Amazon Fraud Detector

User Guide

Version latest

Amazon Fraud Detector: User Guide

Table of Contents

What is Amazon Fraud Detector? ... 1
- Benefits ... 1
- Core concepts and terms ... 2
- How Amazon Fraud Detector works ... 4
- Detecting fraud with Amazon Fraud Detector ... 4
- Accessing Amazon Fraud Detector ... 5
  - Availability ... 6
  - Interfaces ... 6
- Pricing ... 6

Set up for Amazon Fraud Detector ... 7
- Sign up for AWS ... 7
- Set up permissions to access Amazon Fraud Detector interfaces ... 7
- Set up interfaces to access Amazon Fraud Detector with ... 8
  - Access Amazon Fraud Detector console ... 8
  - Set up AWS CLI ... 9
  - Set up AWS SDK ... 9

Get started with Amazon Fraud Detector ... 10
- Get and upload example dataset ... 10
- Tutorial: Get started using the Amazon Fraud Detector console ... 11
  - Part A: Build, train, and deploy an Amazon Fraud Detector model ... 11
  - Part B: Generate fraud predictions ... 13
- Tutorial: Get started using the AWS SDK for Python (Boto3) ... 17
  - Prerequisites ... 17
  - Get started ... 17
  - (Optional) Explore the Amazon Fraud Detector APIs with a Jupyter (iPython) Notebook ... 23
- Next steps ... 24

Create event dataset ... 25
- Prepare event dataset ... 25
- Gather event data ... 26
- Dataset validation ... 30
- Dataset storage ... 30

Create event type ... 32
- Create an event type using the AWS SDK for Python (Boto3) ... 32

Create a model ... 33
- Choose a model type ... 33
  - Online fraud insights ... 33
  - Transaction fraud insights ... 34
- Choose storage location for your event dataset ... 36
  - Stored events ... 36
  - Store your event data with Amazon S3 ... 44
- Build a model ... 47
  - Train and deploy a model using the AWS SDK for Python (Boto3) ... 47
- Model scores ... 48
- Model performance metrics ... 49
- Model variable importance ... 50
  - Using model variable importance values ... 51
  - Evaluating model variable importance values ... 51
  - Viewing model variable importance ranking ... 51
  - Understanding how the model variable importance value is calculated ... 52
- Import a SageMaker model ... 52
  - Import a SageMaker model using the AWS SDK for Python (Boto3) ... 52

Create a detector ... 54
- Create a detector using the AWS SDK for Python (Boto3) ... 54
- Create a detector version ... 54

Rule execution mode 54
Create a detector version using the AWS SDK for Python (Boto3) 55
Create a rule 55
Expressions 56
Create a rule using the AWS SDK for Python (Boto3) 56
Rule language reference 57
Create resources 61
Create a variable 61
Data types 61
Default value 61
Variable types 62
Variable enrichments 67
Create a variable using the Amazon Fraud Detector console 72
Create a variable using the AWS SDK for Python (Boto3) 72
Create an outcome 73
Create an outcome using the AWS SDK for Python (Boto3) 73
Create an entity type 74
Create an entity type using the AWS SDK for Python (Boto3) 74
Create a label 74
Create a label using the AWS SDK for Python (Boto3) 74
Delete resources 76
Delete a detector, detector version, or rule version 77
Delete a model or model version 78
Delete an event or event type 79
Delete an entity type 80
Delete an outcome 80
Delete a label 81
Delete a variable 81
Delete stored event data 81
Delete individual stored event data 81
Delete imported event data 82
Manage resources using AWS CloudFormation 83
Creating Amazon Fraud Detector templates 83
Managing Amazon Fraud Detector stacks 83
Organizing your Amazon Fraud Detector stacks 83
Understanding Amazon Fraud Detector CloudFormation parameters 84
Sample AWS CloudFormation template for Amazon Fraud Detector resources 84
Learn more about AWS CloudFormation 85
Get fraud predictions 86
Real time prediction 87
How real time fraud prediction works 87
Getting real time fraud prediction 87
Batch predictions 88
How batch predictions work 88
Input and output files 88
Getting batch predictions 89
Guidance on IAM roles 89
Get batch fraud predictions using the AWS SDK for Python (Boto3) 90
Prediction explanations 90
Viewing prediction explanations 91
Understanding how prediction explanations are calculated 93
Security 94
Data Protection 94
Encryption at rest 95
Encryption in transit 95
Key management 95
VPC endpoints (AWS PrivateLink) 96

Opting out ... 98
Identity and access management ... 98
Audience ... 99
Authenticating with identities ... 99
Managing access using policies ... 101
How Amazon Fraud Detector works with IAM ... 102
Identity-based policy examples ... 105
Confused deputy prevention ... 110
Troubleshooting ... 111
Monitoring Amazon Fraud Detector ... 114
Compliance validation ... 114
Resilience ... 115
Infrastructure Security ... 115
Monitor Amazon Fraud Detector ... 116
Monitoring with CloudWatch ... 116
Using CloudWatch Metrics for Amazon Fraud Detector ... 116
Amazon Fraud Detector Metrics ... 118
Logging Amazon Fraud Detector API Calls with AWS CloudTrail ... 120
Amazon Fraud Detector Information in CloudTrail ... 121
Understanding Amazon Fraud Detector Log File Entries ... 121
Troubleshoot ... 123
Troubleshoot training data issues ... 123
Unstable fraud rate in the given dataset ... 123
Insufficient data ... 124
Missing or different EVENT_LABEL values ... 125
Missing or incorrect EVENT_TIMESTAMP values ... 126
Data not ingested ... 127
Insufficient variables ... 127
Missing or incorrect variable type ... 128
Missing variable values ... 128
Insufficient unique variable values ... 128
Incorrect variable expression ... 129
Insufficient unique entities ... 130
Quotas ... 131
Amazon Fraud Detector models ... 131
Amazon Fraud Detector detectors / variables / outcomes / rules ... 131
Amazon Fraud Detector API ... 132
Document history ... 133

What is Amazon Fraud Detector?

Amazon Fraud Detector is a fully managed fraud detection service that automates the detection of potentially fraudulent activities online. These activities include unauthorized transactions and the creation of fake accounts. Amazon Fraud Detector works by using machine learning to analyze your data. It does this in a way that builds off of the seasoned expertise of more than 20 years of fraud detection at Amazon.

You can use Amazon Fraud Detector to build customized fraud-detection models, add decision logic to interpret the model's fraud evaluations, and assign outcomes such as pass or send for review for each possible fraud evaluation. With Amazon Fraud Detector, you don't need machine learning expertise to detect fraudulent activities.

To get started, collect and prepare fraud data that you collected at your organization. Amazon Fraud Detector then uses this data to train, test, and deploy a custom fraud detection model on your behalf. As a part of this process, Amazon Fraud Detector uses machine learning models that have learned patterns of fraud from AWS and Amazon's own fraud expertise to evaluate your fraud data and generate model scores and model performance data. You configure decision logic to interpret the model's score and assign outcomes for how to deal with each fraud evaluation.

Benefits

Amazon Fraud Detector provides the following benefits. These benefits make it possible for you to detect fraud fast without needing to invest the time and resources that are traditionally required to build and maintain a fraud management system.

Automated fraud model creation

Amazon Fraud Detector's fraud detection models are fully automated machine learning models customized to meet your specific business needs. You can use Amazon Fraud Detector models to identify potential fraud in any online transactions such as new account creations, online payments, and guest checkout.

Because fraud models are created through an automated process, you can forgo many of the steps associated with creating and training a model. These steps include data validation and enrichment, feature engineering, algorithm selection, hyperparameter tuning, and model deployment.

To create a fraud detection model using Amazon Fraud Detector, you only upload your company's historical fraud dataset and select the model type. Then, Amazon Fraud Detector automatically finds the most suitable fraud detection algorithm for your use case and creates the model. You do not need to know coding or have machine learning expertise to create fraud detection models.

Fraud models that evolve and learn

Fraud detection models must constantly evolve to keep up with the changing fraud landscape. Amazon Fraud Detector does this automatically by calculating information including account age, time since last activity, and activity count. The result is that your model learns the difference between trusted customers who frequently make transactions and the continued attempts typical of fraudsters. This helps to maintain the performance of your model longer between retraining sessions.

Fraud model performance visualization

After your model is trained using the data that you provide, Amazon Fraud Detector validates your model performance. It also provides visual tools for you to assess the performance. For each model that you train, you can see the model performance score, the score distribution graph, the confusion matrix, the threshold table, and all of the inputs that you provided ranked by their impact on model

performance. Using these performance tools, you can learn how your model is performing and what inputs are driving your model performance. If required, you can tweak your model to improve its overall performance.

Fraud prediction

Amazon Fraud Detector generates fraud prediction for your organization's business activities. Fraud prediction is an evaluation of a business activity for fraud risk. Amazon Fraud Detector generates predictions using the prediction logic with the data that's associated with the activity. You provided this data when you created your fraud detection model. You can get fraud predictions for a single activity in real time or get fraud predictions offline for a set of activities.

Fraud prediction explanation visualization

Amazon Fraud Detector generates prediction explanations as part of the fraud prediction process. Prediction explanations provide insight into how each data element used to train your model has impacted your model's fraud prediction score. Prediction explanations are provided using the visual tools such as tables and graphs. You can use these tools to identify visually how much influence each data element has on the prediction scores. Then, you can use this information to analyze the fraud patterns across your data set and detect bias, if any. Last you can also use the prediction explanations to identify top risk indicators during a manual fraud investigation process. This helps you narrow down the root causes that lead to false positive predictions.

Rule-based actions

After your fraud detection model is trained you can add rules to take actions on the evaluated data, such as accept the data, send data for review, or collect more data. A rule is a condition that tells Amazon Fraud Detector how to interpret data during fraud prediction. For example, you can create a rule that flags suspicious customer accounts to be reviewed. You can set this rule to be initiated if both the detected model score is greater than your predetermined threshold and if the account payment's AUTH_Code isn't valid.

Core concepts and terms

The following is a list of core concepts and terms that are used in Amazon Fraud Detector:

Event

An event is your organization's business activity that's evaluated for fraud risk. Amazon Fraud Detector generates fraud predictions for events.

Label

A label classifies a single event as fraudulent or legitimate. Labels are used to train machine learning models in Amazon Fraud Detector.

Entity

An entity represents who is performing the event. You provide entity ID as part of your company's fraud data to indicate the specific entity who performed the event.

Event type

An event type defines the structure for an event sent to Amazon Fraud Detector. This includes the data sent as part of the event, the entity performing the event (such as a customer), and the labels that classify the event. Example event types include online payment transactions, account registrations, and authentication.

Entity type

An entity type classifies the entity. Example classifications include customer, merchant, or account.

Event dataset

The event dataset is your company's historical data of a specific business activity or an event. For example, your company's event might be online account registration. Data from a single event (registration) might include the associated IP address, email address, billing address, and event timestamp. You provide event dataset to Amazon Fraud Detector to create and train fraud detection models.

Model

A model is an output of machine learning algorithms. These algorithms are implemented in code and run on event data you provide.

Model type

The model type defines the algorithms, enrichments, and feature transformations that are used during model training. It also defines the data requirements to train the model. These definitions function to optimize your model for a specific type of fraud. You specify the model type to use when you create your model.

Model training

Model training is the process of using a provided event dataset to create a model that can predict fraudulent events. All steps in the model training process are fully automated. These steps include data validation, data transformation, feature engineering, algorithm selection, and model optimization.

Model score

Model score is the evaluation result of your company's historical fraud data. During the model training process, Amazon Fraud Detector evaluates the dataset for fraudulent activities and generates a score between 0 and 1000. For this score, 0 represents low fraud risk whereas 1000 represents the highest fraud risk. The score itself is directly related to false positive rate (FPR).

Model version

A model version is an output from training a model.

