AWS SDK for Python
- AWS SDK for Ruby V3

CreateModel

Creates a machine learning 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": {
      "LabelGroupName": "string",
      "S3InputConfiguration": {
         "Bucket": "string",
         "Prefix": "string"
      }
   },
   "ModelDiagnosticsOutputConfiguration": {
      "KmsKeyId": "string",
      "S3OutputConfiguration": {
         "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

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

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 object

Required: No

DatasetName

The name of the dataset for the machine learning 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

The data schema for the machine learning model being created.

Type: DatasetSchema object

Required: No

EvaluationDataEndTime

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

Type: Timestamp

Required: No

EvaluationDataStartTime

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

Type: Timestamp

Required: No

LabelsInputConfiguration

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

Type: LabelsInputConfiguration object

Required: No

ModelDiagnosticsOutputConfiguration

The Amazon S3 location where you want Amazon Lookout for Equipment to save the pointwise model diagnostics. You must also specify the `RoleArn` request parameter.

Type: ModelDiagnosticsOutputConfiguration object

Required: No

ModelName

The name for the machine learning 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

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

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

Type: String

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

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

Required: No

ServerSideKmsKeyId

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

Any tags associated with the machine learning model being created.

Type: Array of Tag objects

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

Required: No

TrainingDataEndTime

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

Type: Timestamp

Required: No

TrainingDataStartTime

Indicates the time reference in the dataset that should be used to begin the subset of training data for the machine learning 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

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

Indicates the status of the `CreateModel` operation.

Type: String

Valid Values: `IN_PROGRESS | SUCCESS | FAILED | IMPORT_IN_PROGRESS`

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 v2
- AWS SDK for Java V2
- AWS SDK for JavaScript V3

- AWS SDK for PHP V3
- AWS SDK for Python
- AWS SDK for Ruby V3

CreateRetrainingScheduler

Creates a retraining scheduler on the specified model.

Request Syntax

```
{
   "ClientToken": "string",
   "LookbackWindow": "string",
   "ModelName": "string",
   "PromoteMode": "string",
   "RetrainingFrequency": "string",
   "RetrainingStartDate": number
}
```

Request Parameters

The request accepts the following data in JSON format.

ClientToken

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

LookbackWindow

The number of past days of data that will be used for retraining.

Type: String

Pattern: `^P180D$|^P360D$|^P540D$|^P720D$`

Required: Yes

ModelName

The name of the model to add the retraining scheduler to.

Type: String

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

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

Required: Yes

PromoteMode

Indicates how the service will use new models. In `MANAGED` mode, new models will automatically be used for inference if they have better performance than the current model. In `MANUAL` mode, the new models will not be used until they are manually activated.

Type: String

Valid Values: `MANAGED | MANUAL`

Required: No

RetrainingFrequency

This parameter uses the ISO 8601 standard to set the frequency at which you want retraining to occur in terms of Years, Months, and/or Days (note: other parameters like Time are not currently supported). The minimum value is 30 days (P30D) and the maximum value is 1 year (P1Y). For example, the following values are valid:

- P3M15D – Every 3 months and 15 days
- P2M – Every 2 months
- P150D – Every 150 days

Type: String

Length Constraints: Minimum length of 1. Maximum length of 10.

Pattern: `^P(\dY)?(\d{1,2}M)?(\d{1,3}D)?$`

Required: Yes

RetrainingStartDate

The start date for the retraining scheduler. Lookout for Equipment truncates the time you provide to the nearest UTC day.

Type: Timestamp

Required: No

Response Syntax

```
{
   "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.

ModelArn

The ARN of the model that you added the retraining scheduler to.

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

The name of the model that you added the retraining scheduler to.

Type: String

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

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

Status

The status of the retraining 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 v2
- AWS SDK for Java V2
- AWS SDK for JavaScript V3
- 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

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

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 v2
- AWS SDK for Java V2
- AWS SDK for JavaScript V3

- AWS SDK for PHP V3
- AWS SDK for Python
- AWS SDK for Ruby V3

DeleteInferenceScheduler

Deletes an inference scheduler that has been set up. Prior inference results will not be deleted.

Request Syntax

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

Request Parameters

The request accepts the following data in JSON format.

InferenceSchedulerName

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 v2
- AWS SDK for Java V2
- AWS SDK for JavaScript V3
- AWS SDK for PHP V3
- AWS SDK for Python
- AWS SDK for Ruby V3

DeleteLabel

Deletes a label.

Request Syntax

```
{
   "LabelGroupName": "string",
   "LabelId": "string"
}
```

Request Parameters

The request accepts the following data in JSON format.

LabelGroupName

The name of the label group that contains the label that you want to delete. Data in this field will be retained for service usage. Follow best practices for the security of your data.

Type: String

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

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

Required: Yes

LabelId

The ID of the label that you want to delete.

Type: String

Length Constraints: Maximum length of 32.

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

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 v2
- AWS SDK for Java V2
- AWS SDK for JavaScript V3
- AWS SDK for PHP V3
- AWS SDK for Python
- AWS SDK for Ruby V3

DeleteLabelGroup

Deletes a group of labels.

Request Syntax

```
{
   "LabelGroupName": "string"
}
```

Request Parameters

The request accepts the following data in JSON format.

LabelGroupName

The name of the label group that you want to delete. Data in this field will be retained for service usage. Follow best practices for the security of your data.

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 v2
- AWS SDK for Java V2
- AWS SDK for JavaScript V3
- AWS SDK for PHP V3
- AWS SDK for Python
- AWS SDK for Ruby V3

DeleteModel

Deletes a machine learning 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

The name of the machine learning 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

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 v2
- AWS SDK for Java V2
- AWS SDK for JavaScript V3
- AWS SDK for PHP V3
- AWS SDK for Python
- AWS SDK for Ruby V3

DeleteResourcePolicy

Deletes the resource policy attached to the resource.

Request Syntax

```
{
   "ResourceArn": "string"
}
```

Request Parameters

The request accepts the following data in JSON format.

ResourceArn

The Amazon Resource Name (ARN) of the resource for which the resource policy should be deleted.

Type: String

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

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

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 v2
- AWS SDK for Java V2
- AWS SDK for JavaScript V3
- AWS SDK for PHP V3
- AWS SDK for Python
- AWS SDK for Ruby V3

DeleteRetrainingScheduler

Deletes a retraining scheduler from a model. The retraining scheduler must be in the STOPPED status.

Request Syntax

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

Request Parameters

The request accepts the following data in JSON format.

ModelName

The name of the model whose retraining scheduler you want to delete.

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 v2
- AWS SDK for Java V2
- AWS SDK for JavaScript V3
- 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, and status.

Request Syntax

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

Request Parameters

The request accepts the following data in JSON format.

JobId

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,
   "DataEndTime": number,
   "DataQualitySummary": {
      "DuplicateTimestamps": {
         "TotalNumberOfDuplicateTimestamps": number
      },
      "InsufficientSensorData": {
         "MissingCompleteSensorData": {
            "AffectedSensorCount": number
         },
         "SensorsWithShortDateRange": {
            "AffectedSensorCount": number
```

```
         }
      },
      "InvalidSensorData": {
         "AffectedSensorCount": number,
         "TotalNumberOfInvalidValues": number
      },
      "MissingSensorData": {
         "AffectedSensorCount": number,
         "TotalNumberOfMissingValues": number
      },
      "UnsupportedTimestamps": {
         "TotalNumberOfUnsupportedTimestamps": number
      }
   },
   "DatasetArn": "string",
   "DataStartTime": number,
   "FailedReason": "string",
   "IngestedDataSize": number,
   "IngestedFilesSummary": {
      "DiscardedFiles": [
         {
            "Bucket": "string",
            "Key": "string"
         }
      ],
      "IngestedNumberOfFiles": number,
      "TotalNumberOfFiles": number
   },
   "IngestionInputConfiguration": {
      "S3InputConfiguration": {
         "Bucket": "string",
         "KeyPattern": "string",
         "Prefix": "string"
      }
   },
   "JobId": "string",
   "RoleArn": "string",
   "SourceDatasetArn": "string",
   "Status": "string",
   "StatusDetail": "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

The time at which the data ingestion job was created.

Type: Timestamp

DataEndTime

Indicates the latest timestamp corresponding to data that was successfully ingested during this specific ingestion job.

Type: Timestamp

DataQualitySummary

Gives statistics about a completed ingestion job. These statistics primarily relate to quantifying incorrect data such as MissingCompleteSensorData, MissingSensorData, UnsupportedDateFormats, InsufficientSensorData, and DuplicateTimeStamps.

Type: DataQualitySummary object

DatasetArn

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\/[0-9a-zA-Z_-]{1,200}\/.+`

DataStartTime

Indicates the earliest timestamp corresponding to data that was successfully ingested during this specific ingestion job.

Type: Timestamp

FailedReason

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}`

IngestedDataSize

Indicates the size of the ingested dataset.

Type: Long

Valid Range: Minimum value of 0.

IngestedFilesSummary

Gives statistics about how many files have been ingested, and which files have not been ingested, for a particular ingestion job.

Type: IngestedFilesSummary object

IngestionInputConfiguration

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

Type: IngestionInputConfiguration object

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}`

RoleArn

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/.+`

SourceDatasetArn

The Amazon Resource Name (ARN) of the source dataset from which the data used for the data ingestion job was imported from.

Type: String

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

Pattern: `arn:aws(-[^:]+)?:lookoutequipment:[a-zA-Z0-9\-]*:[0-9]{12}:dataset\/[0-9a-zA-Z_-]{1,200}\/.+`

Status

Indicates the status of the `DataIngestionJob` operation.

Type: String

Valid Values: `IN_PROGRESS | SUCCESS | FAILED | IMPORT_IN_PROGRESS`

StatusDetail

Provides details about status of the ingestion job that is currently in progress.

Type: String

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

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

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 v2
- AWS SDK for Java V2
- AWS SDK for JavaScript V3
- AWS SDK for PHP V3
- AWS SDK for Python
- AWS SDK for Ruby V3

DescribeDataset

Provides a JSON description of the data 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

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,
   "DataEndTime": number,
   "DataQualitySummary": {
      "DuplicateTimestamps": {
         "TotalNumberOfDuplicateTimestamps": number
      },
      "InsufficientSensorData": {
         "MissingCompleteSensorData": {
            "AffectedSensorCount": number
         },
         "SensorsWithShortDateRange": {
            "AffectedSensorCount": number
```

```
         }
      },
      "InvalidSensorData": {
         "AffectedSensorCount": number,
         "TotalNumberOfInvalidValues": number
      },
      "MissingSensorData": {
         "AffectedSensorCount": number,
         "TotalNumberOfMissingValues": number
      },
      "UnsupportedTimestamps": {
         "TotalNumberOfUnsupportedTimestamps": number
      }
   },
   "DatasetArn": "string",
   "DatasetName": "string",
   "DataStartTime": number,
   "IngestedFilesSummary": {
      "DiscardedFiles": [
         {
            "Bucket": "string",
            "Key": "string"
         }
      ],
      "IngestedNumberOfFiles": number,
      "TotalNumberOfFiles": number
   },
   "IngestionInputConfiguration": {
      "S3InputConfiguration": {
         "Bucket": "string",
         "KeyPattern": "string",
         "Prefix": "string"
      }
   },
   "LastUpdatedAt": number,
   "RoleArn": "string",
   "Schema": "string",
   "ServerSideKmsKeyId": "string",
   "SourceDatasetArn": "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

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

Type: Timestamp

DataEndTime

Indicates the latest timestamp corresponding to data that was successfully ingested during the most recent ingestion of this particular dataset.

Type: Timestamp

DataQualitySummary

Gives statistics associated with the given dataset for the latest successful associated ingestion job id. These statistics primarily relate to quantifying incorrect data such as MissingCompleteSensorData, MissingSensorData, UnsupportedDateFormats, InsufficientSensorData, and DuplicateTimeStamps.

Type: DataQualitySummary object

DatasetArn

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\/[0-9a-zA-Z_-]{1,200}\/.+`

DatasetName

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}$`

DataStartTime

Indicates the earliest timestamp corresponding to data that was successfully ingested during the most recent ingestion of this particular dataset.

Type: Timestamp

IngestedFilesSummary

IngestedFilesSummary associated with the given dataset for the latest successful associated ingestion job id.

Type: IngestedFilesSummary object

IngestionInputConfiguration

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

Type: IngestionInputConfiguration object

LastUpdatedAt

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

Type: Timestamp

RoleArn

The Amazon Resource Name (ARN) of the IAM role that you are using for this the data ingestion job.

Type: String

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

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

Schema

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

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\-\/]+`

SourceDatasetArn

The Amazon Resource Name (ARN) of the source dataset from which the current data being described was imported from.

Type: String

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

Pattern: `arn:aws(-[^:]+)?:lookoutequipment:[a-zA-Z0-9\-]*:[0-9]{12}:dataset\/[0-9a-zA-Z_-]{1,200}\/.+`

Status

Indicates the status of the dataset.

Type: String

Valid Values: `CREATED | INGESTION_IN_PROGRESS | ACTIVE | IMPORT_IN_PROGRESS`

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 v2
- AWS SDK for Java V2
- AWS SDK for JavaScript V3
- 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

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",
   "LatestInferenceResult": "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

Specifies the time at which the inference scheduler was created.

Type: Timestamp

DataDelayOffsetInMinutes

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

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

Type: InferenceInputConfiguration object

DataOutputConfiguration

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

Type: InferenceOutputConfiguration object

DataUploadFrequency

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

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

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}$`

LatestInferenceResult

Indicates whether the latest execution for the inference scheduler was Anomalous (anomalous events found) or Normal (no anomalous events found).

Type: String

Valid Values: `ANOMALOUS | NORMAL`

ModelArn

The Amazon Resource Name (ARN) of the machine learning 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

The name of the machine learning 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

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

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

Indicates the status of the inference scheduler.

Type: String

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

UpdatedAt

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 v2
- AWS SDK for Java V2
- AWS SDK for JavaScript V3
- AWS SDK for PHP V3
- AWS SDK for Python
- AWS SDK for Ruby V3

DescribeLabel

Returns the name of the label.

Request Syntax

```
{
   "LabelGroupName": "string",
   "LabelId": "string"
}
```

Request Parameters

The request accepts the following data in JSON format.

LabelGroupName

Returns the name of the group containing the label.

Type: String

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

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

Required: Yes

LabelId

Returns the ID of the label.

Type: String

Length Constraints: Maximum length of 32.

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

Required: Yes

Response Syntax

```
{
   "CreatedAt": number,
```

```
    "EndTime": number,
    "Equipment": "string",
    "FaultCode": "string",
    "LabelGroupArn": "string",
    "LabelGroupName": "string",
    "LabelId": "string",
    "Notes": "string",
    "Rating": "string",
    "StartTime": 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

The time at which the label was created.

Type: Timestamp

EndTime

The end time of the requested label.

Type: Timestamp

Equipment

Indicates that a label pertains to a particular piece of equipment.

Type: String

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

Pattern: `[\P{M}\p{M}]{1,200}`

FaultCode

Indicates the type of anomaly associated with the label.

Data in this field will be retained for service usage. Follow best practices for the security of your data.

Type: String

Length Constraints: Minimum length of 1. Maximum length of 100.

Pattern: `[\P{M}\p{M}]{1,100}`

LabelGroupArn

The Amazon Resource Name (ARN) of the requested label group.

Type: String

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

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

LabelGroupName

The name of the requested label group.

Type: String

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

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

LabelId

The ID of the requested label.

Type: String

Length Constraints: Maximum length of 32.

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

Notes

Metadata providing additional information about the label.

Data in this field will be retained for service usage. Follow best practices for the security of your data.

Type: String

Length Constraints: Minimum length of 1. Maximum length of 2560.

Pattern: `[\P{M}\p{M}]{1,2560}`

Rating

Indicates whether a labeled event represents an anomaly.

Type: String

Valid Values: `ANOMALY | NO_ANOMALY | NEUTRAL`

StartTime

The start time of the requested label.

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 v2
- AWS SDK for Java V2
- AWS SDK for JavaScript V3
- AWS SDK for PHP V3
- AWS SDK for Python
- AWS SDK for Ruby V3

DescribeLabelGroup

Returns information about the label group.

Request Syntax

```
{
   "LabelGroupName": "string"
}
```

Request Parameters

The request accepts the following data in JSON format.

LabelGroupName

Returns the name of the label group.

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,
   "FaultCodes": [ "string" ],
   "LabelGroupArn": "string",
   "LabelGroupName": "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

The time at which the label group was created.

Type: Timestamp

FaultCodes

Codes indicating the type of anomaly associated with the labels in the lagbel group.

Type: Array of strings

Array Members: Minimum number of 0 items. Maximum number of 50 items.

Length Constraints: Minimum length of 1. Maximum length of 100.

Pattern: `[\P{M}\p{M}]{1,100}`

LabelGroupArn

The Amazon Resource Name (ARN) of the label group.

Type: String

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

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

LabelGroupName

The name of the label group.

Type: String

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

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

UpdatedAt

The time at which the label group was updated.

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 v2
- AWS SDK for Java V2

- AWS SDK for JavaScript V3
- 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 machine learning 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

The name of the machine learning 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

```
{
   "AccumulatedInferenceDataEndTime": number,
   "AccumulatedInferenceDataStartTime": number,
   "ActiveModelVersion": number,
   "ActiveModelVersionArn": "string",
   "CreatedAt": number,
   "DataPreProcessingConfiguration": {
      "TargetSamplingRate": "string"
   },
   "DatasetArn": "string",
   "DatasetName": "string",
   "EvaluationDataEndTime": number,
   "EvaluationDataStartTime": number,
```

```
   "FailedReason": "string",
   "ImportJobEndTime": number,
   "ImportJobStartTime": number,
   "LabelsInputConfiguration": {
      "LabelGroupName": "string",
      "S3InputConfiguration": {
         "Bucket": "string",
         "Prefix": "string"
      }
   },
   "LastUpdatedTime": number,
   "LatestScheduledRetrainingAvailableDataInDays": number,
   "LatestScheduledRetrainingFailedReason": "string",
   "LatestScheduledRetrainingModelVersion": number,
   "LatestScheduledRetrainingStartTime": number,
   "LatestScheduledRetrainingStatus": "string",
   "ModelArn": "string",
   "ModelDiagnosticsOutputConfiguration": {
      "KmsKeyId": "string",
      "S3OutputConfiguration": {
         "Bucket": "string",
         "Prefix": "string"
      }
   },
   "ModelMetrics": "string",
   "ModelName": "string",
   "ModelQuality": "string",
   "ModelVersionActivatedAt": number,
   "NextScheduledRetrainingStartDate": number,
   "OffCondition": "string",
   "PreviousActiveModelVersion": number,
   "PreviousActiveModelVersionArn": "string",
   "PreviousModelVersionActivatedAt": number,
   "PriorModelMetrics": "string",
   "RetrainingSchedulerStatus": "string",
   "RoleArn": "string",
   "Schema": "string",
   "ServerSideKmsKeyId": "string",
   "SourceModelVersionArn": "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.

AccumulatedInferenceDataEndTime

Indicates the end time of the inference data that has been accumulated.

Type: Timestamp

AccumulatedInferenceDataStartTime

Indicates the start time of the inference data that has been accumulated.

Type: Timestamp

ActiveModelVersion

The name of the model version used by the inference schedular when running a scheduled inference execution.

Type: Long

Valid Range: Minimum value of 1.

ActiveModelVersionArn

The Amazon Resource Name (ARN) of the model version used by the inference scheduler when running a scheduled 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\/[0-9a-zA-Z_-]{1,200}\/.+\/model-version\/[0-9]{1,}$`

CreatedAt

Indicates the time and date at which the machine learning model was created.

Type: Timestamp

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*

Type: DataPreProcessingConfiguration object

DatasetArn

The Amazon Resouce Name (ARN) of the dataset used to create the machine learning 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\/[0-9a-zA-Z_-]{1,200}\/.+`

DatasetName

The name of the dataset being used by the machine learning being described.

Type: String

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

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

EvaluationDataEndTime

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

Type: Timestamp

EvaluationDataStartTime

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

Type: Timestamp

FailedReason

If the training of the machine learning 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}`

ImportJobEndTime

The date and time when the import job was completed. This field appears if the active model version was imported.

Type: Timestamp

ImportJobStartTime

The date and time when the import job was started. This field appears if the active model version was imported.

Type: Timestamp

LabelsInputConfiguration

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

Type: LabelsInputConfiguration object

LastUpdatedTime

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

Type: Timestamp

LatestScheduledRetrainingAvailableDataInDays

Indicates the number of days of data used in the most recent scheduled retraining run.

Type: Integer

LatestScheduledRetrainingFailedReason

If the model version was generated by retraining and the training 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}`

LatestScheduledRetrainingModelVersion

Indicates the most recent model version that was generated by retraining.

Type: Long

Valid Range: Minimum value of 1.

LatestScheduledRetrainingStartTime

Indicates the start time of the most recent scheduled retraining run.

Type: Timestamp

LatestScheduledRetrainingStatus

Indicates the status of the most recent scheduled retraining run.

Type: String

Valid Values: `IN_PROGRESS | SUCCESS | FAILED | IMPORT_IN_PROGRESS | CANCELED`

ModelArn

The Amazon Resource Name (ARN) of the machine learning 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\/.+`

ModelDiagnosticsOutputConfiguration

Configuration information for the model's pointwise model diagnostics.

Type: ModelDiagnosticsOutputConfiguration object

ModelMetrics

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

The name of the machine learning model being described.

Type: String

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

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

ModelQuality

Provides a quality assessment for a model that uses labels. If Lookout for Equipment determines that the model quality is poor based on training metrics, the value is `POOR_QUALITY_DETECTED`. Otherwise, the value is `QUALITY_THRESHOLD_MET`.

If the model is unlabeled, the model quality can't be assessed and the value of `ModelQuality` is `CANNOT_DETERMINE_QUALITY`. In this situation, you can get a model quality assessment by adding labels to the input dataset and retraining the model.

For information about using labels with your models, see Understanding labeling.

For information about improving the quality of a model, see Best practices with Amazon Lookout for Equipment.

Type: String

Valid Values: `QUALITY_THRESHOLD_MET | CANNOT_DETERMINE_QUALITY | POOR_QUALITY_DETECTED`

ModelVersionActivatedAt

The date the active model version was activated.

Type: Timestamp

NextScheduledRetrainingStartDate

Indicates the date and time that the next scheduled retraining run will start on. Lookout for Equipment truncates the time you provide to the nearest UTC day.

Type: Timestamp

OffCondition

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.

PreviousActiveModelVersion

The model version that was set as the active model version prior to the current active model version.

Type: Long

Valid Range: Minimum value of 1.

PreviousActiveModelVersionArn

The ARN of the model version that was set as the active model version prior to the current active model version.

Type: String

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

Pattern: `^arn:aws(-[^:]+)?:lookoutequipment:[a-zA-Z0-9\-]*:[0-9]{12}:model\/[0-9a-zA-Z_-]{1,200}\/.+\/model-version\/[0-9]{1,}$`

PreviousModelVersionActivatedAt

The date and time when the previous active model version was activated.

Type: Timestamp

PriorModelMetrics

If the model version was retrained, this field shows a summary of the performance of the prior model on the new training range. You can use the information in this JSON-formatted object to compare the new model version and the prior model version.

Type: String

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

RetrainingSchedulerStatus

Indicates the status of the retraining scheduler.

Type: String

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

RoleArn

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

Type: String

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

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

Schema

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

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\-\/]+`

SourceModelVersionArn

The Amazon Resource Name (ARN) of the source model version. This field appears if the active model version was imported.

Type: String

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

Pattern: `^arn:aws(-[^:]+)?:lookoutequipment:[a-zA-Z0-9\-]*:[0-9]{12}:model\/[0-9a-zA-Z_-]{1,200}\/.+\/model-version\/[0-9]{1,}$`

Status

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 | IMPORT_IN_PROGRESS`

TrainingDataEndTime

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

Type: Timestamp

TrainingDataStartTime

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

Type: Timestamp

TrainingExecutionEndTime

Indicates the time at which the training of the machine learning model was completed.

Type: Timestamp

TrainingExecutionStartTime

Indicates the time at which the training of the machine learning 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 v2
- AWS SDK for Java V2
- AWS SDK for JavaScript V3
- AWS SDK for PHP V3
- AWS SDK for Python
- AWS SDK for Ruby V3

DescribeModelVersion

Retrieves information about a specific machine learning model version.

Request Syntax