Model deployment

Model deployment is a process for activating a model version and making it available for generating fraud predictions.

Amazon SageMaker model endpoint

In addition to building models using Amazon Fraud Detector, you can optionally use SageMaker-hosted model endpoints in Amazon Fraud Detector evaluations.

For more information about building a model in SageMaker, see Train a Model with Amazon SageMaker.

Detector

A detector contains the detection logic such as the model and rules for a particular event that you want to evaluate for fraud. You create a detector using a model version.

Detector version

A detector can have multiple versions, with each version having a status of `Draft`, `Active`, or `Inactive`. Only one detector version can be in `Active` status at a time.

Variable

A variable represents a data element associated with an event that you want to use in a fraud prediction. Variables can either be sent with an event as part of a fraud prediction or derived, such as the output of an Amazon Fraud Detector model or Amazon SageMaker.

Rule

A rule is a condition that tells Amazon Fraud Detector how to interpret variable values during a fraud prediction. A rule consists of one or more variables, a logic expression, and one or more outcomes. The variables used in the rule must be part of the event dataset that the detector evaluates. Moreover each detector must have at least one rule associated with it.

Outcome

This is the result, or output, from a fraud prediction. Each rule that is used in a fraud prediction must specify one or more outcomes.

Fraud prediction

Fraud prediction is an evaluation of fraud for either a single event or a set of events. Amazon Fraud Detector generates fraud predictions for a single online event in real time by synchronously providing a model score and an outcome based on the rules. Amazon Fraud Detector generates fraud predictions for a set of events offline. You can use the predictions to perform an offline proof-of-concept, or to retrospectively evaluate fraud risk on an hourly, daily, or weekly basis.

Fraud prediction explanation

Fraud prediction explanations provide insight into how each variable impacted your model's fraud prediction score. It provides information about how each variable influences the risk scores in terms of magnitude (ranging from 0 to 5 with 5 being highest) and direction (driving the score higher or lower).

How Amazon Fraud Detector works

To generate fraud predictions, Amazon Fraud Detector uses machine learning models that are trained with historical fraud data that you provide. Each model is trained using a model type. A model type is a specialized recipe that's used to build a fraud detection model for a specific fraud use case. Deployed models are imported to detectors. There you can configure decision logic (for example, rules) to interpret the model's score and assign outcomes. Outcomes can be approve transaction, review transaction, or send transaction for further investigation.

Amazon Fraud Detector components include event dataset, models, detectors, rules, and outcomes. Using these components, you can build an evaluation that contains your fraud detection logic.

For information about your workflow for detecting fraud using Amazon Fraud Detector, see Detecting fraud with Amazon Fraud Detector (p. 4)

Detecting fraud with Amazon Fraud Detector

This section describes a typical workflow for detecting fraud with Amazon Fraud Detector. It also summarizes how you can accomplish those tasks. The following diagram provides a high-level view of the workflow for detecting fraud with Amazon Fraud Detector.

Fraud detection is a continuous process. After you deploy your model, make sure to evaluate its performance scores and metrics based on the prediction explanations. By doing so, you can identify top risk indicators, narrow down root causes that lead to false positives, and analyze fraud patterns across your dataset and detect bias, if any exist. To increase the accuracy of the predictions, you can tweak your dataset to include new or revised data. Then, you can retrain your model with the updated dataset. As more data becomes available, you continue retraining your model to increase accuracy.

Accessing Amazon Fraud Detector

Amazon Fraud Detector is available in multiple AWS Regions and can be accessed using AWS interfaces.

Availability

Amazon Fraud Detector is available in the US East (N. Virginia), US East (Ohio), US West (Oregon), Europe (Ireland), Asia Pacific (Singapore),and Asia Pacific (Sydney) AWS Regions.

Interfaces

You can create, train, deploy, test, run, and manage fraud detection models and detectors using any of the following interfaces:

AWS Management Console - Amazon Fraud Detector provides a web-based user interface, the Amazon Fraud Detector console. If you signed up for an AWS account, you can access the Amazon Fraud Detector console. For more information, see Set up Amazon Fraud Detector.

AWS Command Line Interface (AWS CLI) - Provides an interface that you can use to interact with a broad set of AWS services, including Amazon Fraud Detector, using commands in your command-line shell. AWS CLI commands for Amazon Fraud Detector implement functionality that's equivalent to that provided by the Amazon Fraud Detector console.

AWS SDK - Provides language-specific APIs and manage many of the connection details, such as signature calculation, request retry handling, and error handling. For more information, see AWS SDK.

AWS CloudFormation - Provides templates that you can use to define your Amazon Fraud Detector resources and properties. For more information, see Amazon Fraud Detector resource type reference in the AWS CloudFormation User Guide.

Pricing

With Amazon Fraud Detector, you pay only for what you use. There are no minimum fees or upfront commitments. You're charged based on the compute hours that are used to train and host your models, the amount of storage you use, and the quantity of fraud predictions that you make. For more information, see Amazon Fraud Detector pricing.

Set up for Amazon Fraud Detector

To use Amazon Fraud Detector, you first need an Amazon Web Services (AWS) account and then you must set up the permissions that give your AWS account access to all the interfaces. You also must grant permissions that allow Amazon Fraud Detector to access your account to perform tasks on your behalf and to access resources that you own.

Complete the following tasks in this section to get set up for using Amazon Fraud Detector:

- Sign up for AWS
- Set up permissions that allow access to Amazon Fraud Detector interfaces
- Set up interfaces you want to use to access Amazon Fraud Detector

After you complete these steps, see Get started with Amazon Fraud Detector (p. 10) to continue getting started with Amazon Fraud Detector.

Sign up for AWS

When you sign up for Amazon Web Services (AWS), your AWS account is automatically signed up for all services on AWS, including Amazon Fraud Detector. You're charged only for the services that you use. If you already have an AWS account, skip to the next task.

To create an AWS account

1. Go to https://aws.amazon.com and choose **Create an AWS account**.
2. Follow the on-screen instructions to create the account. Note your **12-digit AWS account number**. You use it later.

Set up permissions to access Amazon Fraud Detector interfaces

To use Amazon Fraud Detector, set up permissions to access the Amazon Fraud Detector console and API operations. To perform tasks on your behalf and to access resources that you own, grant Amazon Fraud Detector the necessary permissions.

Following security best practice create an AWS Identity and Access Management (IAM) user with access restricted to Amazon Fraud Detector operations and with required permissions. You can add other permissions as needed.

The following policies provide the required permission to use Amazon Fraud Detector:

- `AmazonFraudDetectorFullAccessPolicy`

  Allows you to perform the following actions:

  - Access all Amazon Fraud Detector resources
  - List and describe all model endpoints in SageMaker

  - List all IAM roles in the account
  - List all Amazon S3 buckets
  - Allow IAM Pass Role to pass a role to Amazon Fraud Detector
- `AmazonS3FullAccess`

  Allows full access to Amazon Simple Storage Service. This is required if you need to upload training datasets to Amazon S3.

The following describes how to create an IAM user and assign the needed permissions.

To create an IAM user and assign required permissions

1. Sign in to the AWS Management Console and open the IAM console at https://console.aws.amazon.com/iam/.
2. In the navigation pane, choose **Users** and then choose **Add user**.
3. For **User name**, enter **`AmazonFraudDetectorUser`**.
4. Select the **AWS Management Console access** check box, and then configure the user password.
5. (Optional) By default, AWS requires the new user to create a new password when they first sign in. You can clear the check box next to **User must create a new password at next sign-in** to allow the new user to reset their password after they sign in.
6. Choose **Next: Permissions**.
7. Choose **Create group**.
8. For **Group name** enter **`AmazonFraudDetectorGroup`**.
9. In the policy list, select the check box for **AmazonFraudDetectorFullAccessPolicy** and **AmazonS3FullAccess**. Choose **Create group**.
10. In the list of groups, select the check box for your new group. Choose **Refresh** if you don't see the group in the list.
11. Choose **Next: Tags**.
12. (Optional) Add metadata to the user by attaching tags as key-value pairs. For instructions on how to use tags in IAM, see Tagging IAM Users and Roles.
13. Choose **Next: Review** to see the **User details** and **Permissions summary** for the new user. When you're ready to proceed, choose **Create user**.

Set up interfaces to access Amazon Fraud Detector with

You can access Amazon Fraud Detector using the Amazon Fraud Detector console, AWS CLI, or AWS SDK. Before you can use them, first set up the AWS CLI and AWS SDK.

Access Amazon Fraud Detector console

You can access the Amazon Fraud Detector console and other AWS services through the AWS Management Console. Your AWS account, grants you access to the AWS Management Console.

To access Amazon Fraud Detector console,

1. Go to https://console.aws.amazon.com and sign in to your AWS account.
2. Navigate to Amazon Fraud Detector.

With Amazon Fraud Detector console, you can create and manage your models and your fraud detection resources such as Detectors, Variables, Events, Entities, Labels, and Outcomes. You can generate predictions and evaluate the performance and predictions of your model.

Set up AWS CLI

You can use AWS Command Line Interface (AWS CLI) to interact with Amazon Fraud Detector by running commands in your command line shell. With minimal configuration, you can use the AWS CLI to run commands for similar functionality to that provided by the Amazon Fraud Detector console from the command prompt in your terminal.

To set up the AWS CLI

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

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

For information about Amazon Fraud Detector commands, see Available Commands

Set up AWS SDK

You can use the AWS SDKs to write code for creating and managing your fraud detection resources and for getting fraud predictions. The AWS SDKs support Amazon Fraud Detector in Java and Python.

To set up AWS SDK for Python (Boto3)

You can use AWS SDK for Python (Boto3) to create, configure, and manage AWS services. For instructions on how to install Boto, see AWS SDK for Python (Boto3). Make sure that you're using Boto3 SDK version 1.14.29 or higher.

After you install AWS SDK for Python (Boto3), run the following Python example to confirm that your environment is configured correctly. If it's configured correctly, the response contains a list of detectors. If no detectors were created, the list is empty.

```
import boto3
fraudDetector = boto3.client('frauddetector')

response = fraudDetector.get_detectors()
print(response)
```

To set up AWS SDKs for Java

For instructions on how to install and load the AWS SDK for JavaScript, see Setting up the SDK for JavaScript.

Get started with Amazon Fraud Detector

Before you get started, make sure that you have read Detecting fraud with Amazon Fraud Detector (p. 4) and completed steps in Set up for Amazon Fraud Detector (p. 7).

Use the hands-on tutorials in this section to learn how you can use Amazon Fraud Detector to build, train, and deploy a fraud detection model using one of the following interfaces.

- The Amazon Fraud Detector Console
- The AWS SDK for Python (Boto3)

For this tutorial, you assume the role of a fraud analyst using machine learning model to predict whether a new account registration is fraudulent. The model must be trained using data from account registrations. Amazon Fraud Detector provides an example account registration dataset for this tutorial. You must first get and upload the dataset for building and training the model.

Get and upload example dataset

The example dataset you use in this tutorial provides details of online account registrations. The dataset is in a text file that uses comma-separated value (CSV) in the UTF-8 format. The first row of the CSV dataset file contains the headers. The header row is followed by multiple rows of data. Each of these rows consists of data elements from a single account registration. The data is labeled for your convenience. A column in the dataset identifies whether the account registration is fraudulent.

To get and upload example dataset

1. Download the zip file Training _Data.zip to your computer.

   Extract the zip file on your computer. There are two data files. The file `registration_data_20K_minimum` contains only two variables: *ip_address* and *email_address*. The file `registration_data_20K_full` contains other variables. These variables are for each event. These variables include *billing_address*, *phone_number*, and *user_agent*. Both data files also contain two mandatory fields:

   - EVENT_TIMESTAMP – Defines when the event occurred
   - EVENT_LABEL – Classifies the event as fraudulent or legitimate

   You can use either one of the two files for this tutorial.
2. Create an Amazon Simple Storage Service (Amazon S3) bucket.