```
{
   "ModelName": "string",
   "ModelVersion": number
}
```

Request Parameters

The request accepts the following data in JSON format.

ModelName

The name of the machine learning model that this version belongs to.

Type: String

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

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

Required: Yes

ModelVersion

The version of the machine learning model.

Type: Long

Valid Range: Minimum value of 1.

Required: Yes

Response Syntax

```
{
   "AutoPromotionResult": "string",
   "AutoPromotionResultReason": "string",
   "CreatedAt": number,
   "DataPreProcessingConfiguration": {
```

```
      "TargetSamplingRate": "string"
   },
   "DatasetArn": "string",
   "DatasetName": "string",
   "EvaluationDataEndTime": number,
   "EvaluationDataStartTime": number,
   "FailedReason": "string",
   "ImportedDataSizeInBytes": number,
   "ImportJobEndTime": number,
   "ImportJobStartTime": number,
   "LabelsInputConfiguration": {
      "LabelGroupName": "string",
      "S3InputConfiguration": {
         "Bucket": "string",
         "Prefix": "string"
      }
   },
   "LastUpdatedTime": number,
   "ModelArn": "string",
   "ModelDiagnosticsOutputConfiguration": {
      "KmsKeyId": "string",
      "S3OutputConfiguration": {
         "Bucket": "string",
         "Prefix": "string"
      }
   },
   "ModelDiagnosticsResultsObject": {
      "Bucket": "string",
      "Key": "string"
   },
   "ModelMetrics": "string",
   "ModelName": "string",
   "ModelQuality": "string",
   "ModelVersion": number,
   "ModelVersionArn": "string",
   "OffCondition": "string",
   "PriorModelMetrics": "string",
   "RetrainingAvailableDataInDays": number,
   "RoleArn": "string",
   "Schema": "string",
   "ServerSideKmsKeyId": "string",
   "SourceModelVersionArn": "string",
   "SourceType": "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.

AutoPromotionResult

Indicates whether the model version was promoted to be the active version after retraining or if there was an error with or cancellation of the retraining.

Type: String

Valid Values: `MODEL_PROMOTED | MODEL_NOT_PROMOTED | RETRAINING_INTERNAL_ERROR | RETRAINING_CUSTOMER_ERROR | RETRAINING_CANCELLED`

AutoPromotionResultReason

Indicates the reason for the `AutoPromotionResult`. For example, a model might not be promoted if its performance was worse than the active version, if there was an error during training, or if the retraining scheduler was using `MANUAL` promote mode. The model will be promoted in `MANAGED` promote mode if the performance is better than the previous model.

Type: String

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

CreatedAt

Indicates the time and date at which the machine learning model version was created.

Type: Timestamp

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*

Type: DataPreProcessingConfiguration object

DatasetArn

The Amazon Resource Name (ARN) of the dataset used to train the model version.

Type: String

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

Pattern: `arn:aws(-[^:]+)?:lookoutequipment:[a-zA-Z0-9\-]*:[0-9]{12}:dataset\/[0-9a-zA-Z_-]{1,200}\/.+`

DatasetName

The name of the dataset used to train the model version.

Type: String

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

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

EvaluationDataEndTime

The date on which the data in the evaluation set began being gathered. If you imported the version, this is the date that the evaluation set data in the source version finished being gathered.

Type: Timestamp

EvaluationDataStartTime

The date on which the data in the evaluation set began being gathered. If you imported the version, this is the date that the evaluation set data in the source version began being gathered.

Type: Timestamp

FailedReason

The failure message if the training of the model version failed.

Type: String

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

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

ImportedDataSizeInBytes

The size in bytes of the imported data. This field appears if the model version was imported.

Type: Long

Valid Range: Minimum value of 0.

ImportJobEndTime

The date and time when the import job completed. This field appears if the model version was imported.

Type: Timestamp

ImportJobStartTime

The date and time when the import job began. This field appears if the model version was imported.

Type: Timestamp

LabelsInputConfiguration

Contains the configuration information for the S3 location being used to hold label data.

Type: LabelsInputConfiguration object

LastUpdatedTime

Indicates the last time the machine learning model version was updated.

Type: Timestamp

ModelArn

The Amazon Resource Name (ARN) of the parent machine learning model that this version belong to.

Type: String

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

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

ModelDiagnosticsOutputConfiguration

The Amazon S3 location where Amazon Lookout for Equipment saves the pointwise model diagnostics for the model version.

Type: ModelDiagnosticsOutputConfiguration object

ModelDiagnosticsResultsObject

Contains the Amazon S3 object path where Amazon Lookout for Equipment writes the model diagnostics results for the model version. The format is `User_Provided_Prefix/Model_Name/Model_Version/model_diagnostics_results.json.gz`.

`User_Provided_Prefix` is the prefix that you specify in the `ModelDiagnosticsOutputConfiguration` request parameter to the `CreateModel` or `UpdateModel` operations.

Type: S3Object object

ModelMetrics

Shows an aggregated summary, in JSON format, of the model's performance within the evaluation time range. These metrics are created when evaluating the model.

Type: String

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

ModelName

The name of the machine learning model that this version belongs to.

Type: String

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

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

ModelQuality

Provides a quality assessment for a model that uses labels. If Lookout for Equipment determines that the model quality is poor based on training metrics, the value is `POOR_QUALITY_DETECTED`. Otherwise, the value is `QUALITY_THRESHOLD_MET`.

If the model is unlabeled, the model quality can't be assessed and the value of `ModelQuality` is `CANNOT_DETERMINE_QUALITY`. In this situation, you can get a model quality assessment by adding labels to the input dataset and retraining the model.

For information about using labels with your models, see Understanding labeling.

For information about improving the quality of a model, see Best practices with Amazon Lookout for Equipment.

Type: String

Valid Values: `QUALITY_THRESHOLD_MET | CANNOT_DETERMINE_QUALITY | POOR_QUALITY_DETECTED`

ModelVersion

The version of the machine learning model.

Type: Long

Valid Range: Minimum value of 1.

ModelVersionArn

The Amazon Resource Name (ARN) of the model version.

Type: String

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

Pattern: `^arn:aws(-[^:]+)?:lookoutequipment:[a-zA-Z0-9\-]*:[0-9]{12}:model\/[0-9a-zA-Z_-]{1,200}\/.+\/model-version\/[0-9]{1,}$`

OffCondition

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.

PriorModelMetrics

If the model version was retrained, this field shows a summary of the performance of the prior model on the new training range. You can use the information in this JSON-formatted object to compare the new model version and the prior model version.

Type: String

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

RetrainingAvailableDataInDays

Indicates the number of days of data used in the most recent scheduled retraining run.

Type: Integer

RoleArn

The Amazon Resource Name (ARN) of the role that was used to train the model version.

Type: String

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

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

Schema

The schema of the data used to train the model version.

Type: String

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

ServerSideKmsKeyId

The identifier of the AWS KMS key key used to encrypt model version 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\-\/]+`

SourceModelVersionArn

If model version was imported, then this field is the arn of the source model version.

Type: String

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

Pattern: `^arn:aws(-[^:]+)?:lookoutequipment:[a-zA-Z0-9\-]*:[0-9]{12}:model\/[0-9a-zA-Z_-]{1,200}\/.+\/model-version\/[0-9]{1,}$`

SourceType

Indicates whether this model version was created by training or by importing.

Type: String

Valid Values: `TRAINING | RETRAINING | IMPORT`

Status

The current status of the model version.

Type: String

Valid Values: `IN_PROGRESS | SUCCESS | FAILED | IMPORT_IN_PROGRESS | CANCELED`

TrainingDataEndTime

The date on which the training data finished being gathered. If you imported the version, this is the date that the training data in the source version finished being gathered.

Type: Timestamp

TrainingDataStartTime

The date on which the training data began being gathered. If you imported the version, this is the date that the training data in the source version began being gathered.

Type: Timestamp

TrainingExecutionEndTime

The time when the training of the version completed.

Type: Timestamp

TrainingExecutionStartTime

The time when the training of the version 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 v2
- AWS SDK for Java V2
- AWS SDK for JavaScript V3
- AWS SDK for PHP V3
- AWS SDK for Python
- AWS SDK for Ruby V3

DescribeResourcePolicy

Provides the details of a resource policy attached to a resource.

Request Syntax

```
{
   "ResourceArn": "string"
}
```

Request Parameters

The request accepts the following data in JSON format.

ResourceArn

The Amazon Resource Name (ARN) of the resource that is associated with the resource policy.