   In this step, you create an external storage to store the dataset. This external storage is Amazon S3 bucket. For more information about Amazon S3, see What is Amazon S3?

   a. Sign in to the AWS Management Console and open the Amazon S3 console at https://console.aws.amazon.com/s3/.
   b. In **Buckets**, choose **Create bucket**.
   c. For **Bucket name**, enter a bucket name. Make sure that you follow the bucket naming rules in the console, and provide a globally unique name. We recommend you use a name that describes the bucket's purpose.

d. For **AWS Region**, choose the AWS Region where you want to create your bucket. The Region that you choose must support Amazon Fraud Detector. To reduce latency, choose the AWS Region that's closest to your geographic location. For a list of Regions that support Amazon Fraud Detector, see the Region Table in the *Global Infrastructure Guide*.

e. Leave the default settings for **Object Ownership**, **Bucket settings for Block Public Access**, **Bucket Versioning**, and **Tags** for this tutorial.

f. For **Default encryption**, choose **Disable** for this tutorial.

g. Review your bucket configuration, and then choose **Create bucket**.

3. Upload example data file to Amazon S3 bucket.

   Now that you have a bucket, upload one of the example files that you downloaded previously to the Amazon S3 bucket that you just created.

   a. In the **Buckets**, your bucket name is listed. Choose your bucket.

   b. Choose **Upload**.

   c. In **Files and folders**, choose **Add files**.

   d. Choose one of the example data files that you downloaded on your computer, and then choose **Open**.

   e. Leave the default settings for **Destination**, **Permissions**, and **Properties**.

   f. Review configurations, and then choose **Upload**.

   g. The example data file is uploaded to Amazon S3 bucket. Make a note of the bucket location. In the **Objects**, choose the example data file that you just uploaded.

   h. In the **Object overview**, copy the location under **S3 URI**. This is the Amazon S3 location of your example data file. You use it later. You can additionally copy the **Amazon Resource Name (ARN)** of your S3 bucket and save it.

Tutorial: Get started using the Amazon Fraud Detector console

This tutorial consists of two parts. The first part describes how to build, train, and deploy a fraud detection model. The second part covers how to use the model to generate fraud predictions in real time. The model is trained using the example data file that you upload to an S3 bucket. By the end of this tutorial, you complete the following actions:

- Build and train an Amazon Fraud Detector model
- Generate real-time fraud predictions

Part A: Build, train, and deploy an Amazon Fraud Detector model

In part A, you define your event, build a model, train the model, evaluate model's performance, and deploy the model.

Step 1: Create event type

- In this step, you define the business activity (event) to evaluate for fraud. Defining the event involves setting the variables that are in your dataset, the entity initiating event, and the labels that classify the event. For this tutorial, you define the account registration event.

a. Open the AWS Management Console and sign in to your account. Navigate to Amazon Fraud Detector.

b. In the left navigation pane, choose **Events**.

c. In the **Events type** page, choose **Create**.

d. Under **Event type details**, enter `sample_registration` as the event type name and, optionally, enter a description of the event.

e. For **Entity**, choose **Create entity**.

f. In the **Create entity** page, enter `sample_customer` as the entity type name. Optionally, enter a description of the entity type.

g. Choose **Create entity**.

h. Under **Event variables**, for **Choose how to define this event's variables**, choose **Select variables from a training dataset**.

i. For **IAM role**, choose **Create IAM role**.

j. In the **Create IAM role** page, enter the name of the S3 bucket that you uploaded your example data to and choose **Create role**.

k. In **Data location**, enter the path to your example data. This is the `S3 URI` path that you saved after uploading the example data. The path is similar to this: `S3://`*`your-bucket-name`*`/`*`example dataset filename`*`.csv`.

l. Choose **Upload**.

Amazon Fraud Detector extracts the headers from your example data file and maps them with a variable type. The mapping is displayed in the console.

m. Under **Labels - optional**, for **Labels**, choose **Create new labels**.

n. In **Create label** page, enter `fraud` as the name. This label corresponds to the value that represents the fraudulent account registration in the example dataset.

o. Choose **Create label**.

p. Create a second label, then enter `legit` as the name. This label corresponds to the value that represents the legitimate account registration in the example dataset.

q. Choose **Create event type**.

Step 2: Create model

1. On the **Models** page, choose **Add model**, and then choose **Create model**.
2. For **Step 1 – Define model details**, enter `sample_fraud_detection_model` as the model name. Optionally, add a description of the model.
3. For **Model Type**, choose the **Online Fraud Insights** model.
4. For **Event type**, choose **sample_registration**. This is the event type that you created in Step 1.
5. In **Historical event data**,
   a. In **Event data source**, choose **Event data stored in S3**.
   b. For **IAM role**, select the role that you created in Step 1.
   c. In **Training data location**, enter the S3 URI path to your example data file.
6. Choose **Next**.

Step 3: Train model

1. In **Model inputs**, leave all checkboxes checked. By default, Amazon Fraud Detector uses all variables from your historical event dataset as model inputs.

2. In **Label classification**, for **Fraud labels** choose **fraud** as this label corresponds to the value that represents fraudulent events in the example dataset. For **Legitimate labels**, choose **legit** as this label corresponds to the value that represents legitimate events in the example dataset.
3. Choose **Next**.
4. After reviewing, choose **Create and train model**. Amazon Fraud Detector creates a model and begins to train a new version of the model.

   In **Model versions** the **Status** column indicates the status of model training. Model training that uses the example dataset takes approximately 45 minutes to complete. The status changes to **Ready to deploy** after model training is complete.

Step 4: Review model performance

An important step in using Amazon Fraud Detector is to assess the accuracy of your model using model scores and performance metrics. After model training is complete, Amazon Fraud Detector validates model performance using the 15% of your data that wasn't used to train the model and generates a model performance score and other performance metrics.

1. To view model's performance,

   a. In the left navigation pane of the Amazon Fraud Detector console, choose **Models**.
   b. In the **Models** page, choose the model that you just trained (**sample_fraud_detection_model**), and then choose **1.0**. This is the version Amazon Fraud Detector created of your model.
2. Look at the **Model performance** overall score and all other metrics that Amazon Fraud Detector generated for this model.

   To learn more about the model performance score and performance metrics on this page, see Model scores (p. 48) and Model performance metrics (p. 49).

   You can expect all your trained Amazon Fraud Detector models to have real-world fraud detection performance metrics that are similar to the performance metrics that you see for the model in this tutorial.

Step 5: Deploy model

After you reviewed the performance metrics of your trained model and are ready to use it generate fraud predictions, you can deploy the model.

1. In left navigation pane of the Amazon Fraud Detector console, choose **Models**.
2. In the **Models** page, choose **sample_fraud_detection_model**, and then choose the specific model version that you want to deploy. For this tutorial, choose **1.0**.
3. On the **Model version** page, choose **Actions** and then choose **Deploy model version**.
4. In the **Model versions**, the **Status** shows the status of the deployment. The status changes to **Active** after the deployment completes. This indicates that the model version is activated and available to generate fraud predictions. Continue with Part B: Generate fraud predictions (p. 13) to complete steps for generating fraud predictions.

Part B: Generate fraud predictions

Fraud prediction is an evaluation of fraud for a business activity (event). Amazon Fraud Detector uses detectors to generate fraud prediction. A detector contains detection logic, such as models and rules, for a specific event that you want to evaluate for fraud. Detection logic uses rules to tell Amazon Fraud Detector how to interpret the data associated with the model. In this tutorial, you evaluate the account registration event using the account registration example dataset that you uploaded earlier.

In Part A, you created, trained, and deployed your model. In Part B, you build a detector for the `sample_registration` event type, add the deployed model, create rules and a rule execution order, and then create and activate a version of the detector that you use to generate fraud predictions.

Step 1: Build detector

To create detector

1. In the left navigation pane of the Amazon Fraud Detector console, choose **Detectors**.
2. Choose **Create detector**.
3. In the **Define detector details** page, enter `sample_detector` for detector name. Optionally, enter a description for the detector, such as `my sample fraud detector`.
4. For **Event Type**, select **sample_registration**. This is the event that you created in Part A of this tutorial.
5. Choose **Next**.

Step 2: Add model

If you completed Part A of this tutorial, then you likely already have an Amazon Fraud Detector model that's available to add to your detector. If you didn't already create a model, go to Part A and complete the steps to create, train, and deploy a model and then continue with Part B.

1. In the **Add model - optional**, choose **Add Model**.
2. In the **Add model** page, for **Select model**, choose the Amazon Fraud Detector model name that you deployed earlier. For **Select version**, choose the model version of the deployed model.
3. Choose **Add model**.
4. Choose **Next**.

Step 3: Add rules

A rule is a condition that tells Amazon Fraud Detector how to interpret model performance score when evaluating for fraud prediction. For this tutorial, you create three rules: `high_fraud_risk`, `medium_fraud_risk`, and `low_fraud_risk`.

1. In the **Add rules** page, under **Define a rule**, enter `high_fraud_risk` for the rule name and under **Description - optional**, enter **`This rule captures events with a high ML model score`** as the description for the rule.
2. In **Expression**, enter the following rule expression using the Amazon Fraud Detector simplified rule expression language:

   ```
   $sample_fraud_detection_model_insightscore > 900
   ```
3. In **Outcomes**, choose **Create a new outcome**. An outcome is the result from a fraud prediction and is returned if the rule matches during an evaluation.
4. In **Create a new outcome**, enter `verify_customer` as the outcome name. Optionally, enter a description.
5. Choose **Save outcome**.
6. Choose **Add rule** to run the rule validation checker and save the rule. After it's created, Amazon Fraud Detector makes the rule available for use in your detector.
7. Choose **Add another rule**, and then choose the **Create rule** tab.
8. Repeat this process twice more to create your `medium_fraud_risk` and `low_fraud_risk` rules using the following rule details:

   - medium_fraud_risk

Rule name: `medium_fraud_risk`

Outcome: `review`

Expression:

```
$sample_fraud_detection_model_insightscore <= 900 and

$sample_fraud_detection_model_insightscore > 700
```

- low_fraud_risk

  Rule name: `low_fraud_risk`

  Outcome: `approve`

  Expression:

  ```
  $sample_fraud_detection_model_insightscore <= 700
  ```

These values are examples used for this tutorial. When you create rules for your own detector, use values that are appropriate for your model and your use case,

9. After you have created all three rules, choose **Next**.

   For more information about creating and writing rules, see Create a rule (p. 55) and Rule language reference (p. 57).

Step 4: Configure rule execution and rule order

The rule execution mode for the rules that are included in the detector determines if all the rules you define are evaluated, or if rule evaluation stops at the first matched rule. And the rule order determines the order that you want the rule to be run in.

The default rule execution mode is `FIRST_MATCHED`.

First matched

First matched rule execution mode returns the outcomes for the first matching rule based on defined rule order. If you specify `FIRST_MATCHED`, Amazon Fraud Detector evaluates rules sequentially, first to last, stopping at the first matched rule. Amazon Fraud Detector then provides the outcomes for that single rule.

The order that you run rules in can affect the resulting fraud prediction outcome. After you have created your rules, re-order the rules to run them in the desired order by following these steps:

If your `high_fraud_risk` rule isn't already on the top of your rule list, choose **Order**, and then choose **1**. This moves `high_fraud_risk` to the first position.

Repeat this process so that your `medium_fraud_risk` rule is in the second position and your `low_fraud_risk` rule is in the third position.

All matched

All matched rule execution mode returns outcomes for all matched rules, regardless of rule order. If you specify `ALL_MATCHED`, Amazon Fraud Detector evaluates all rules and returns the outcomes for all matched rules.

Select `FIRST_MATCHED` for this tutorial and then choose **Next**.

Step 5: Review and create detector version

A detector version defines the specific models and rules that are used for generating fraud predictions.