Type: String

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

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

Required: Yes

Response Syntax

```
{
   "CreationTime": number,
   "LastModifiedTime": number,
   "PolicyRevisionId": "string",
   "ResourcePolicy": "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.

CreationTime

The time when the resource policy was created.

Type: Timestamp

LastModifiedTime

The time when the resource policy was last modified.

Type: Timestamp

PolicyRevisionId

A unique identifier for a revision of the resource policy.

Type: String

Length Constraints: Maximum length of 50.

Pattern: `[0-9A-Fa-f]+`

ResourcePolicy

The resource policy in a JSON-formatted string.

Type: String

Length Constraints: Minimum length of 1. Maximum length of 20000.

Pattern: `[\u0009\u000A\u000D\u0020-\u00FF]+`

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 v2
- AWS SDK for Java V2
- AWS SDK for JavaScript V3
- AWS SDK for PHP V3
- AWS SDK for Python
- AWS SDK for Ruby V3

DescribeRetrainingScheduler

Provides a description of the retraining scheduler, including information such as the model name and retraining parameters.

Request Syntax

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

Request Parameters

The request accepts the following data in JSON format.

ModelName

The name of the model that the retraining scheduler is attached to.

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,
   "LookbackWindow": "string",
   "ModelArn": "string",
   "ModelName": "string",
   "PromoteMode": "string",
   "RetrainingFrequency": "string",
   "RetrainingStartDate": number,
   "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

Indicates the time and date at which the retraining scheduler was created.

Type: Timestamp

LookbackWindow

The number of past days of data used for retraining.

Type: String

Pattern: `^P180D$|^P360D$|^P540D$|^P720D$`

ModelArn

The ARN of the model that the retraining scheduler is attached to.

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

The name of the model that the retraining scheduler is attached to.

Type: String

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

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

PromoteMode

Indicates how the service uses new models. In `MANAGED` mode, new models are used for inference if they have better performance than the current model. In `MANUAL` mode, the new models are not used until they are manually activated.

Type: String

Valid Values: `MANAGED | MANUAL`

RetrainingFrequency

The frequency at which the model retraining is set. This follows the ISO 8601 guidelines.

Type: String

Length Constraints: Minimum length of 1. Maximum length of 10.

Pattern: `^P(\dY)?(\d{1,2}M)?(\d{1,3}D)?$`

RetrainingStartDate

The start date for the retraining scheduler. Lookout for Equipment truncates the time you provide to the nearest UTC day.

Type: Timestamp

Status

The status of the retraining scheduler.

Type: String

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

UpdatedAt

Indicates the time and date at which the retraining scheduler was updated.

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 v2
- AWS SDK for Java V2
- AWS SDK for JavaScript V3
- AWS SDK for PHP V3
- AWS SDK for Python
- AWS SDK for Ruby V3

ImportDataset

Imports a dataset.

Request Syntax

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

Request Parameters

The request accepts the following data in JSON format.

ClientToken

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

The name of the machine learning dataset to be created. If the dataset already exists, Amazon Lookout for Equipment overwrites the existing dataset. If you don't specify this field, it is filled with the name of the source dataset.

Type: String

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

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

Required: No

ServerSideKmsKeyId

Provides the identifier of the AWS KMS key 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

SourceDatasetArn

The Amazon Resource Name (ARN) of the dataset to import.

Type: String

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

Pattern: `arn:aws(-[^:]+)?:lookoutequipment:[a-zA-Z0-9\-]*:[0-9]{12}:dataset\/[0-9a-zA-Z_-]{1,200}\/.+`

Required: Yes

Tags

Any tags associated with the dataset to be created.

Type: Array of Tag objects

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

Required: No

Response Syntax

```
{
```

```
   "DatasetArn": "string",
   "DatasetName": "string",
   "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.

DatasetArn

The Amazon Resource Name (ARN) of the dataset that was imported.

Type: String

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

Pattern: `arn:aws(-[^:]+)?:lookoutequipment:[a-zA-Z0-9\-]*:[0-9]{12}:dataset\/[0-9a-zA-Z_-]{1,200}\/.+`

DatasetName

The name of the created machine learning dataset.

Type: String

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

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

JobId

A unique identifier for the job of importing the dataset.

Type: String

Length Constraints: Maximum length of 32.

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

Status

The status of the `ImportDataset` operation.

Type: String

Valid Values: `CREATED | INGESTION_IN_PROGRESS | ACTIVE | IMPORT_IN_PROGRESS`

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 v2
- AWS SDK for Java V2
- AWS SDK for JavaScript V3
- AWS SDK for PHP V3
- AWS SDK for Python
- AWS SDK for Ruby V3

ImportModelVersion

Imports a model that has been trained successfully.

Request Syntax

```
{
   "ClientToken": "string",
   "DatasetName": "string",
   "InferenceDataImportStrategy": "string",
   "LabelsInputConfiguration": {
      "LabelGroupName": "string",
      "S3InputConfiguration": {
         "Bucket": "string",
         "Prefix": "string"
      }
   },
   "ModelName": "string",
   "RoleArn": "string",
   "ServerSideKmsKeyId": "string",
   "SourceModelVersionArn": "string",
   "Tags": [
      {
         "Key": "string",
         "Value": "string"
      }
   ]
}
```

Request Parameters

The request accepts the following data in JSON format.

ClientToken

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

The name of the dataset for the machine learning model being imported.

Type: String

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

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

Required: Yes

InferenceDataImportStrategy

Indicates how to import the accumulated inference data when a model version is imported. The possible values are as follows:

- NO_IMPORT – Don't import the data.
- ADD_WHEN_EMPTY – Only import the data from the source model if there is no existing data in the target model.
- OVERWRITE – Import the data from the source model and overwrite the existing data in the target model.

Type: String

Valid Values: `NO_IMPORT | ADD_WHEN_EMPTY | OVERWRITE`

Required: No

LabelsInputConfiguration

Contains the configuration information for the S3 location being used to hold label data.

Type: LabelsInputConfiguration object

Required: No

ModelName

The name for the machine learning model to be created. If the model already exists, Amazon Lookout for Equipment creates a new version. If you do not specify this field, it is filled with the name of the source model.

Type: String

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

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

Required: No

RoleArn

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

Type: String

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

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

Required: No

ServerSideKmsKeyId

Provides the identifier of the AWS KMS key 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

SourceModelVersionArn

The Amazon Resource Name (ARN) of the model version to import.

Type: String

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

Pattern: `^arn:aws(-[^:]+)?:lookoutequipment:[a-zA-Z0-9\-]*:[0-9]{12}:model\/[0-9a-zA-Z_-]{1,200}\/.+\/model-version\/[0-9]{1,}$`

Required: Yes

Tags

The tags associated with the machine learning model to be created.

Type: Array of Tag objects

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

Required: No

Response Syntax

```
{
   "ModelArn": "string",
   "ModelName": "string",
   "ModelVersion": number,
   "ModelVersionArn": "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

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\/.+`

ModelName

The name for the machine learning model.

Type: String

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

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

ModelVersion

The version of the model being created.

Type: Long

Valid Range: Minimum value of 1.

ModelVersionArn

The Amazon Resource Name (ARN) of the model version 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\/[0-9a-zA-Z_-]{1,200}\/.+\/model-version\/[0-9]{1,}$`

Status

The status of the `ImportModelVersion` operation.

Type: String

Valid Values: `IN_PROGRESS | SUCCESS | FAILED | IMPORT_IN_PROGRESS | CANCELED`

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 v2
- AWS SDK for Java V2
- AWS SDK for JavaScript V3

- 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

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

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

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

Indicates the status of the data ingestion job.

Type: String

Valid Values: `IN_PROGRESS | SUCCESS | FAILED | IMPORT_IN_PROGRESS`

Required: No

Response Syntax

```
{
   "DataIngestionJobSummaries": [
      {
         "DatasetArn": "string",
         "DatasetName": "string",
         "IngestionInputConfiguration": {
            "S3InputConfiguration": {
               "Bucket": "string",
               "KeyPattern": "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

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

Type: Array of DataIngestionJobSummary objects

NextToken

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 v2
- AWS SDK for Java V2
- AWS SDK for JavaScript V3
- 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

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

Specifies the maximum number of datasets to list.

Type: Integer

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

Required: No

NextToken

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

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

Type: Array of DatasetSummary objects

NextToken

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 v2
- AWS SDK for Java V2
- AWS SDK for JavaScript V3
- AWS SDK for PHP V3
- AWS SDK for Python
- AWS SDK for Ruby V3

ListInferenceEvents

Lists all inference events that have been found for the specified inference scheduler.

Request Syntax

```
{
   "InferenceSchedulerName": "string",
   "IntervalEndTime": number,
   "IntervalStartTime": number,
   "MaxResults": number,
   "NextToken": "string"
}
```

Request Parameters

The request accepts the following data in JSON format.

InferenceSchedulerName

The name of the inference scheduler for the inference events listed.

Type: String

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

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

Required: Yes

IntervalEndTime

Returns all the inference events with an end start time equal to or greater than less than the end time given.

Type: Timestamp

Required: Yes

IntervalStartTime

Lookout for Equipment will return all the inference events with an end time equal to or greater than the start time given.

Type: Timestamp

Required: Yes

MaxResults

Specifies the maximum number of inference events to list.

Type: Integer

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

Required: No

NextToken

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

Type: String

Length Constraints: Maximum length of 8192.

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

Required: No

Response Syntax

```
{
   "InferenceEventSummaries": [
      {
         "Diagnostics": "string",
         "EventDurationInSeconds": number,
         "EventEndTime": number,
         "EventStartTime": number,
         "InferenceSchedulerArn": "string",
         "InferenceSchedulerName": "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.

InferenceEventSummaries

Provides an array of information about the individual inference events returned from the `ListInferenceEvents` operation, including scheduler used, event start time, event end time, diagnostics, and so on.

Type: Array of InferenceEventSummary objects

NextToken

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 v2
- AWS SDK for Java V2
- AWS SDK for JavaScript V3
- 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

The time reference in the inferenced dataset before which Amazon Lookout for Equipment stopped the inference execution.

Type: Timestamp

Required: No

DataStartTimeAfter

The time reference in the inferenced dataset after which Amazon Lookout for Equipment started the inference execution.

Type: Timestamp

Required: No

InferenceSchedulerName

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

Specifies the maximum number of inference executions to list.

Type: Integer

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

Required: No

NextToken

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

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",
         "ModelVersion": number,
         "ModelVersionArn": "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

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.

> **Note**
>
> If you don't supply the `InferenceSchedulerName` request parameter, or if you supply the name of an inference scheduler that doesn't exist, `ListInferenceExecutions` returns an empty array in `InferenceExecutionSummaries`.

Type: Array of InferenceExecutionSummary objects

NextToken

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 v2
- AWS SDK for Java V2
- AWS SDK for JavaScript V3
- 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",
   "Status": "string"
}
```

Request Parameters

The request accepts the following data in JSON format.

InferenceSchedulerNameBeginsWith

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

Specifies the maximum number of inference schedulers to list.

Type: Integer

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

Required: No

ModelName

The name of the machine learning 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

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

Status

Specifies the current status of the inference schedulers.

Type: String

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

Required: No

Response Syntax

```
{
   "InferenceSchedulerSummaries": [
      {
         "DataDelayOffsetInMinutes": number,
         "DataUploadFrequency": "string",
         "InferenceSchedulerArn": "string",
         "InferenceSchedulerName": "string",
         "LatestInferenceResult": "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

Provides information about the specified inference scheduler, including data upload frequency, model name and ARN, and status.

Type: Array of InferenceSchedulerSummary objects

NextToken

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 v2
- AWS SDK for Java V2
- AWS SDK for JavaScript V3
- AWS SDK for PHP V3
- AWS SDK for Python
- AWS SDK for Ruby V3

ListLabelGroups

Returns a list of the label groups.

Request Syntax

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

Request Parameters

The request accepts the following data in JSON format.

LabelGroupNameBeginsWith

The beginning of the name of the label groups 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

Specifies the maximum number of label groups to list.

Type: Integer

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

Required: No

NextToken

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

Type: String

Length Constraints: Maximum length of 8192.

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

Required: No

Response Syntax

```
{
   "LabelGroupSummaries": [
      {
         "CreatedAt": number,
         "LabelGroupArn": "string",
         "LabelGroupName": "string",
         "UpdatedAt": number
      }
   ],
   "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.