1. In the **Review and create** page, review the detector details, models, and rules that you configured. If you need to make any changes, choose **Edit** next to the corresponding section.
2. Choose **Create detector**. After it's created, the first version of your detector appears in the Detector versions table with `Draft` status.

   You use the **Draft** version to test your Detector.

Step 6: Test and activate detector version

In the Amazon Fraud Detector console, you can test the logic of your detector using a mock data with the **Run test** feature. For this tutorial, you can use account registration data from the example dataset.

1. Scroll to **Run test** at the bottom of the **Detector version details** page.
2. For **Event metadata**, enter a timestamp of when the event occurred and enter a unique identifier for the entity performing the event. For this tutorial, select a date from the date picker for timestamp, and enter "1234" for the Entity ID.
3. For **Event variable**, enter the variable values that you want to test. For this tutorial, you only need the `ip_address` and `email_address` fields. This is because they are the inputs that are used to train your Amazon Fraud Detector model. You can use the following example values. This assumes that you used the suggested variable names:

   - ip_address: `205.251.233.178`
   - email_address: `johndoe@exampledomain.com`
4. Choose **Run test**.
5. Amazon Fraud Detector returns the fraud prediction outcome based on the rule execution mode. If the rule execution mode is `FIRST_MATCHED`, the returned outcome corresponds to the first rule that matched. The first rule is the rule with the highest priority. It's matched if it's evaluated as true. If the rule execution mode is `ALL_MATCHED`, the returned outcome corresponds to all rules that matched. That means that they're all evaluated to be true. Amazon Fraud Detector also returns the model score for any models added to your detector.

   You can change the inputs and run couple of tests to see different outcomes. You can use the *ip_address* and *email_address* values from your example dataset for the tests and check if the outcomes are as expected.
6. When you're satisfied with how the detector is working, promote it from `Draft` to `Active`. Doing so makes the detector available for use in real-time fraud detection.

   On the **Detector version details** page, choose **Actions**, **Publish**, **Publish version**. This changes the status of the detector from **Draft** to **Active.**

   At this point, your model and the associated detector logic are ready to evaluate online activities for fraud in real time using the Amazon Fraud Detector `GetEventPrediction` API. You can also evaluate events offline using a CSV input file and the `CreateBatchPredictionJob` API. For more information about fraud prediction, see Get fraud predictions (p. 86)

By completing this tutorial, you did the following:

- Uploaded an example event dataset to Amazon S3.
- Created and trained an Amazon Fraud Detector fraud detection model using the example dataset.
- Viewed the model performance score and other performance metrics that Amazon Fraud Detector generated.

- Deployed the fraud detection model.
- Created a detector and added the deployed model.
- Added rules, the rule execution order, and outcomes to the detector.
- Tested the detector by providing different inputs and checking if the rules and rule execution order worked as expected.
- Activated the detector by publishing it.

Tutorial: Get started using the AWS SDK for Python (Boto3)

This tutorial describes how to build and train an Amazon Fraud Detector model and then using this model to generate real-time fraud predictions using the AWS SDK for Python (Boto3). The model is trained using the account registration example data file that you upload to Amazon S3 bucket.

By the end of this tutorial, you complete the following actions:

- Build and train an Amazon Fraud Detector model
- Generate real-time fraud predictions

Prerequisites

The following are prerequisite steps for this tutorial.

- Complete Set up for Amazon Fraud Detector (p. 7).

  Make sure that you're using Boto3 SDK version 1.14.29 or higher.
- Complete Get and upload example dataset (p. 10) to upload the example dataset file that's required for this tutorial.

Get started

Step 1: Set up and verify your Python environment

Boto is the Amazon Web Services (AWS) SDK for Python. You can use it to create, configure, and manage AWS services. For instructions on how to install Boto3, see AWS SDK for Python (Boto3).

After you install AWS SDK for Python (Boto3), run the following Python example command to confirm that your environment is configured correctly. If your environment is configured correctly, the response contains a list of detectors. If no detectors were created, the list is empty.

```
import boto3
fraudDetector = boto3.client('frauddetector')

response = fraudDetector.get_detectors()
print(response)
```

Step 2: Create variables, entity type, and labels

In this step, you create resources that are used to define model, event, and rules.

Create variable

A variable is a data element from your dataset that you want to use to create event type, model, and rules.

In the following example,the CreateVariable API is used to create two variables. The variables are `email_address` and `ip_address`. Assign them to the corresponding variable types: `EMAIL_ADDRESS` and `IP_ADDRESS`. These variables are part of the example dataset you uploaded. When you specify the variable type, Amazon Fraud Detector interprets the variable during model training and when getting predictions. Only variables with an associated variable type can be used for model training.

```
import boto3
fraudDetector = boto3.client('frauddetector')

 #Create variable email_address
fraudDetector.create_variable(
    name = 'email_address',
    variableType = 'EMAIL_ADDRESS',
    dataSource = 'EVENT',
    dataType = 'STRING',
    defaultValue = '<unknown>'
)

#Create variable ip_address
fraudDetector.create_variable(
    name = 'ip_address',
    variableType = 'IP_ADDRESS',
    dataSource = 'EVENT',
    dataType = 'STRING',
    defaultValue = '<unknown>'
)
```

Create entity type

An entity represents who is performing the event and an entity type classifies the entity. Example classifications include *customer*, *merchant*, or *account*.

In the following example, PutEntityType API is used to create a `sample_customer` entity type.

```
import boto3
fraudDetector = boto3.client('frauddetector')

fraudDetector.put_entity_type(
   name = 'sample_customer',
   description = 'sample customer entity type'
)
```

Create label

A label classifies an event as fraudulent or legitimate and is used to train the fraud detection model. The model learns to classify events using these label values.

In the following example, the Putlabel API is used to create two labels, `fraud` and `legit`.

```
import boto3
fraudDetector = boto3.client('frauddetector')

fraudDetector.put_label(
```

```
    name = 'fraud',
    description = 'label for fraud events'
)

fraudDetector.put_label(
    name = 'legit',
    description = 'label for legitimate events'
)
```

Step 3: Create event type

With Amazon Fraud Detector, you build models that evaluate risks and generate fraud predictions for individual events. An event type defines the structure of an individual event.

In the following example, the PutEventType API is used to create an event type `sample_registration`. You define the event type by specifying the variables (`email_address`,`ip_address`), entity type (`sample_customer`), and labels (`fraud`, `legit`) that you created in the previous step.

```
import boto3
fraudDetector = boto3.client('frauddetector')

fraudDetector.put_event_type (
     name = 'sample_registration',
     eventVariables = ['ip_address', 'email_address'],
     labels = ['legit', 'fraud'],
     entityTypes = ['sample_customer'])
```

Step 4: Create, train, and deploy model

Amazon Fraud Detector trains models to learn to detect fraud for a specific event type. In the previous step, you created the event type. In this step, you create and train a model for the event type. The model acts as a container for your model versions. Each time you train a model, a new version is created.

Use following example codes to create and train an Online Fraud Insights model. This model is called `sample_fraud_detection_model`. It's for the event type `sample_registration` using the account registration example dataset that you uploaded to Amazon S3.

For more information about different model types that Amazon Fraud Detector supports, see Choose a model type (p. 33).

Create a model

In the following example, the CreateModel API is used to create a model.

```
import boto3
fraudDetector = boto3.client('frauddetector')

fraudDetector.create_model (
       modelId = 'sample_fraud_detection_model',
       eventTypeName = 'sample_registration',
       modelType = 'ONLINE_FRAUD_INSIGHTS')
```

Train a model

In the following example, the CreateModelVersion API is used to train the model. Specify `'EXTERNAL_EVENTS'` for the `trainingDataSource` and the Amazon S3 location where you stored your example dataset and the *RoleArn* of the Amazon S3 bucket for `externalEventsDetail`. For

`trainingDataSchema` parameter, specify how Amazon Fraud Detector interprets the example data. More specifically, specify which variables to include and how to classify the event labels.

```
import boto3
fraudDetector = boto3.client('frauddetector')

fraudDetector.create_model_version (
         modelId = 'sample_fraud_detection_model',
         modelType = 'ONLINE_FRAUD_INSIGHTS',
         trainingDataSource = 'EXTERNAL_EVENTS',
         trainingDataSchema = {
            'modelVariables' : ['ip_address', 'email_address'],
            'labelSchema' : {
               'labelMapper' : {
                   'FRAUD' : ['fraud'],
                   'LEGIT' : ['legit']
        }
    }
},
         externalEventsDetail = {
              'dataLocation' : 's3://your-S3-bucket-name/your-example-data-filename.csv',
              'dataAccessRoleArn' : 'role_arn'
}
)
```

You can train your model multiple times. Each time that you train a model, a new version is created. After model training is complete, the model version status updates to `TRAINING_COMPLETE`. You can review the model performance score and other model performance metrics.

Review model performance

An important step in using Amazon Fraud Detector is to assess the accuracy of your model using model scores and performance metrics. After model training is complete, Amazon Fraud Detector validates model performance using the 15% of your data that wasn't used to train the model. It generates a model performance score and other performance metrics.

Use the DescribeModelVersions API to review model performance. Look at the **Model performance** overall score and all other metrics generated by Amazon Fraud Detector for this model.

To learn more about the model performance score and performance metrics, see Model scores (p. 48) and Model performance metrics (p. 49).

You can expect all your trained Amazon Fraud Detector models to have real-world fraud detection performance metrics, which are similar to the metrics in this tutorial.

Deploy a model

After you reviewed the performance metrics of your trained model, deploy the model and make it available to Amazon Fraud Detector to generate fraud predictions. To deploy the trained model, use the UpdateModelVersionStatus API. In the following example, it's used to update the model version status to ACTIVE.

```
import boto3
fraudDetector = boto3.client('frauddetector')

fraudDetector.update_model_version_status (
     modelId = 'sample_fraud_detection_model',
     modelType = 'ONLINE_FRAUD_INSIGHTS',
     modelVersionNumber = '1.00',
     status = 'ACTIVE'
)
```

Step 5: Create detector, outcomes, rules, and detector version

A detector contains the detection logic, such as the models and rules. This logic is for a particular event that you want to evaluate for fraud. A rule is a condition that you specify to tell Amazon Fraud Detector how to interpret variable values during prediction. And outcome is the result of a fraud prediction. A detector can have multiple versions with each version having a status of *DRAFT*, *ACTIVE*, or *INACTIVE*. A detector version must have at least one rule that's associated with it.

Use the following example codes to create detector, rules, outcome, and to publish the detector.

Create a detector

In the following example, the PutDetector API is used to create a `sample_detector` detector for `sample_registration` event type.

```
import boto3
fraudDetector = boto3.client('frauddetector')

fraudDetector.put_detector (
     detectorId = 'sample_detector',
     eventTypeName = 'sample_registration'
)
```

Create outcomes

Outcomes are created for each possible fraud prediction result. In the following example, the PutOutcome API is used to create three outcomes - `verify_customer`, `review`, and `approve`. These outcomes are later assigned to rules.

```
import boto3
fraudDetector = boto3.client('frauddetector')

fraudDetector.put_outcome(
     name = 'verify_customer',
     description = 'this outcome initiates a verification workflow'
    )

fraudDetector.put_outcome(
     name = 'review',
     description = 'this outcome sidelines event for review'
    )

fraudDetector.put_outcome(
     name = 'approve',
     description = 'this outcome approves the event'
)
```

Create rules

Rule consists of one or more variables from your dataset, a logic expression, and one or more outcomes.