LabelGroupSummaries

A summary of the label groups.

Type: Array of LabelGroupSummary objects

NextToken

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

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 v2
- AWS SDK for Java V2
- AWS SDK for JavaScript V3
- AWS SDK for PHP V3
- AWS SDK for Python
- AWS SDK for Ruby V3

ListLabels

Provides a list of labels.

Request Syntax

```
{
   "Equipment": "string",
   "FaultCode": "string",
   "IntervalEndTime": number,
   "IntervalStartTime": number,
   "LabelGroupName": "string",
   "MaxResults": number,
   "NextToken": "string"
}
```

Request Parameters

The request accepts the following data in JSON format.

Equipment

Lists the labels that pertain to a particular piece of equipment.

Type: String

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

Pattern: `[\P{M}\p{M}]{1,200}`

Required: No

FaultCode

Returns labels with a particular fault code.

Type: String

Length Constraints: Minimum length of 1. Maximum length of 100.

Pattern: `[\P{M}\p{M}]{1,100}`

Required: No

IntervalEndTime

Returns all labels with a start time earlier than the end time given.

Type: Timestamp

Required: No

IntervalStartTime

Returns all the labels with a end time equal to or later than the start time given.

Type: Timestamp

Required: No

LabelGroupName

Returns the name of the label group.

Type: String

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

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

Required: Yes

MaxResults

Specifies the maximum number of labels to list.

Type: Integer

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

Required: No

NextToken

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

Type: String

Length Constraints: Maximum length of 8192.

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

Required: No

Response Syntax

```
{
   "LabelSummaries": [
      {
         "CreatedAt": number,
         "EndTime": number,
         "Equipment": "string",
         "FaultCode": "string",
         "LabelGroupArn": "string",
         "LabelGroupName": "string",
         "LabelId": "string",
         "Rating": "string",
         "StartTime": number
      }
   ],
   "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.

LabelSummaries

A summary of the items in the label group.

> **Note**
>
> If you don't supply the `LabelGroupName` request parameter, or if you supply the name of a label group that doesn't exist, `ListLabels` returns an empty array in `LabelSummaries`.

Type: Array of LabelSummary objects

NextToken

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 v2
- AWS SDK for Java V2
- AWS SDK for JavaScript V3
- 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

The beginning of the name of the dataset of the machine learning 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

Specifies the maximum number of machine learning models to list.

Type: Integer

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

Required: No

ModelNameBeginsWith

The beginning of the name of the machine learning 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

An opaque pagination token indicating where to continue the listing of machine learning models.

Type: String

Length Constraints: Maximum length of 8192.

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

Required: No

Status

The status of the machine learning model.

Type: String

Valid Values: `IN_PROGRESS | SUCCESS | FAILED | IMPORT_IN_PROGRESS`

Required: No

Response Syntax

```
{
   "ModelSummaries": [
      {
         "ActiveModelVersion": number,
         "ActiveModelVersionArn": "string",
         "CreatedAt": number,
         "DatasetArn": "string",
         "DatasetName": "string",
         "LatestScheduledRetrainingModelVersion": number,
```

```
         "LatestScheduledRetrainingStartTime": number,
         "LatestScheduledRetrainingStatus": "string",
         "ModelArn": "string",
         "ModelDiagnosticsOutputConfiguration": {
            "KmsKeyId": "string",
            "S3OutputConfiguration": {
               "Bucket": "string",
               "Prefix": "string"
            }
         },
         "ModelName": "string",
         "ModelQuality": "string",
         "NextScheduledRetrainingStartDate": number,
         "RetrainingSchedulerStatus": "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

Provides information on the specified model, including created time, model and dataset ARNs, and status.

Type: Array of ModelSummary objects

NextToken

An opaque pagination token indicating where to continue the listing of machine learning 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 v2
- AWS SDK for Java V2
- AWS SDK for JavaScript V3
- AWS SDK for PHP V3
- AWS SDK for Python
- AWS SDK for Ruby V3

ListModelVersions

Generates a list of all model versions for a given model, including the model version, model version ARN, and status. To list a subset of versions, use the `MaxModelVersion` and `MinModelVersion` fields.

Request Syntax

```
{
   "CreatedAtEndTime": number,
   "CreatedAtStartTime": number,
   "MaxModelVersion": number,
   "MaxResults": number,
   "MinModelVersion": number,
   "ModelName": "string",
   "NextToken": "string",
   "SourceType": "string",
   "Status": "string"
}
```

Request Parameters

The request accepts the following data in JSON format.

CreatedAtEndTime

Filter results to return all the model versions created before this time.

Type: Timestamp

Required: No

CreatedAtStartTime

Filter results to return all the model versions created after this time.

Type: Timestamp

Required: No

MaxModelVersion

Specifies the highest version of the model to return in the list.

Type: Long

Valid Range: Minimum value of 1.

Required: No

MaxResults

Specifies the maximum number of machine learning model versions to list.

Type: Integer

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

Required: No

MinModelVersion

Specifies the lowest version of the model to return in the list.

Type: Long

Valid Range: Minimum value of 1.

Required: No

ModelName

Then name of the machine learning model for which the model versions are to be listed.

Type: String

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

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

Required: Yes

NextToken

If the total number of results exceeds the limit that the response can display, the response returns an opaque pagination token indicating where to continue the listing of machine learning model versions. Use this token in the `NextToken` field in the request to list the next page of results.

Type: String

Length Constraints: Maximum length of 8192.

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

Required: No

SourceType

Filter the results based on the way the model version was generated.

Type: String

Valid Values: `TRAINING | RETRAINING | IMPORT`

Required: No

Status

Filter the results based on the current status of the model version.

Type: String

Valid Values: `IN_PROGRESS | SUCCESS | FAILED | IMPORT_IN_PROGRESS | CANCELED`

Required: No

Response Syntax

```
{
   "ModelVersionSummaries": [
      {
         "CreatedAt": number,
         "ModelArn": "string",
         "ModelName": "string",
         "ModelQuality": "string",
         "ModelVersion": number,
         "ModelVersionArn": "string",
         "SourceType": "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.

ModelVersionSummaries

Provides information on the specified model version, including the created time, model and dataset ARNs, and status.

> **Note**
>
> If you don't supply the `ModelName` request parameter, or if you supply the name of a model that doesn't exist, `ListModelVersions` returns an empty array in `ModelVersionSummaries`.

Type: Array of ModelVersionSummary objects

NextToken

If the total number of results exceeds the limit that the response can display, the response returns an opaque pagination token indicating where to continue the listing of machine learning model versions. Use this token in the `NextToken` field in the request to list the next page of results.

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 v2
- AWS SDK for Java V2
- AWS SDK for JavaScript V3
- AWS SDK for PHP V3
- AWS SDK for Python
- AWS SDK for Ruby V3

ListRetrainingSchedulers

Lists all retraining schedulers in your account, filtering by model name prefix and status.

Request Syntax

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

Request Parameters

The request accepts the following data in JSON format.

MaxResults

Specifies the maximum number of retraining schedulers to list.

Type: Integer

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

Required: No

ModelNameBeginsWith

Specify this field to only list retraining schedulers whose machine learning models begin with the value you specify.

Type: String

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

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

Required: No

NextToken

If the number of results exceeds the maximum, a pagination token is returned. Use the token in the request to show the next page of retraining schedulers.

Type: String

Length Constraints: Maximum length of 8192.

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

Required: No

Status

Specify this field to only list retraining schedulers whose status matches the value you specify.

Type: String

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

Required: No

Response Syntax

```
{
   "NextToken": "string",
   "RetrainingSchedulerSummaries": [
      {
         "LookbackWindow": "string",
         "ModelArn": "string",
         "ModelName": "string",
         "RetrainingFrequency": "string",
         "RetrainingStartDate": number,
         "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.

NextToken

If the number of results exceeds the maximum, this pagination token is returned. Use this token in the request to show the next page of retraining schedulers.

Type: String

Length Constraints: Maximum length of 8192.

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

RetrainingSchedulerSummaries

Provides information on the specified retraining scheduler, including the model name, model ARN, status, and start date.

Type: Array of RetrainingSchedulerSummary objects

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 v2
- AWS SDK for Java V2
- AWS SDK for JavaScript V3
- AWS SDK for PHP V3
- AWS SDK for Python
- AWS SDK for Ruby V3

ListSensorStatistics

Lists statistics about the data collected for each of the sensors that have been successfully ingested in the particular dataset. Can also be used to retreive Sensor Statistics for a previous ingestion job.

Request Syntax

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

Request Parameters

The request accepts the following data in JSON format.

DatasetName

The name of the dataset associated with the list of Sensor Statistics.

Type: String

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

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

Required: Yes

IngestionJobId

The ingestion job id associated with the list of Sensor Statistics. To get sensor statistics for a particular ingestion job id, both dataset name and ingestion job id must be submitted as inputs.

Type: String

Length Constraints: Maximum length of 32.

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

Required: No

MaxResults

Specifies the maximum number of sensors for which to retrieve statistics.

Type: Integer

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

Required: No

NextToken

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

Type: String

Length Constraints: Maximum length of 8192.

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

Required: No

Response Syntax

```
{
   "NextToken": "string",
   "SensorStatisticsSummaries": [
      {
         "CategoricalValues": {
            "NumberOfCategory": number,
            "Status": "string"
         },
         "ComponentName": "string",
         "DataEndTime": number,
         "DataExists": boolean,
         "DataStartTime": number,
         "DuplicateTimestamps": {
            "Count": number,
            "Percentage": number
         },
         "InvalidDateEntries": {
            "Count": number,
            "Percentage": number
         },
         "InvalidValues": {
```

```
            "Count": number,
            "Percentage": number
         },
         "LargeTimestampGaps": {
            "MaxTimestampGapInDays": number,
            "NumberOfLargeTimestampGaps": number,
            "Status": "string"
         },
         "MissingValues": {
            "Count": number,
            "Percentage": number
         },
         "MonotonicValues": {
            "Monotonicity": "string",
            "Status": "string"
         },
         "MultipleOperatingModes": {
            "Status": "string"
         },
         "SensorName": "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.

NextToken

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

Type: String

Length Constraints: Maximum length of 8192.

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

SensorStatisticsSummaries

Provides ingestion-based statistics regarding the specified sensor with respect to various validation types, such as whether data exists, the number and percentage of missing values, and the number and percentage of duplicate timestamps.

Type: Array of SensorStatisticsSummary objects

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 v2
- AWS SDK for Java V2
- AWS SDK for JavaScript V3
- 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

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

Any tags associated with the resource.

Type: Array of Tag 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 v2
- AWS SDK for Java V2
- AWS SDK for JavaScript V3
- AWS SDK for PHP V3
- AWS SDK for Python
- AWS SDK for Ruby V3

PutResourcePolicy

Creates a resource control policy for a given resource.

Request Syntax

```
{
   "ClientToken": "string",
   "PolicyRevisionId": "string",
   "ResourceArn": "string",
   "ResourcePolicy": "string"
}
```

Request Parameters

The request accepts the following data in JSON format.

ClientToken

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

PolicyRevisionId

A unique identifier for a revision of the resource policy.

Type: String

Length Constraints: Maximum length of 50.

Pattern: `[0-9A-Fa-f]+`

Required: No

ResourceArn

The Amazon Resource Name (ARN) of the resource for which the policy is 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}:.+`

Required: Yes

ResourcePolicy

The JSON-formatted resource policy to create.

Type: String

Length Constraints: Minimum length of 1. Maximum length of 20000.

Pattern: `[\u0009\u000A\u000D\u0020-\u00FF]+`

Required: Yes

Response Syntax

```
{
   "PolicyRevisionId": "string",
   "ResourceArn": "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.

PolicyRevisionId

A unique identifier for a revision of the resource policy.

Type: String

Length Constraints: Maximum length of 50.

Pattern: `[0-9A-Fa-f]+`

ResourceArn

The Amazon Resource Name (ARN) of the resource for which the policy was created.

Type: String

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

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

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 v2
- AWS SDK for Java V2
- AWS SDK for JavaScript V3
- 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",
         "KeyPattern": "string",
         "Prefix": "string"
      }
   },
   "RoleArn": "string"
}
```

Request Parameters

The request accepts the following data in JSON format.

ClientToken

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

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

Specifies information for the input data for the data ingestion job, including dataset S3 location.

Type: IngestionInputConfiguration object

Required: Yes

RoleArn

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

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

Indicates the status of the `StartDataIngestionJob` operation.

Type: String

Valid Values: `IN_PROGRESS | SUCCESS | FAILED | IMPORT_IN_PROGRESS`

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 v2
- AWS SDK for Java V2
- AWS SDK for JavaScript V3
- 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

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

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

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

The Amazon Resource Name (ARN) of the machine learning 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

The name of the machine learning 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

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 v2
- AWS SDK for Java V2
- AWS SDK for JavaScript V3
- AWS SDK for PHP V3
- AWS SDK for Python
- AWS SDK for Ruby V3

StartRetrainingScheduler

Starts a retraining scheduler.

Request Syntax

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

Request Parameters

The request accepts the following data in JSON format.

ModelName

The name of the model whose retraining scheduler you want to start.

Type: String

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

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

Required: Yes

Response Syntax

```
{
   "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.

ModelArn

The ARN of the model whose retraining scheduler is 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}:model\/.+`

ModelName

The name of the model whose retraining scheduler is being started.

Type: String

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

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

Status

The status of the retraining 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 v2
- AWS SDK for Java V2
- AWS SDK for JavaScript V3
- 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

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

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

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

The Amazon Resource Name (ARN) of the machine learning 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

The name of the machine learning 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

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 v2
- AWS SDK for Java V2
- AWS SDK for JavaScript V3
- AWS SDK for PHP V3
- AWS SDK for Python
- AWS SDK for Ruby V3

StopRetrainingScheduler

Stops a retraining scheduler.

Request Syntax

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

Request Parameters

The request accepts the following data in JSON format.

ModelName

The name of the model whose retraining scheduler you want to stop.

Type: String

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

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

Required: Yes

Response Syntax

```
{
   "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.

ModelArn

The ARN of the model whose retraining scheduler is 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

The name of the model whose retraining scheduler is being stopped.

Type: String

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

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

Status

The status of the retraining 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 v2
- AWS SDK for Java V2
- AWS SDK for JavaScript V3
- 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

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

The tag or tags to be associated with a specific resource. Both the tag key and value are specified.

Type: Array of Tag 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 v2
- AWS SDK for Java V2
- AWS SDK for JavaScript V3
- 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

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

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 v2
- AWS SDK for Java V2

- AWS SDK for JavaScript V3
- AWS SDK for PHP V3
- AWS SDK for Python
- AWS SDK for Ruby V3

UpdateActiveModelVersion

Sets the active model version for a given machine learning model.

Request Syntax

```
{
   "ModelName": "string",
   "ModelVersion": number
}
```

Request Parameters

The request accepts the following data in JSON format.

ModelName

The name of the machine learning model for which the active model version is being set.

Type: String

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

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

Required: Yes

ModelVersion

The version of the machine learning model for which the active model version is being set.

Type: Long

Valid Range: Minimum value of 1.

Required: Yes

Response Syntax

```
{
   "CurrentActiveVersion": number,
   "CurrentActiveVersionArn": "string",
```

```
    "ModelArn": "string",
    "ModelName": "string",
    "PreviousActiveVersion": number,
    "PreviousActiveVersionArn": "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.

CurrentActiveVersion

The version that is currently active of the machine learning model for which the active model version was set.

Type: Long

Valid Range: Minimum value of 1.

CurrentActiveVersionArn

The Amazon Resource Name (ARN) of the machine learning model version that is the current active model version.

Type: String

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

Pattern: `^arn:aws(-[^:]+)?:lookoutequipment:[a-zA-Z0-9\-]*:[0-9]{12}:model\/[0-9a-zA-Z_-]{1,200}\/.+\/model-version\/[0-9]{1,}$`

ModelArn

The Amazon Resource Name (ARN) of the machine learning model for which the active model version was set.

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

The name of the machine learning model for which the active model version was set.

Type: String

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

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

PreviousActiveVersion

The previous version that was active of the machine learning model for which the active model version was set.

Type: Long

Valid Range: Minimum value of 1.

PreviousActiveVersionArn

The Amazon Resource Name (ARN) of the machine learning model version that was the previous active model version.

Type: String

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

Pattern: `^arn:aws(-[^:]+)?:lookoutequipment:[a-zA-Z0-9\-]*:[0-9]{12}:model\/[0-9a-zA-Z_-]{1,200}\/.+\/model-version\/[0-9]{1,}$`

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 v2
- AWS SDK for Java V2
- AWS SDK for JavaScript V3
- 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

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

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

Type: InferenceInputConfiguration object

Required: No

DataOutputConfiguration

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

Type: InferenceOutputConfiguration object

Required: No

DataUploadFrequency

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

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

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 v2
- AWS SDK for Java V2
- AWS SDK for JavaScript V3
- AWS SDK for PHP V3
- AWS SDK for Python
- AWS SDK for Ruby V3

UpdateLabelGroup

Updates the label group.

Request Syntax

```
{
   "FaultCodes": [ "string" ],
   "LabelGroupName": "string"
}
```

Request Parameters

The request accepts the following data in JSON format.

FaultCodes

Updates the code indicating the type of anomaly associated with the label.

Data in this field will be retained for service usage. Follow best practices for the security of your data.

Type: Array of strings

Array Members: Minimum number of 0 items. Maximum number of 50 items.

Length Constraints: Minimum length of 1. Maximum length of 100.

Pattern: `[\P{M}\p{M}]{1,100}`

Required: No

LabelGroupName

The name of the label group to be updated.

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 v2
- AWS SDK for Java V2
- AWS SDK for JavaScript V3
- AWS SDK for PHP V3
- AWS SDK for Python
- AWS SDK for Ruby V3

UpdateModel

Updates a model in the account.

Request Syntax

```
{
   "LabelsInputConfiguration": {
      "LabelGroupName": "string",
      "S3InputConfiguration": {
         "Bucket": "string",
         "Prefix": "string"
      }
   },
   "ModelDiagnosticsOutputConfiguration": {
      "KmsKeyId": "string",
      "S3OutputConfiguration": {
         "Bucket": "string",
         "Prefix": "string"
      }
   },
   "ModelName": "string",
   "RoleArn": "string"
}
```

Request Parameters

The request accepts the following data in JSON format.

LabelsInputConfiguration

Contains the configuration information for the S3 location being used to hold label data.

Type: LabelsInputConfiguration object

Required: No

ModelDiagnosticsOutputConfiguration

The Amazon S3 location where you want Amazon Lookout for Equipment to save the pointwise model diagnostics for the model. You must also specify the `RoleArn` request parameter.

Type: ModelDiagnosticsOutputConfiguration object

Required: No

ModelName

The name of the model to update.

Type: String

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

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

Required: Yes

RoleArn

The ARN of the model to update.

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 v2
- AWS SDK for Java V2
- AWS SDK for JavaScript V3
- AWS SDK for PHP V3
- AWS SDK for Python
- AWS SDK for Ruby V3

UpdateRetrainingScheduler

Updates a retraining scheduler.

Request Syntax

```
{
   "LookbackWindow": "string",
   "ModelName": "string",
   "PromoteMode": "string",
   "RetrainingFrequency": "string",
   "RetrainingStartDate": number
}
```

Request Parameters

The request accepts the following data in JSON format.

LookbackWindow

The number of past days of data that will be used for retraining.

Type: String

Pattern: `^P180D$|^P360D$|^P540D$|^P720D$`

Required: No

ModelName

The name of the model whose retraining scheduler you want to update.

Type: String

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

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

Required: Yes

PromoteMode

Indicates how the service will use new models. In `MANAGED` mode, new models will automatically be used for inference if they have better performance than the current model. In `MANUAL` mode, the new models will not be used until they are manually activated.

Type: String

Valid Values: `MANAGED | MANUAL`

Required: No

RetrainingFrequency

This parameter uses the ISO 8601 standard to set the frequency at which you want retraining to occur in terms of Years, Months, and/or Days (note: other parameters like Time are not currently supported). The minimum value is 30 days (P30D) and the maximum value is 1 year (P1Y). For example, the following values are valid:

- P3M15D – Every 3 months and 15 days
- P2M – Every 2 months
- P150D – Every 150 days

Type: String

Length Constraints: Minimum length of 1. Maximum length of 10.

Pattern: `^P(\dY)?(\d{1,2}M)?(\d{1,3}D)?$`

Required: No

RetrainingStartDate

The start date for the retraining scheduler. Lookout for Equipment truncates the time you provide to the nearest UTC day.

Type: Timestamp

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 v2
- AWS SDK for Java V2

- AWS SDK for JavaScript V3
- AWS SDK for PHP V3
- AWS SDK for Python
- AWS SDK for Ruby V3

Data Types

The following data types are supported:

- CategoricalValues
- CountPercent
- DataIngestionJobSummary
- DataPreProcessingConfiguration
- DataQualitySummary
- DatasetSchema
- DatasetSummary
- DuplicateTimestamps
- InferenceEventSummary
- InferenceExecutionSummary
- InferenceInputConfiguration
- InferenceInputNameConfiguration
- InferenceOutputConfiguration
- InferenceS3InputConfiguration
- InferenceS3OutputConfiguration
- InferenceSchedulerSummary
- IngestedFilesSummary
- IngestionInputConfiguration
- IngestionS3InputConfiguration
- InsufficientSensorData
- InvalidSensorData
- LabelGroupSummary

- LabelsInputConfiguration
- LabelsS3InputConfiguration
- LabelSummary
- LargeTimestampGaps
- MissingCompleteSensorData
- MissingSensorData
- ModelDiagnosticsOutputConfiguration
- ModelDiagnosticsS3OutputConfiguration
- ModelSummary
- ModelVersionSummary
- MonotonicValues
- MultipleOperatingModes
- RetrainingSchedulerSummary
- S3Object
- SensorStatisticsSummary
- SensorsWithShortDateRange
- Tag
- UnsupportedTimestamps

CategoricalValues

Entity that comprises information on categorical values in data.

Contents

Status

Indicates whether there is a potential data issue related to categorical values.