In the following example, the CreateRule API is used to create three different rules: `high_risk`, `medium_risk`, and `low_risk`. Create rule expressions to compare the model performance score `sample_fraud_detection_model_insightscore` value against various thresholds. This is to determine the level of risk for an event and assign outcome that was defined in the previous step.

```
import boto3
fraudDetector = boto3.client('frauddetector')

fraudDetector.create_rule(
     ruleId = 'high_fraud_risk',
     detectorId = 'sample_detector',
     expression = '$sample_fraud_detection_model_insightscore > 900',
     language = 'DETECTORPL',
     outcomes = ['verify_customer']
     )

fraudDetector.create_rule(
     ruleId = 'medium_fraud_risk',
     detectorId = 'sample_detector',
     expression = '$sample_fraud_detection_model_insightscore <= 900 and
 $sample_fraud_detection_model_insightscore > 700',
     language = 'DETECTORPL',
     outcomes = ['review']
     )

fraudDetector.create_rule(
     ruleId = 'low_fraud_risk',
     detectorId = 'sample_detector',
     expression = '$sample_fraud_detection_model_insightscore <= 700',
     language = 'DETECTORPL',
     outcomes = ['approve']
     )
```

Create a detector version

A detector version defines model and rules that are used to get fraud prediction.

In the following example, the CreateDetectorVersion API is used to create a detector version. It does this by providing model version details, rules, and a rule execution mode FIRST_MATCHED. A rule execution mode specifies the sequence for evaluating rules. The rule execution mode FIRST_MATCHED specifies that the rules are evaluated sequentially, first to last, stopping at first matched rule.

```
import boto3
fraudDetector = boto3.client('frauddetector')

fraudDetector.create_detector_version(
      detectorId = 'sample_detector',
      rules = [{
          'detectorId' : 'sample_detector',
          'ruleId' : 'high_fraud_risk',
          'ruleVersion' : '1'
},
{
          'detectorId' : 'sample_detector',
          'ruleId' : 'medium_fraud_risk',
          'ruleVersion' : '1'
},
{
          'detectorId' : 'sample_detector',
          'ruleId' : 'low_fraud_risk',
          'ruleVersion' : '1'
}
],
      modelVersions = [{
          'modelId' : 'sample_fraud_detection_model',
          'modelType': 'ONLINE_FRAUD_INSIGHTS',
```

```
          'modelVersionNumber' : '1.00'
}      ],
      ruleExecutionMode = 'FIRST_MATCHED'
)
```

Step 6: Generate fraud predictions

The last step of this tutorial uses the detector `sample_detector` created in the previous step to generate fraud predictions for `sample_registration` event type in real time. The detector evaluates the example data that's uploaded to Amazon S3. The response includes model performance scores as well as any outcomes that are associated to the matched rules.

In the following example, the GetEventPrediction API is used to provide data from a single account registration with each request. For this tutorial, take data (email_address and ip_address) from the account registration example data file. Each line (row) after the top header line represents data from a single account registration event.

```
import boto3
fraudDetector = boto3.client('frauddetector')

fraudDetector.get_event_prediction(
      detectorId = 'sample_detector',
      eventId = '802454d3-f7d8-482d-97e8-c4b6db9a0428',
      eventTypeName = 'sample_registration',
      eventTimestamp = '2020-07-13T23:18:21Z',
      entities = [{'entityType':'sample_customer', 'entityId':'12345'}],
 eventVariables = {
      'email_address': 'johndoe@exampledomain.com',
      'ip_address': '1.2.3.4'
}
)
```

After you completed this tutorial, you did the following:

- Uploaded an example event dataset to Amazon S3.
- Created variables, entities, and labels that are used to create and train a model.
- Created and trained a model using the example dataset.
- Viewed the model performance score and other performance metrics that Amazon Fraud Detector generated.
- Deployed the fraud detection model.
- Created a detector and added the deployed model.
- Added rules, the rule execution order, and outcomes to the detector.
- Created detector version.
- Tested the detector by providing different inputs and checking if the rules and rule execution order worked as expected.

(Optional) Explore the Amazon Fraud Detector APIs with a Jupyter (iPython) Notebook

For more examples for how to use the Amazon Fraud Detector APIs, see aws-fraud-detector-samples GitHub repository. The topics that the notebooks cover include both building models and detectors

using the Amazon Fraud Detector APIs and making batch fraud prediction requests using the `GetEventPrediction` API.

Next steps

Now that you created a model and a detector, you can take a deeper dive and start to create models and detectors and generate fraud predictions.

The following sections in the Amazon Fraud Detector User Guide describe how your business or organization can use Amazon Fraud Detector to detect fraud.

- Prepare and create your event dataset for training your model.
- Create event type
- Create model
- Create detector
- Get fraud predictions
- Manage your Amazon Fraud Detector resources (specifically, Variables, Entities, Outcomes, and Labels)
- Configure Amazon Fraud Detector to meet your security and compliance objectives
- Monitor Amazon Fraud Detector and log Amazon Fraud Detector API calls
- Troubleshoot issues with Amazon Fraud Detector

Create event dataset

An event dataset is the historical fraud data for your company. You provide this data to Amazon Fraud Detector to create fraud detection models.

Amazon Fraud Detector uses machine learning models for generating fraud predictions. Each model is trained using a model type. The model type specifies the algorithms and transformations that are used for training the model. Model training is the process of using a dataset that you provide to create a model that can predict fraudulent events. For more information, see How Amazon Fraud Detector works

The dataset used for creating fraud detection model provides details of an event. An event is a business activity that is evaluated for fraud risk. For example, an account registration can be an event. The data associated with the account registration event can be event dataset. Amazon Fraud Detector uses this dataset to evaluate account registration fraud.

Before you provide your dataset to Amazon Fraud Detector for creating a model, make sure to define your goal for creating the model. You also need to determine how you want to use the model and define your metrics for evaluating if the model is performing based on your specific requirements.

For example, your goals for creating a fraud detection model that evaluates account registration fraud can be the following:

- To auto-approve legitimate registrations.
- To capture fraudulent registrations for later investigation.

After you determined your goal, the next step is to decide how you want to use the model. Some examples for using fraud detection model to evaluate registration fraud are the following:

- For real-time fraud detection for each account registration.
- For offline evaluation of all account registrations every hour.

Some examples of metrics that can be used to measure the performance of the model include the following:

- Performs consistently better than the current baseline in production.
- Captures X% fraud registrations with Y% false positives rate.
- Accepts up to 5% of auto-approved registrations that are fraudulent.

Prepare event dataset

Amazon Fraud Detector requires that you provide your event dataset in a text file using comma-separated value (CSV) in the UTF-8 format. The first line of your CSV dataset file must contain file headers. The file header consists of event metadata and event variables that describe each data element that's associated with the event. The header is followed by event data. Each line consists of data elements from a single event.

Event metadata

We recommend that you provide the following metadata in your CSV file header. The event metadata must be in uppercase letters.

- EVENT_ID - A unique identifier for the event.
- ENTITY_TYPE - The entity that performs the event, such as a merchant or a customer.
- ENTITY_ID - An identifier for the entity performing the event.
- EVENT_TIMESTAMP - The timestamp when the event occurred. The timestamp must be in ISO 8601 standard in UTC.
- EVENT_LABEL- A label that classifies the event as fraudulent or legitimate. You can use any labels, such as "fraud", "legit", "1", or "0".
- LABEL_TIMESTAMP - The timestamp when the label was last updated.

> **Note**
> Event metadata requirements can change depending on the type of model you want to create. For example, if you're creating Online Fraud Insights model or evaluating events in real-time to make fraud predictions, then you don't need to provide ENTITY_ID, ENTITY_TYPE, and LABEL_TIMESTAMP in CSV file header and in the dataset. For more information, see Choose a model type and Get fraud predictions.

Event variable

Depending on your event and the model that you create, you might want to include data elements such as email_address, billing_address, ip_address, and phone_number, in your dataset. These data elements are called *event variables* and are provided as part of the file header in addition to the event metadata. The event variables must be in lowercase letters.

Because every business situation is different, we recommend that you add as many variables that are typically available and collected at the time of the event. When you start to create your model, you can choose to eliminate event variables that might not be helpful or if they have the potential to create a model that does not generalize well. For more information about event variables that can be included in your model for predicting fraud, see Create a variable.

Event data

The file header in your CSV file must be followed by event data with each row consisting of data elements collected from a single event. The event data values must correspond with the file header.

The following is an example of a CSV file containing data from an account registration event. The header row contains both event metadata in uppercase and event variables in lowercase followed by the event data. Each row in the dataset contains data elements associated with single account registration with each data element corresponding with the header.

Gather event data

Gathering your event data is an important step in creating your model. This is because the performance of your model in predicting fraud is dependent on the quality of your dataset. As you start to gather your event data, keep in mind the goals that you defined earlier for creating the model and how you intend to use it. This helps you decide what event variables to include. It's also important to decide the format of each event variable you intend to include and the total size of your dataset.

Event dataset quality

To gather high quality dataset for your model, we recommend the following:

- **Collect mature data-** Using the most recent data helps to identity the most recent fraud pattern. However, to detect fraud use cases, allow the data to mature. The maturity period is dependent on your business, and can take anywhere from two weeks to three months. For example, if your event includes credit card transaction, then the maturity of the data might be determined by the chargeback period of the credit card or time taken by an investigator to make determination.

  Ensure that the dataset used to train the model have had sufficient time to mature as per your business.
- **Make sure the data distribution doesn't drift significantly-** Amazon Fraud Detector model training process samples and partitions your dataset based on EVENT_TIMESTAMP. For example, if your dataset consists of fraud events pulled from last 6 months, but only the last month of legitimate events are included, the data distribution is considered to be drifting and unstable. An unstable dataset might lead to biases in model performance evaluation. If you find the data distribution to be drifting significantly, consider balancing your dataset by collecting data similar to the current data distribution.
- **Make sure the dataset is representative of the use case where the model is implemented/tested-** Otherwise, the estimated performance could be biased. Let us say that you are using a model to automatically decline all in-door applicants, but your model is trained with a dataset that has historical data/labels which were previously approved. Then, your model's evaluation might be inaccurate because the evaluation is based on the dataset that does not have representation from declined applicants.

Event data format

Amazon Fraud Detector transforms most of your data to the required format as part of its model training process. However, there are some standard formats you can easily use for providing your data that can help avoid issues later when Amazon Fraud Detector validates your dataset. The following table provides guidance on the formats for providing the recommended metadata.

Metadata name	Format	Required
EVENT_ID	If provided, it must meet the following requirements: • It is unique for that event. • It represents information that's meaningful to your business. • It follows the regular expression pattern (for example, `^[0-9a-z_-]+$.`) • In addition to the above requirements, we recommend that you do not append a timestamp to the EVENT_ID. Doing so might cause issues when you update the event. This because you must provide the exact same EVENT_ID if you do this.	Depends on the model type
EVENT_TIMESTAMP	• It must be specified in one of the following formats:	Yes

Metadata name	Format	Required
	• %yyyy-%mm-%ddT%hh:%mm:%ssZ (ISO 8601 standard in UTC only with no milliseconds) Example: 2019-11-30T13:01:01Z • %yyyy/%mm/%dd %hh:%mm:%ss (AM/PM) Examples: 2019/11/30 1:01:01 PM, or 2019/11/30 13:01:01 • %mm/%dd/%yyyy %hh:%mm:%ss Examples: 11/30/2019 1:01:01 PM, 11/30/2019 13:01:01 • %mm/%dd/%yy %hh:%mm:%ss Examples: 11/30/19 1:01:01 PM, 11/30/19 13:01:01 • Amazon Fraud Detector makes the following assumptions when parsing date/timestamp formats for event timestamps: • If you are using the ISO 8601 standard, it must be an exact match of the preceding specification • If you are using one of the other formats, there is additional flexibility: • For months and days, you can provide single or double digits. For example, 1/12/2019 is a valid date. • You do not need to include hh:mm:ss if you do not have them (that is, you can simply provide a date). You can also provide a subset of just the hour and minutes (for example, hh:mm). Just providing hour is not supported. Milliseconds are also not supported.	