Type: String

Valid Values: `POTENTIAL_ISSUE_DETECTED | NO_ISSUE_DETECTED`

Required: Yes

NumberOfCategory

Indicates the number of categories in the data.

Type: Integer

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 Java V2
- AWS SDK for Ruby V3

CountPercent

Entity that comprises information of count and percentage.

Contents

Count

Indicates the count of occurences of the given statistic.

Type: Integer

Required: Yes

Percentage

Indicates the percentage of occurances of the given statistic.

Type: Float

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 Java V2
- AWS SDK for Ruby V3

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\/[0-9a-zA-Z_-]{1,200}\/.+`

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 Amazon S3 location parameters.

Type: IngestionInputConfiguration 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 | IMPORT_IN_PROGRESS`

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 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 Java V2
- AWS SDK for Ruby V3

DataQualitySummary

DataQualitySummary gives aggregated statistics over all the sensors about a completed ingestion job. It primarily gives more information about statistics over different incorrect data like MissingCompleteSensorData, MissingSensorData, UnsupportedDateFormats, InsufficientSensorData, DuplicateTimeStamps.

Contents

DuplicateTimestamps

Parameter that gives information about duplicate timestamps in the input data.

Type: DuplicateTimestamps object

Required: Yes

InsufficientSensorData

Parameter that gives information about insufficient data for sensors in the dataset. This includes information about those sensors that have complete data missing and those with a short date range.

Type: InsufficientSensorData object

Required: Yes

InvalidSensorData

Parameter that gives information about data that is invalid over all the sensors in the input data.

Type: InvalidSensorData object

Required: Yes

MissingSensorData

Parameter that gives information about data that is missing over all the sensors in the input data.

Type: MissingSensorData object

Required: Yes

UnsupportedTimestamps

Parameter that gives information about unsupported timestamps in the input data.

Type: UnsupportedTimestamps 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 Java V2
- AWS SDK for Ruby V3

DatasetSchema

Provides information about the data schema used with the given dataset.

Contents

InlineDataSchema

The data schema used within the given dataset.

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 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\/[0-9a-zA-Z_-]{1,200}\/.+`

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 | IMPORT_IN_PROGRESS`

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 Java V2
- AWS SDK for Ruby V3

DuplicateTimestamps

Entity that comprises information abount duplicate timestamps in the dataset.

Contents

TotalNumberOfDuplicateTimestamps

Indicates the total number of duplicate timestamps.

Type: Integer

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 Java V2
- AWS SDK for Ruby V3

InferenceEventSummary

Contains information about the specific inference event, including start and end time, diagnostics information, event duration and so on.

Contents

Diagnostics

An array which specifies the names and values of all sensors contributing to an inference event.

Type: String

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

Required: No

EventDurationInSeconds

Indicates the size of an inference event in seconds.

Type: Long

Valid Range: Minimum value of 0.

Required: No

EventEndTime

Indicates the ending time of an inference event.

Type: Timestamp

Required: No

EventStartTime

Indicates the starting time of an inference event.

Type: Timestamp

Required: No

InferenceSchedulerArn

The Amazon Resource Name (ARN) of the inference scheduler being used for the inference event.

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 events.

Type: String

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

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

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 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

The S3 object that the inference execution results were uploaded to.

Type: S3Object 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 object

Required: No

DataOutputConfiguration

Specifies configuration information for the output results from for the inference execution, including the output Amazon S3 location.

Type: InferenceOutputConfiguration 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 machine learning 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 machine learning 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

ModelVersion

The model version used for the inference execution.

Type: Long

Valid Range: Minimum value of 1.

Required: No

ModelVersionArn

The Amazon Resource Number (ARN) of the model version 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\/[0-9a-zA-Z_-]{1,200}\/.+\/model-version\/[0-9]{1,}$`

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 Java V2
- AWS SDK for Ruby V3

InferenceInputConfiguration

Specifies configuration information for the input data for the inference, including Amazon S3 location of input data..

Contents

InferenceInputNameConfiguration

Specifies configuration information for the input data for the inference, including timestamp format and delimiter.

Type: InferenceInputNameConfiguration object

Required: No

InputTimeZoneOffset

Indicates the difference between your time zone and Coordinated Universal Time (UTC).

Type: String

Pattern: `^(\+|\-)[0-9]{2}\:[0-9]{2}$`

Required: No

S3InputConfiguration

Specifies configuration information for the input data for the inference, including Amazon S3 location of input data.

Type: InferenceS3InputConfiguration 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 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 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

S3OutputConfiguration

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

Type: InferenceS3OutputConfiguration object

Required: Yes

KmsKeyId

The ID number for the AWS KMS key 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

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 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: `(^$)|([\u0009\u000A\u000D\u0020-\u00FF]{1,1023}/$)`

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 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: `(^$)|([\u0009\u000A\u000D\u0020-\u00FF]{1,1023}/$)`

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 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

LatestInferenceResult

Indicates whether the latest execution for the inference scheduler was Anomalous (anomalous events found) or Normal (no anomalous events found).

Type: String

Valid Values: `ANOMALOUS | NORMAL`

Required: No

ModelArn

The Amazon Resource Name (ARN) of the machine learning 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 machine learning 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 Java V2
- AWS SDK for Ruby V3

IngestedFilesSummary

Gives statistics about how many files have been ingested, and which files have not been ingested, for a particular ingestion job.

Contents

IngestedNumberOfFiles

Indicates the number of files that were successfully ingested.

Type: Integer

Required: Yes

TotalNumberOfFiles

Indicates the total number of files that were submitted for ingestion.

Type: Integer

Required: Yes

DiscardedFiles

Indicates the number of files that were discarded. A file could be discarded because its format is invalid (for example, a jpg or pdf) or not readable.

Type: Array of S3Object objects

Array Members: Minimum number of 0 items.

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 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 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 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

KeyPattern

The pattern for matching the Amazon S3 files that will be used for ingestion. If the schema was created previously without any KeyPattern, then the default KeyPattern {prefix}/{component_name}/* is used to download files from Amazon S3 according to the schema. This field is required when ingestion is being done for the first time.

Valid Values: {prefix}/{component_name}_* | {prefix}/{component_name}/* | {prefix}/{component_name}[DELIMITER]* (Allowed delimiters : space, dot, underscore, hyphen)

Type: String

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

Required: No

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: `(^$)|([\u0009\u000A\u000D\u0020-\u00FF]{1,1023}/$)`

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 Java V2
- AWS SDK for Ruby V3

InsufficientSensorData

Entity that comprises aggregated information on sensors having insufficient data.

Contents

MissingCompleteSensorData

Parameter that describes the total number of sensors that have data completely missing for it.

Type: MissingCompleteSensorData object

Required: Yes

SensorsWithShortDateRange

Parameter that describes the total number of sensors that have a short date range of less than 14 days of data overall.

Type: SensorsWithShortDateRange 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 Java V2
- AWS SDK for Ruby V3

InvalidSensorData

Entity that comprises aggregated information on sensors having insufficient data.

Contents

AffectedSensorCount

Indicates the number of sensors that have at least some invalid values.

Type: Integer

Required: Yes

TotalNumberOfInvalidValues

Indicates the total number of invalid values across all the sensors.

Type: Integer

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 Java V2
- AWS SDK for Ruby V3

LabelGroupSummary

Contains information about the label group.

Contents

CreatedAt

The time at which the label group was created.

Type: Timestamp

Required: No

LabelGroupArn

The Amazon Resource Name (ARN) of the label group.

Type: String

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

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

Required: No

LabelGroupName

The name of the label group.

Type: String

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

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

Required: No

UpdatedAt

The time at which the label group was updated.

Type: Timestamp

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 Java V2
- AWS SDK for Ruby V3

LabelsInputConfiguration

Contains the configuration information for the S3 location being used to hold label data.

Contents

LabelGroupName

The name of the label group to be used for label data.

Type: String

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

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

Required: No

S3InputConfiguration

Contains location information for the S3 location being used for label data.

Type: LabelsS3InputConfiguration 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 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: `(^$)|([\u0009\u000A\u000D\u0020-\u00FF]{1,1023}/$)`

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 Java V2
- AWS SDK for Ruby V3

LabelSummary

Information about the label.

Contents

CreatedAt

The time at which the label was created.

Type: Timestamp

Required: No

EndTime

The timestamp indicating the end of the label.

Type: Timestamp

Required: No

Equipment

Indicates that a label pertains to a particular piece of equipment.

Type: String

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

Pattern: `[\P{M}\p{M}]{1,200}`

Required: No

FaultCode

Indicates the type of anomaly associated with the label.

Data in this field will be retained for service usage. Follow best practices for the security of your data.

Type: String

Length Constraints: Minimum length of 1. Maximum length of 100.

Pattern: `[\P{M}\p{M}]{1,100}`

Required: No

LabelGroupArn

The Amazon Resource Name (ARN) of the label group.

Type: String

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

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

Required: No

LabelGroupName

The name of the label group.

Type: String

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

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

Required: No

LabelId

The ID of the label.

Type: String

Length Constraints: Maximum length of 32.

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

Required: No

Rating

Indicates whether a labeled event represents an anomaly.

Type: String

Valid Values: `ANOMALY | NO_ANOMALY | NEUTRAL`

Required: No

StartTime

The timestamp indicating the start of the label.

Type: Timestamp

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 Java V2
- AWS SDK for Ruby V3

LargeTimestampGaps

Entity that comprises information on large gaps between consecutive timestamps in data.

Contents

Status

Indicates whether there is a potential data issue related to large gaps in timestamps.

Type: String

Valid Values: `POTENTIAL_ISSUE_DETECTED | NO_ISSUE_DETECTED`

Required: Yes

MaxTimestampGapInDays

Indicates the size of the largest timestamp gap, in days.

Type: Integer

Required: No

NumberOfLargeTimestampGaps

Indicates the number of large timestamp gaps, if there are any.

Type: Integer

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 Java V2
- AWS SDK for Ruby V3

MissingCompleteSensorData

Entity that comprises information on sensors that have sensor data completely missing.

Contents

AffectedSensorCount

Indicates the number of sensors that have data missing completely.

Type: Integer

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 Java V2
- AWS SDK for Ruby V3

MissingSensorData

Entity that comprises aggregated information on sensors having missing data.

Contents

AffectedSensorCount

Indicates the number of sensors that have atleast some data missing.

Type: Integer

Required: Yes

TotalNumberOfMissingValues

Indicates the total number of missing values across all the sensors.

Type: Integer

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 Java V2
- AWS SDK for Ruby V3

ModelDiagnosticsOutputConfiguration

Output configuration information for the pointwise model diagnostics for an Amazon Lookout for Equipment model.

Contents

S3OutputConfiguration

The Amazon S3 location for the pointwise model diagnostics.

Type: ModelDiagnosticsS3OutputConfiguration object

Required: Yes

KmsKeyId

The AWS Key Management Service (KMS) key identifier to encrypt the pointwise model diagnostics files.

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

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 Java V2
- AWS SDK for Ruby V3

ModelDiagnosticsS3OutputConfiguration

The Amazon S3 location for the pointwise model diagnostics for an Amazon Lookout for Equipment model.

Contents

Bucket

The name of the Amazon S3 bucket where the pointwise model diagnostics are located. You must be the owner of the Amazon 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

Prefix

The Amazon S3 prefix for the location of the pointwise model diagnostics. The prefix specifies the folder and evaluation result file name. (`bucket`).