Metadata name	Format	Required
	• If you provide AM/PM labels, a 12-hour clock is assumed. If there is no AM/PM information, a 24-hour clock is assumed. • You can use "/" or "-" as delimiters for the date elements. ":" is assumed for the timestamp elements.	
ENTITY_ID	• It must follow the regular expression pattern: `^[0-9A-Za-z_.@+-]+$`. • If the entity id isn't available at the time of evaluation, specify the entity id as *unknown*.	Depends on the model type
ENTITY_TYPE	You can use any string	Depends on the model type
EVENT_LABEL	You can use any labels, such as "fraud", "legit", "1", or "0".	Required if LABEL_TIMESTAMP is included
LABEL_TIMESTAMP	It must follow the timestamp format.	Required if EVENT_LABEL is included

For information about the supported formats for event variables, see Create a variable.

Null or missing values

The EVENT_TIMESTAMP and EVENT_LABEL variables must not contain any null or missing values. You can have null or missing values for other variables. However, we recommend that you only use a small number nulls for those variables. If Amazon Fraud Detector determines that there are too many null or missing values for an event variables, it will automatically omit variable from your model.

Minimum variables

When you create your model, the dataset must include at least two event variables in addition to the required event metadata. The two event variables must pass the validation check.

Event dataset size

Required

- Minimum of 10K total events data
- Dataset must include at least 400 events (rows) classified as fraudulent and 400 events (rows) classified as legitimate
- If your model type requires ENTITY_ID, your dataset must have more than 100 unique entities.

Recommended

- Minimum of three weeks of historic data, but at best six months of data.

Dataset validation

Before Amazon Fraud Detector starts to create your model, it checks if the variables included in the dataset for training the model meets the size, format, and other requirements. If the dataset doesn't pass the validation, model isn't created. You must first fix the variables that didn't pass the validation before you create the model. Amazon Fraud Detector provides you with a *Data profiler* which you can use to help you identify and fix issues with your dataset before you start to train your model

Data profiler

Amazon Fraud Detector provides an open-source tool for profiling and preparing your data for model training. This automated data profiler helps you avoid common data preparation errors and identify potential issues like mis-mapped variable types that would negatively impact model performance. The profiler generates an intuitive and comprehensive report of your dataset, including variable statistics, label distribution, categorical and numeric analysis, and variable and label correlations. It provides guidance on variable types as well as an option to transform the dataset into a format that Amazon Fraud Detector requires.

Using data profiler

The automated data profiler is built with an AWS CloudFormation stack, which you can easily launch with a few clicks. All codes are available on Github. For information on how to use data profiler, follow directions in our blog Train models faster with an automated data profiler for Amazon Fraud Detector

Common event dataset errors

The following are some of the common issues Amazon Fraud Detector comes across when validating an event dataset. After you run the data profiler, use this list to check your dataset for errors before creating your model.

- CSV file isn't in the UTF-8 format.
- The number of events in the dataset is less than 10K.
- The number of events identified as fraud or legitimate is less than 400.
- The number of unique entities associated to a fraud event is less than 100.
- More than 0.1% of values in EVENT_TIMESTAMP contains nulls or values other than the supported date/timestamp formats.
- More than 1% of the values in EVENT_LABEL contains nulls or values other than those defined in the event type.
- Less than two variables are available for model training.

Dataset storage

After you gathered your dataset, you store your dataset internally using Amazon Fraud Detector or externally with Amazon Simple Storage Service (Amazon S3). We recommend that you choose where to store your dataset based on the model you use for generating fraud predictions. For more information, see Choose a model type. The following is a detailed breakdown of these two storage options.

- **Internal storage-** Your dataset is stored with Amazon Fraud Detector. All event data associated with an event is stored together. You can upload the event dataset that's stored with Amazon Fraud Detector at any time. You can either stream events one at a time to an Amazon Fraud Detector API, or import large datasets (up to 1GB) using the batch import feature. When you train a model using the dataset stored with Amazon Fraud Detector, you can specify a time range to limit the size of your dataset.
- **External storage-** Your dataset is stored in an external data source other than Amazon Fraud Detector. Currently, Amazon Fraud Detector supports using Amazon Simple Storage Service(Amazon S3) for this

purpose. If your model is on a file that's uploaded to Amazon S3, that file can't be more than 5GB of uncompressed data. If it's more than that, make sure to shorten the time range of your dataset.

Create event type

With Amazon Fraud Detector you generate fraud predictions for events. An event type defines the structure for an individual event sent to Amazon Fraud Detector. Once defined, you can build models and detectors that evaluate the risk for specific event types.

The structure of an event includes the following:

- Entity Type: Classifies who is performing the event. During prediction, specify the entity type and entity Id to define who performed the event.
- Variables: Defines what variables can be sent as part of the event. Variables are used by models and rules to evaluate fraud risk. Once added, variables cannot be removed from an event type.
- Labels: Classifies an event as fraudulent or legitimate. Used during model training. Once added, labels cannot be removed form an event type.

For details on how to create an event type using the AWS Console see Step 1: Create event type (p. 11).

Create an event type using the AWS SDK for Python (Boto3)

The following example shows a sample request for the `PutEventType` API. The example assumes you have created the variables `ip_address` and `email_address`, the labels `legit` and `fraud`, and the entity type `sample_customer`. For information about how to create these resources, see Create resources (p. 61).

Note
You must first create variables, entity types, and labels prior to adding them to the event type.

```
import boto3
fraudDetector = boto3.client('frauddetector')

fraudDetector.put_event_type (
name = 'sample_registration',
eventVariables = ['ip_address', 'email_address'],
labels = ['legit', 'fraud'],
entityTypes = ['sample_customer'])
```

Create a model

Amazon Fraud Detector uses machine learning models for generating fraud predictions. Each model is trained using a *model type*. The model type specifies the algorithms and transformations used for training the model. Model training is the process of using a dataset that you provide to create a model that can predict fraudulent events.

To create a model, you must first choose a model type and then prepare and provide data that will be used to train the model.

Choose a model type

The following model types are available in Amazon Fraud Detector. Choose a model type that works for your use case.

- **Online Fraud Insights**

  The *Online Fraud Insights* model type is optimized to detect fraud when little historical data is available about the entity being evaluated, for example, a new customer registering online for a new account.
- **Transaction Fraud Insights**

  The *Transaction Fraud Insights* model type is best suited for detecting fraud use cases where the entity that is being evaluated might have a history of interactions that the model can analyze to improve prediction accuracy (for example, an existing customer with history of past purchases).

Online fraud insights

Online Fraud Insights is a supervised machine learning model, which means that it uses historical examples of fraudulent and legitimate transactions to train the model. The Online Fraud Insights model can detect fraud based on little historical data. The model's inputs are flexible, so you can adapt it to detect a variety of fraud risks including fake reviews, promotion abuse, and guest checkout fraud.

The Online Fraud Insights model uses an ensemble of machine learning algorithms for data enrichment, transformation, and fraud classification. As part of the model training process, Online Fraud Insights enriches raw data elements like IP address and BIN number with third-party data such as the geolocation of the IP address or the issuing bank for a credit card. In addition to third-party data, Online Fraud Insights uses deep learning algorithms that take into account fraud patterns that have been seen at Amazon and AWS. These fraud patterns become input features to your model using a gradient tree boosting algorithm.

To increase performance, Online Fraud Insights optimizes the hyper parameters of the gradient tree boosting algorithm via a Bayesian optimization process. It sequentially trains dozens of different models with varying model parameters (such as number of trees, depth of trees, and number of samples per leaf). It also uses different optimization strategies like upweighting the minority fraud population to take care of very low fraud rates.

Selecting data source

When training an Online Fraud Insights model, you can choose to train the model on event data that is either stored externally (outside of Amazon Fraud Detector) or stored within Amazon Fraud Detector. The external storage Amazon Fraud Detector currently supports is Amazon Simple Storage Service (Amazon

S3). If your are using external storage, your event dataset must be uploaded as a comma-separated values (CSV) format to an Amazon S3 bucket. These data storage options are referred to within the model training configuration as EXTERNAL_EVENTS (for external storage) and INGESTED_EVENTS (for internal storage). For more information about the available data sources and how to store data in them, see Choose storage location for your event dataset (p. 36)

Preparing data

Regardless of where you choose to store your event data (Amazon S3 or Amazon Fraud Detector), the requirements for Online Fraud Insights model type are the same.

Your dataset must contain the column header EVENT_LABEL. This variable classifies an event as fraudulent or legitimate. When using a CSV file (external storage), you must include EVENT_LABEL for each event in the file. For internal storage, the EVENT_LABEL field is optional but all events must be labeled to be included within a training dataset. When configuring your model training, you can choose whether to ignore unlabeled events, assume a legitimate label for unlabeled events, or assume a fraudulent label for all unlabeled events.

Selecting data

See Gather event data for information on selecting data for training your Online Fraud Insights model.

The Online Fraud Insights training process samples and partitions historic data based on EVENT_TIMESTAMP. There is no need to manually sample the data and doing so may negatively impact your model results.

Event variables

The Online Fraud Insights model requires at least two variables, apart from the required event metadata, that has passed data validation for model training and allows up to 100 variables per model. Generally, the more variables you provide, the better the model can differentiate between fraud and legitimate events. While the Online Fraud Insights model can support dozens of variables, including custom variables, we recommend including IP address and email address because these variables are typically most effective at identifying the entity being evaluated.

Validating data

As part of the training process, Online Fraud Insights will validate the dataset for data quality issues that may impact model training. After validating the data, Amazon Fraud Detector will take appropriate action to build the best possible model. This includes issuing warnings for potential data quality issues, automatically removing variables that have data quality issues, or issuing an error and stopping the model training process. For more information, see dataset validation.

Transaction fraud insights

The Transaction Fraud Insights model type is designed to detect online, or card-not-present, transaction fraud. Transaction Fraud Insights is a supervised machine learning model, which means that it uses historical examples of fraudulent and legitimate transactions to train the model.

The Transaction Fraud Insights model uses an ensemble of machine learning algorithms for data enrichment, transformation, and fraud classification. It leverages a feature engineering engine to create entity-level and event-level aggregates. As part of the model training process, Transaction Fraud Insights enriches raw data elements like IP address and BIN number with third-party data such as the geolocation of the IP address or the issuing bank for a credit card. In addition to third-party data, Transaction Fraud Insights uses deep learning algorithms that take into account fraud patterns that have been seen at Amazon and AWS These fraud patterns become input features to your model using a gradient tree boosting algorithm.

To increase performance, Transaction Fraud Insights optimizes the hyper parameters of the gradient tree boosting algorithm via a Bayesian optimization process, sequentially training dozens of different models with varying model parameters (such as number of trees, depth of trees, number of samples per leaf) as well as different optimization strategies like upweighting the minority fraud population to take care of very low fraud rates.

As part of the model training process, the Transaction Fraud model's feature engineering engine calculates values for each unique entity within your training dataset to help improve fraud predictions. For example, during the training process, Amazon Fraud Detector computes and stores the last time an entity made a purchase and dynamically updates this value each time you call the `GetEventPrediction` or `SendEvent` API. During a fraud prediction, the event variables are combined with other entity and event metadata to predict whether the transaction is fraudulent.

Selecting data source

Transaction Fraud Insights models are trained on dataset stored internally with Amazon Fraud Detector (INGESTED_EVENTS) only. This allows Amazon Fraud Detector to continuously update calculated values about the entities you are evaluating. For more information about the available data sources, see Choose storage location for your event dataset (p. 36)

Preparing data

Before you train a Transaction Fraud Insights model, ensure that your data file contains all headers as mentioned in Prepare event dataset. The Transaction Fraud Insights model compares new entities that are received with the examples of fraudulent and legitimate entities in the dataset, so it is helpful to provide many examples for each entity.

Amazon Fraud Detector automatically transforms the stored event dataset into the correct format for training. After the model has completed training, you can review the performance metrics and determine whether you should add entities to your training dataset.

Selecting data

By default, Transaction Fraud Insights trains on your entire stored dataset for the Event Type that you select. You can optionally set a time range to reduce the events that are used to train your model. When setting a time range, ensure that the records that are used to train the model have had sufficient time to mature. That is, enough time has passed to ensure legitimate and fraud records have been correctly identified. For example, for chargeback fraud, it often takes 60 days or more to correctly identify fraudulent events. For the best model performance, ensure that all records in your training dataset are mature.

There is no need to select a time range that represents an ideal fraud rate. Amazon Fraud Detector automatically samples your data to achieve balance between fraud rates, time range, and entity counts.

Amazon Fraud Detector returns a validation error during model training if you select a time range for which there are not enough events to successfully train a model. For stored datasets, the EVENT_LABEL field is optional, but events must be labeled to be included in your training dataset. When configuring your model training, you can choose whether to ignore unlabeled events, assume a legitimate label for unlabeled events, or assume a fraudulent label for unlabeled events.

Event variables

The event type used to train the model must contain at least 2 variables, apart from required event metadata, that has passed data validation and can contain up to 100 variables. Generally, the more variables you provide, the better the model can differentiate between fraud and legitimate events. Although the Transaction Fraud Insight model can support dozens of variables, including custom variables, we recommend that you include IP address, email address, payment instrument type, order price, and card BIN.

Validating data

As part of the training process, Transaction Fraud Insights validates the training dataset for data quality issues that might impact model training. After validating the data, Amazon Fraud Detector takes appropriate action to build the best possible model. This includes issuing warnings for potential data quality issues, automatically removing variables that have data quality issues, or issuing an error and stopping the model training process. For more information, see Dataset validation.

Amazon Fraud Detector will issue a warning but continue training a model if the number of unique entities is less than 1,500 because this can impact the quality of the training data. If you receive a warning, review the performance metric (p. 49).

Choose storage location for your event dataset

As part of the model version configuration, Amazon Fraud Detector requires you to specify where your events data is stored. Amazon Fraud Detector supports two types of storage for your dataset. External storage and Internal storage.

- External storage - Event data is stored in a data source other than Amazon Fraud Detector. Currently Amazon Fraud Detector supports event data stored in Amazon Simple Storage Service(Amazon S3).

  This option is referred to within the model training configuration as `EXTERNAL_EVENTS`.
- Internal storage - Event data is stored in Amazon Fraud Detector. Events are stored at the Event Type resource level, so all events of the same Event Type are stored together in a single Event Type dataset. You can store or update events in Amazon Fraud Detector using the `GetEventPrediction` or `SendEvent` API operation. Or, you can import large datasets (upto 1GB) using batch import feature.When training a model using internal storage, you can optionally specify a time range of events to limit the size of your training dataset.

  This option is referred to within the model training configuration as `INGESTED_EVENTS` and as **Stored events** in the Amazon Fraud Detector console.

Model type	Compatible training data source
Online Fraud Insights	External storage, Internal storage
Transaction Fraud Insights	Internal storage

Stored events

You can choose to store event data in Amazon Fraud Detector and use the stored data later to train your models. By storing event data in Amazon Fraud Detector, you can train models that use auto-computed variables to improve performance, simplify model retraining, and update fraud labels to close the machine learning feedback loop. Events are stored at the Event Type resource level, so all events of the same event type are stored together in a single event type dataset. As part of defining an event type, you can optionally specify whether to store events for that event type by toggling the *Event Ingestion* setting in the Amazon Fraud Detector console.

You can either store single events or import large number of event datasets in Amazon Fraud Detector. Single events can be streamed using the `GetEventPrediction` API or the `SendEvent` API. Large datasets can be quickly and easily imported to Amazon Fraud Detector using the batch import feature.

You can use the Amazon Fraud Detector console at any time to check the number of events already stored for each event type.

Preparing event data for storage

You can store your historical events data in Amazon Fraud Detector. Event data is stored at the Event Type resource level, so all events of the same Event Type are stored in a single Event Type. The stored events can later be used to train a new model or re-train an existing model. When training a model using the stored event data, you can optionally specify a time range of events to limit the size of your training dataset.

Amazon Fraud Detector provides sample dataset you can optionally use for training your Transaction fraud insights model. If you want to use the sample dataset, go to Amazon Fraud Detector samples, download `transaction_data_100k_full` and then follow instructions to store the sample dataset either using the SendEvents API operation or using batch import.

Create a CSV file

Amazon Fraud Detector imports data only from files that are in the comma-separated values (CSV) format. The first row of your CSV file must contain column headers that exactly match the variables defined in the associated event type plus four mandatory variables: EVENT_ID, EVENT_TIMESTAMP, ENTITY_ID, and ENTITY_TYPE. You can also optionally include EVENT_LABEL and LABEL_TIMESTAMP (LABEL_TIMESTAMP is required if EVENT_LABEL is included).

Define mandatory variables

Mandatory variables are considered as event metadata and they must be specified in uppercase. Event metadata are automatically included for model training. The following table lists the mandatory variables, description of each variable, and required format for the variable.

Name	Description	Requirements
EVENT_ID	An identifier for the event. For example, if your event is an online transaction, the EVENT_ID might be the transaction reference number that was provided to your customer.	• The EVENT_ID is required for batch import jobs. • It must be unique for that event. • It should represent information that's meaningful to your business. • It must satisfy the regular expression pattern (for example, `^[0-9a-z_-]+$.`) • We don't recommend that you append a timestamp to the EVENT_ID. Doing so might cause issues when you update the event. This because you must provide the exact same EVENT_ID if you do this.
EVENT_TIMESTAMP	The timestamp of when the event occurred. The timestamp must be in ISO 8601 standard in UTC.	• The EVENT_TIMESTAMP is required for batch import jobs. • It must be specified in one of the following formats: • %yyyy-%mm-%ddT%hh:%mm:%ssZ (ISO 8601 standard in UTC only with no milliseconds)

Name	Description	Requirements
		Example: 2019-11-30T13:01:01Z • %yyyy/%mm/%dd %hh:%mm:%ss (AM/PM) Examples: 2019/11/30 1:01:01 PM, or 2019/11/30 13:01:01 • %mm/%dd/%yyyy %hh:%mm:%ss Examples: 11/30/2019 1:01:01 PM, 11/30/2019 13:01:01 • %mm/%dd/%yy %hh:%mm:%ss Examples: 11/30/19 1:01:01 PM, 11/30/19 13:01:01 • Amazon Fraud Detector makes the following assumptions when parsing date/timestamp formats for event timestamps: • If you are using the ISO 8601 standard, it must be an exact match of the preceding specification • If you are using one of the other formats, there is additional flexibility: • For months and days, you can provide single or double digits. For example, 1/12/2019 is a valid date. • You do not need to include hh:mm:ss if you do not have them (that is, you can simply provide a date). You can also provide a subset of just the hour and minutes (for example, hh:mm). Just providing hour is not supported. Milliseconds are also not supported. • If you provide AM/PM labels, a 12-hour clock is assumed. If there is no AM/PM information, a 24-hour clock is assumed.

Name	Description	Requirements
		• You can use "/" or "-" as delimiters for the date elements. ":" is assumed for the timestamp elements.
ENTITY_ID	An identifier for the entity performing the event.	• ENTITY_ID is required for batch import jobs • It must follow the regular expression pattern: `^[0-9A-Za-z_.@+-]+$`. • If the entity id isn't available at the time of evaluation, specify the entity id as *unknown*.
ENTITY_TYPE	The entity that performs the event, such as a merchant or a customer	ENTITY_TYPE is required for batch import jobs
EVENT_LABEL	Classifies the event as `fraudulent` or `legitimate`	EVENT_LABEL is required if LABEL_TIMESTAMP is included
LABEL_TIMESTAMP	The timestamp when the event label was last populated or updated	• LABEL_TIMESTAMP is required if EVENT_LABEL is included. • It must follow the timestamp format.

Event Timestamp formats for SendEvent API

If you are storing event data using `SendEvent` API, you must ensure that your event timestamp is in the required format. Amazon Fraud Detector supports the following date/timestamp formats:

- %yyyy-%mm-%ddT%hh:%mm:%ssZ (ISO 8601 standard in UTC only with no milliseconds)

  Example: 2019-11-30T13:01:01Z
- %yyyy/%mm/%dd %hh:%mm:%ss (AM/PM)

  Examples: 2019/11/30 1:01:01 PM, or 2019/11/30 13:01:01
- %mm/%dd/%yyyy %hh:%mm:%ss

  Examples: 11/30/2019 1:01:01 PM, 11/30/2019 13:01:01
- %mm/%dd/%yy %hh:%mm:%ss

  Examples: 11/30/19 1:01:01 PM, 11/30/19 13:01:01

Amazon Fraud Detector makes the following assumptions when parsing date/timestamp formats for event timestamps:

- If you are using the ISO 8601 standard, it must be an exact match of the preceding specification
- If you are using one of the other formats, there is additional flexibility:
  - For months and days, you can provide single or double digits. For example, 1/12/2019 is a valid date.

- You do not need to include hh:mm:ss if you do not have them (that is, you can simply provide a date). You can also provide a subset of just the hour and minutes (for example, hh:mm). Just providing hour is not supported. Milliseconds are also not supported.
- If you provide AM/PM labels, a 12-hour clock is assumed. If there is no AM/PM information, a 24-hour clock is assumed.
- You can use "/" or "-" as delimiters for the date elements. ":" is assumed for the timestamp elements.

Validating stored data

When uploading events via the `SendEvent` or `GetEventPrediction` API operation, Amazon Fraud Detector validates the following:

- The EventIngestion setting for that event type is ENABLED.
- Event timestamps cannot be updated. An event with a repeated event ID and different EVENT_TIMESTAMP will be treated as an error.
- Variable names and values match their expected format. For more information, see Create a variable (p. 61)
- Required variables are populated with a value.
- All event timestamps are not older than 18 months and are not in the future.

Updating event labels

You might need to add or update fraud labels for events that are already stored in Amazon Fraud Detector, such as when you perform an offline fraud investigation for an event and want to close the machine learning feed back loop. To update the label for an event that is already stored in Amazon Fraud Detector, use the `UpdateEventLabel` API operation. The following shows an example UpdateEventLabel API call.

```
import boto3
fraudDetector = boto3.client('frauddetector')

fraudDetector.update_event_label(
            eventId        = '802454d3-f7d8-482d-97e8-c4b6db9a0428',
            eventTypeName  = 'sample_registration',
            assignedLabel  = 'fraud',
            labelTimestamp = '2020-07-13T23:18:21Z'
)
```

Storing event data using the GetEventPredictions API operation

By default, all events sent to the `GetEventPrediction` API for evaluation are stored in Amazon Fraud Detector. This means that Amazon Fraud Detector will automatically store event data when you generate a prediction and use that data to update calculated variables in near-real time. You can disable data storage by navigating to the event type in the Amazon Fraud Detector console and setting **Event ingestion** OFF or updating the EventIngestion value to DISABLED using the `PutEventType` API operation. For more information about the `GetEventPrediction` API operation, see Get fraud predictions (p. 86).

> **Important**
> We highly recommend that once you enable *Event ingestion* for an Event type, keep it enabled. Disabling the Event ingestion for the same Event type and then generating predictions might result in inconsistent behavior.

Storing event data using the SendEvent API operation

You can use the `SendEvent` API operation to store events in Amazon Fraud Detector without generating fraud predictions for those events. For example, you can use the `SendEvent` operation to upload a historical dataset, which you can later use to train a model.