When you call `CreateModel` or `UpdateModel`, specify the path within the bucket that you want Lookout for Equipment to save the model to. During training, Lookout for Equipment creates the model evaluation model as a compressed JSON file with the name `model_diagnostics_results.json.gz`.

When you call `DescribeModel` or `DescribeModelVersion`, `prefix` contains the file path and filename of the model evaluation file.

Type: String

Length Constraints: Minimum length of 0. Maximum length of 1024.

Pattern: `(^$)|([\u0009\u000A\u000D\u0020-\u00FF]{1,1023}/$)`

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 Java V2
- AWS SDK for Ruby V3

ModelSummary

Provides information about the specified machine learning model, including dataset and model names and ARNs, as well as status.

Contents

ActiveModelVersion

The model version that the inference scheduler uses to run an inference execution.

Type: Long

Valid Range: Minimum value of 1.

Required: No

ActiveModelVersionArn

The Amazon Resource Name (ARN) of the model version that is set as active. The active model version is the model version that the inference scheduler uses to run an 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\/[0-9a-zA-Z_-]{1,200}\/.+\/model-version\/[0-9]{1,}$`

Required: No

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\/[0-9a-zA-Z_-]{1,200}\/.+`

Required: No

DatasetName

The name of the dataset being used for the machine learning model.

Type: String

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

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

Required: No

LatestScheduledRetrainingModelVersion

Indicates the most recent model version that was generated by retraining.

Type: Long

Valid Range: Minimum value of 1.

Required: No

LatestScheduledRetrainingStartTime

Indicates the start time of the most recent scheduled retraining run.

Type: Timestamp

Required: No

LatestScheduledRetrainingStatus

Indicates the status of the most recent scheduled retraining run.

Type: String

Valid Values: `IN_PROGRESS | SUCCESS | FAILED | IMPORT_IN_PROGRESS | CANCELED`

Required: No

ModelArn

The Amazon Resource Name (ARN) of the machine learning 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

ModelDiagnosticsOutputConfiguration

Output configuration information for the pointwise model diagnostics for an Amazon Lookout for Equipment model.

Type: ModelDiagnosticsOutputConfiguration object

Required: No

ModelName

The name of the machine learning model.

Type: String

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

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

Required: No

ModelQuality

Provides a quality assessment for a model that uses labels. If Lookout for Equipment determines that the model quality is poor based on training metrics, the value is `POOR_QUALITY_DETECTED`. Otherwise, the value is `QUALITY_THRESHOLD_MET`.

If the model is unlabeled, the model quality can't be assessed and the value of `ModelQuality` is `CANNOT_DETERMINE_QUALITY`. In this situation, you can get a model quality assessment by adding labels to the input dataset and retraining the model.

For information about using labels with your models, see Understanding labeling.

For information about improving the quality of a model, see Best practices with Amazon Lookout for Equipment.

Type: String

Valid Values: `QUALITY_THRESHOLD_MET | CANNOT_DETERMINE_QUALITY | POOR_QUALITY_DETECTED`

Required: No

NextScheduledRetrainingStartDate

Indicates the date that the next scheduled retraining run will start on. Lookout for Equipment truncates the time you provide to the nearest UTC day.

Type: Timestamp

Required: No

RetrainingSchedulerStatus

Indicates the status of the retraining scheduler.

Type: String

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

Required: No

Status

Indicates the status of the machine learning model.

Type: String

Valid Values: `IN_PROGRESS | SUCCESS | FAILED | IMPORT_IN_PROGRESS`

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 Java V2
- AWS SDK for Ruby V3

ModelVersionSummary

Contains information about the specific model version.

Contents

CreatedAt

The time when this model version was created.

Type: Timestamp

Required: No

ModelArn

The Amazon Resource Name (ARN) of the model that this model version is a version of.

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 model that this model version is a version of.

Type: String

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

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

Required: No

ModelQuality

Provides a quality assessment for a model that uses labels. If Lookout for Equipment determines that the model quality is poor based on training metrics, the value is `POOR_QUALITY_DETECTED`. Otherwise, the value is `QUALITY_THRESHOLD_MET`.

If the model is unlabeled, the model quality can't be assessed and the value of `ModelQuality` is `CANNOT_DETERMINE_QUALITY`. In this situation, you can get a model quality assessment by adding labels to the input dataset and retraining the model.

For information about improving the quality of a model, see Best practices with Amazon Lookout for Equipment.

Type: String

Valid Values: `QUALITY_THRESHOLD_MET | CANNOT_DETERMINE_QUALITY | POOR_QUALITY_DETECTED`

Required: No

ModelVersion

The version of the model.

Type: Long

Valid Range: Minimum value of 1.

Required: No

ModelVersionArn

The Amazon Resource Name (ARN) of the model version.

Type: String

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

Pattern: `^arn:aws(-[^:]+)?:lookoutequipment:[a-zA-Z0-9\-]*:[0-9]{12}:model\/[0-9a-zA-Z_-]{1,200}\/.+\/model-version\/[0-9]{1,}$`

Required: No

SourceType

Indicates how this model version was generated.

Type: String

Valid Values: `TRAINING | RETRAINING | IMPORT`

Required: No

Status

The current status of the model version.

Type: String

Valid Values: `IN_PROGRESS | SUCCESS | FAILED | IMPORT_IN_PROGRESS | CANCELED`

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 Java V2
- AWS SDK for Ruby V3

MonotonicValues

Entity that comprises information on monotonic values in the data.

Contents

Status

Indicates whether there is a potential data issue related to having monotonic values.

Type: String

Valid Values: `POTENTIAL_ISSUE_DETECTED | NO_ISSUE_DETECTED`

Required: Yes

Monotonicity

Indicates the monotonicity of values. Can be INCREASING, DECREASING, or STATIC.

Type: String

Valid Values: `DECREASING | INCREASING | STATIC`

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 Java V2
- AWS SDK for Ruby V3

MultipleOperatingModes

Entity that comprises information on operating modes in data.

Contents

Status

Indicates whether there is a potential data issue related to having multiple operating modes.

Type: String

Valid Values: `POTENTIAL_ISSUE_DETECTED | NO_ISSUE_DETECTED`

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 Java V2
- AWS SDK for Ruby V3

RetrainingSchedulerSummary

Provides information about the specified retraining scheduler, including model name, status, start date, frequency, and lookback window.

Contents

LookbackWindow

The number of past days of data used for retraining.

Type: String

Pattern: `^P180D$|^P360D$|^P540D$|^P720D$`

Required: No

ModelArn

The ARN of the model that the retraining scheduler is attached to.

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 model that the retraining scheduler is attached to.

Type: String

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

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

Required: No

RetrainingFrequency

The frequency at which the model retraining is set. This follows the ISO 8601 guidelines.

Type: String

Length Constraints: Minimum length of 1. Maximum length of 10.

Pattern: `^P(\dY)?(\d{1,2}M)?(\d{1,3}D)?$`

Required: No

RetrainingStartDate

The start date for the retraining scheduler. Lookout for Equipment truncates the time you provide to the nearest UTC day.

Type: Timestamp

Required: No

Status

The status of the retraining 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 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) 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 Java V2
- AWS SDK for Ruby V3

SensorStatisticsSummary

Summary of ingestion statistics like whether data exists, number of missing values, number of invalid values and so on related to the particular sensor.

Contents

CategoricalValues

Parameter that describes potential risk about whether data associated with the sensor is categorical.

Type: CategoricalValues object

Required: No

ComponentName

Name of the component to which the particular sensor belongs for which the statistics belong to.

Type: String

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

Pattern: `^[0-9a-zA-Z._\-]{1,200}$`

Required: No

DataEndTime

Indicates the time reference to indicate the end of valid data associated with the sensor that the statistics belong to.

Type: Timestamp

Required: No

DataExists

Parameter that indicates whether data exists for the sensor that the statistics belong to.

Type: Boolean

Required: No

DataStartTime

Indicates the time reference to indicate the beginning of valid data associated with the sensor that the statistics belong to.

Type: Timestamp

Required: No

DuplicateTimestamps

Parameter that describes the total number of duplicate timestamp records associated with the sensor that the statistics belong to.

Type: CountPercent object

Required: No

InvalidDateEntries

Parameter that describes the total number of invalid date entries associated with the sensor that the statistics belong to.

Type: CountPercent object

Required: No

InvalidValues

Parameter that describes the total number of, and percentage of, values that are invalid for the sensor that the statistics belong to.

Type: CountPercent object

Required: No

LargeTimestampGaps

Parameter that describes potential risk about whether data associated with the sensor contains one or more large gaps between consecutive timestamps.

Type: LargeTimestampGaps object

Required: No

MissingValues

Parameter that describes the total number of, and percentage of, values that are missing for the sensor that the statistics belong to.

Type: CountPercent object

Required: No

MonotonicValues

Parameter that describes potential risk about whether data associated with the sensor is mostly monotonic.

Type: MonotonicValues object

Required: No

MultipleOperatingModes

Parameter that describes potential risk about whether data associated with the sensor has more than one operating mode.

Type: MultipleOperatingModes object

Required: No

SensorName

Name of the sensor that the statistics belong to.

Type: String

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

Pattern: `^[0-9a-zA-Z:#$.\-_]{1,200}$`

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 Java V2
- AWS SDK for Ruby V3

SensorsWithShortDateRange

Entity that comprises information on sensors that have shorter date range.

Contents

AffectedSensorCount

Indicates the number of sensors that have less than 14 days of data.

Type: Integer

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 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 Java V2
- AWS SDK for Ruby V3

UnsupportedTimestamps

Entity that comprises information abount unsupported timestamps in the dataset.

Contents

TotalNumberOfUnsupportedTimestamps

Indicates the total number of unsupported timestamps across the ingested data.

Type: Integer

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 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

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 Signing AWS API requests in the *IAM User Guide*.

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 Create a signed AWS API request in the *IAM User Guide*.

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 Elements of an AWS API request signature in the *IAM User Guide*.

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 STS, see AWS services that work with IAM in the *IAM User Guide*.

Condition: If you're using temporary security credentials from AWS STS, 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 Create a signed AWS API request in the *IAM User Guide*.

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:** March 8th, 2024

Change	Description	Date
Bulk import resources.	You can now bulk import Amazon Lookout for Equipment resources from a source AWS account to a target AWS account.	March 8th, 2024
Model quality.	You can now get a quality assessment for an Amazon Lookout for Equipment model.	February 21st, 2024
Getting pointwise model diagnostics for a model.	You can now get pointwise model diagnostics for an Amazon Lookout for Equipment model.	February 14, 2024
Importing and versioning.	You can work with multiple versions of your model, and you can import versions from other accounts.	July 28, 2023
AmazonLookoutEquipmentReadOnlyAccess – Update to an existing policy	Lookout for Equipment modified the policy to allow all Describe actions and all List actions.	November 4, 2022

Change	Description	Date
AmazonLookoutEquipmentReadOnlyAccess – Update to an existing policy	Lookout for Equipment modified the policy so as to allow all list and describe APIs.	October 26, 2022
Schema detection	You no longer have to define a schema for your dataset in JSON. Lookout for Equipment will do it for you.	April 22, 2022
Ingestion validation	When you ingest a dataset, Lookout for Equipment helps you determine which sensors should be used in the building of your model.	April 22, 2022
Update to grant retirement policy	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