The following is an example `SendEvent` API call.

```
import boto3
fraudDetector = boto3.client('frauddetector')

fraudDetector.send_event(
            eventId        = '802454d3-f7d8-482d-97e8-c4b6db9a0428',
            eventTypeName  = 'sample_registration',
            eventTimestamp = '2020-07-13T23:18:21Z',
            eventVariables =  {
       'email_address' : 'johndoe@exampledomain.com',
       'ip_address' : '1.2.3.4'},
            assignedLabel  = 'legit',
            labelTimestamp = '2020-07-13T23:18:21Z',
            entities       = [{'entityType':'sample_customer', 'entityId':'12345'}],

)
```

Storing event data using batch import

With the batch import feature, you can quickly and easily upload large historical event datasets in Amazon Fraud Detector using the console, the API, or the AWS SDK. To use batch import, create an input file in CSV format that contains all your event data, upload the CSV file onto Amazon S3 bucket, and start an *Import* job. Amazon Fraud Detector first validates the data based on the event type, and then automatically imports the entire dataset. After the data is imported, it's training ready to be used for training new models or for re-training existing models.

Input and output files

The input CSV file must contain headers that match the variables defined in the associated event type plus four mandatory variables. See Preparing event data for storage (p. 37) for more information. The maximum size of the input data file is 1 Gigabytes (GB), or about 3 million events. The number of events will vary by your event size. If the import job was successful, the output file is empty. If the import was unsuccessful, the output file contains the error logs.

Upload CSV file to Amazon S3 for batch import

After you create a CSV file with your data, upload the file to your Amazon Simple Storage Service (Amazon S3) bucket.

To upload event data to an Amazon S3 bucket

1. Sign in to the AWS Management Console and open the Amazon S3 console at https://console.aws.amazon.com/s3/.
2. Choose **Create bucket**.

   The **Create bucket** wizard opens.
3. In **Bucket name**, enter a DNS-compliant name for your bucket.

   The bucket name must:

- Be unique across all of Amazon S3.
- Be between 3 and 63 characters long.
- Not contain uppercase characters.
- Start with a lowercase letter or number.

After you create the bucket, you can't change its name. For information about naming buckets, see Bucket naming rules in the *Amazon Simple Storage Service User Guide*.

Important
Avoid including sensitive information, such as account numbers, in the bucket name. The bucket name is visible in the URLs that point to the objects in the bucket.

4. In **Region**, choose the AWS Region where you want the bucket to reside. You must select the same Region in which you are using Amazon Fraud Detector, that is US East (N. Virginia), US East (Ohio), US West (Oregon), Europe (Ireland), Asia Pacific (Singapore) or Asia Pacific (Sydney).
5. In **Bucket settings for Block Public Access**, choose the Block Public Access settings that you want to apply to the bucket.

   We recommend that you leave all settings enabled. For more information about blocking public access, see Blocking public access to your Amazon S3 storage in the *Amazon Simple Storage Service User Guide*.
6. Choose **Create bucket**.
7. Upload training data file to your Amazon S3 bucket. Note the Amazon S3 location path for your training file (for example, s3://bucketname/object.csv).

Batch import event data

You can easily import large number of your event datasets in Amazon Fraud Detector console, using the `CreateBatchImportJob` API or using AWS SDK. Before you proceed, make sure that you have followed instructions to prepare your dataset as a CSV file. Make sure that you also uploaded the CSV file to an Amazon S3 bucket.

Using Amazon Fraud Detector console

To batch import event data in console

1. Open the AWS Console and sign in to your account, and navigate to Amazon Fraud Detector.
2. In the left navigation pane, choose **Events**.
3. Choose your event type.
4. Select **Stored events** tab.
5. In the **Stored events details** pane, make sure that the **Event ingestion** is **ON**.
6. In the **Import events data** pane, choose **New Import**.
7. In the **New events import** page, provide the following information:
   - For **IAM role for data**, select the IAM role that you created for the Amazon S3 bucket that holds the CSV file you are planning to import.
   - For **Input data location**, enter the S3 location where you have your CSV file.
   - If you want to specify a separate location to store your import results, click **Separate data location for inputs and results** button and provide a valid Amazon S3 bucket location.
8. Choose **Start**.
9. The **Status** column in **Import events data** pane displays the status of your import job.
10. Choose **Job Id** of an import job to view details.

Note
We recommend waiting 10 minutes after you've finished importing events data into AFD to ensure that they are fully ingested by the system.

Batch import event data using the AWS SDK for Python (Boto3)

The following example shows a sample request for `CreateBatchImportJob` API. A batch import job must include a **jobID**, **inputPath**, **outputPath**, **eventTypeName** and **iamRoleArn**. The jobID can't contain the same ID of a past job, unless the job exists in CREATE_FAILED state. The inputPath and outputPath must be valid S3 paths. You can opt out of specifying the file name in the outputPath, however, you will still need to provide a valid S3 bucket location. The eventTypeName and iamRoleArn must exist.

```
import boto3
fraudDetector = boto3.client('frauddetector')

fraudDetector.create_batch_import_job (
jobId = 'sample_batch_import',
inputPath = 's3://bucket_name/input_file_name.csv',
outputPath = 's3://bucket_name/',
eventTypeName = 'sample_registration',
"iamRoleArn": 'arn:aws:iam::************:role/service-role/AmazonFraudDetector-
DataAccessRole-*************'
)
```

Cancelling batch import job

You can cancel an in-progress batch import job at any time in the Amazon Fraud Detector console, using the `CancelBatchImportJob` API, or AWS SDK.

To cancel a batch import job in console,

1. Open the AWS Console and sign in to your account, and navigate to Amazon Fraud Detector.
2. In the left navigation pane, choose **Events**.
3. Choose your event type.
4. Select **Stored events** tab.
5. In the **Import events data** pane, choose the job Id of an in-progress import job you want to cancel.
6. In the event job page, click **Actions** and select **Cancel events import**.
7. Choose **Stop events import** to cancel the batch import job.

Canceling batch import job using the AWS SDK for Python (Boto3)

The following example shows a sample request for the `CancelBatchImportJob` API. The cancel import job must include the job ID of an in-progress batch import job.

```
import boto3
fraudDetector = boto3.client('frauddetector')
fraudDetector.cancel_batch_import_job (
    jobId = 'sample_batch'
)
```

Getting details of a stored event data

After you store event data in Amazon Fraud Detector, you can check the latest values that are stored to an event using the `GetEvent` operation. See the following example.

```
import boto3
fraudDetector = boto3.client('frauddetector')

fraudDetector.update_event_label(
            eventId        = '802454d3-f7d8-482d-97e8-c4b6db9a0428',
            eventTypeName  = 'sample_registration'
)
```

Viewing metrics of stored event dataset

For each event type, you can view metrics such as, number of stored events, total size of your stored events, and timestamps of the earliest and the latest stored events, in the Amazon Fraud Detector console.

To view stored event metrics of an event type,

1. Open the AWS Console and sign in to your account. Navigate to Amazon Fraud Detector.
2. In the left navigation pane, choose **Events**.
3. Choose your event type.
4. Select **Stored events** tab.
5. The **Stored events details** pane displays the metrics. These metrics are automatically updated once per day.
6. Optionally click **Refresh event metrics** to manually update your metrics.

   Note
   If you have just imported your data, we recommend waiting 5 - 10 minutes after you have finished importing data to refresh and view metrics.

Store your event data with Amazon S3

If you are training an Online Fraud Insights model, you can choose to store your event data externally with Amazon S3. To store your event data in Amazon S3 you must first create a text file in CSV format, add your event data, and then upload the CSV file to an Amazon S3 bucket.

Note
Transaction Fraud Insights model does not support datasets stored externally with Amazon S3

Creating CSV file

Amazon Fraud Detector requires that the first row of your CSV file contain column headers. The column headers in your CSV file must map to the variables that are defined in the event type. For an example dataset, see Get and upload example dataset (p. 10)

The Online Fraud Insights model requires a training dataset that has at least 2 variables and up to 100 variables. In addition to the event variables, the training dataset must contain the following headers:

- EVENT_TIMESTAMP - Defines when the event occurred
- EVENT_LABEL - Classifies the event as fraudulent or legitimate. The values in the column must correspond to the values defined in the event type.

The following sample CSV data represents historical registration events from an online merchant:

```
EVENT_TIMESTAMP,EVENT_LABEL,ip_address,email_address
4/10/2019 11:05,fraud,209.146.137.48,fake_burtonlinda@example.net
```

```
12/20/2018 20:04,legit,203.0.112.189,fake_davidbutler@example.org
3/14/2019 10:56,legit,169.255.33.54,fake_shelby76@example.net
1/3/2019 8:38,legit,192.119.44.26,fake_curtis40@example.com
9/25/2019 3:12,legit,192.169.85.29,fake_rmiranda@example.org
```

Note
The CSV data file can contain double quotes and commas as part of your data.

A simplified version of the corresponding event type is represented below. The event variables correspond to the headers in the CSV file and the values in `EVENT_LABEL` correspond to the values in the labels list.

```
(
name = 'sample_registration',
eventVariables = ['ip_address', 'email_address'],
labels = ['legit', 'fraud'],
entityTypes = ['sample_customer']
)
```

Event Timestamp formats

Ensure that your event timestamp is in the required format. As part of the model build process, the Online Fraud Insights model type orders your data based on the event timestamp, and splits your data for training and testing purposes. To get a fair estimate of performance, the model first trains on the training dataset, and then tests this model on the test dataset.

Amazon Fraud Detector supports the following date/timestamp formats for the values in `EVENT_TIMESTAMP` during model training:

- %yyyy-%mm-%ddT%hh:%mm:%ssZ (ISO 8601 standard in UTC only with no milliseconds)

  Example: 2019-11-30T13:01:01Z
- %yyyy/%mm/%dd %hh:%mm:%ss (AM/PM)

  Examples: 2019/11/30 1:01:01 PM, or 2019/11/30 13:01:01
- %mm/%dd/%yyyy %hh:%mm:%ss

  Examples: 11/30/2019 1:01:01 PM, 11/30/2019 13:01:01
- %mm/%dd/%yy %hh:%mm:%ss

  Examples: 11/30/19 1:01:01 PM, 11/30/19 13:01:01

Amazon Fraud Detector makes the following assumptions when parsing date/timestamp formats for event timestamps:

- If you are using the ISO 8601 standard, it must be an exact match of the preceding specification
- If you are using one of the other formats, there is additional flexibility:
  - For months and days, you can provide single or double digits. For example, 1/12/2019 is a valid date.
  - You do not need to include hh:mm:ss if you do not have them (taht is, you can simply provide a date). You can also provide a subset of just the hour and minutes (for example, hh:mm). Just providing hour is not supported. Milliseconds are also not supported.
  - If you provide AM/PM labels, a 12-hour clock is assumed. If there is no AM/PM information, a 24-hour clock is assumed.
  - You can use “/” or “-” as delimiters for the date elements. “:” is assumed for the timestamp elements.

Sampling your dataset across time

We recommend that you provide examples of fraud and legitimate samples from the same time range. For example, if you provide fraud events from the past 6 months, you should also provide legitimate events that evenly span the same time period. If your dataset contains a highly uneven distribution of fraud and legitimate events, you might receive the following error: *"The fraud distribution across time is unacceptably fluctuant. Cannot split dataset properly."* Typically, the easiest fix for this error is to ensure that the fraud events and legitimate events are sampled evenly across the same timeframe. You also might need to remove data if you experienced a large spike in fraud within a short time period.

If you cannot generate enough data to create an evenly distributed dataset, one approach is to randomize the EVENT_TIMESTAMP of your events such that they are evenly distributed. However, this often results in performance metrics being unrealistic because Amazon Fraud Detector uses EVENT_TIMESTAMP to evaluate models on the appropriate subset of events in your dataset.

Null and missing values

Amazon Fraud Detector handles null and missing values. However, the percentage of nulls for variables should be limited. EVENT_TIMESTAMP and EVENT_LABEL columns should not contain any missing values.

File validation

Amazon Fraud Detector will fail to train a model if any of the following conditions are triggered:

- If the CSV is unable to be parsed
- If the datatype for a column is incorrect

Uploading your event data to an Amazon S3 bucket

After you create a CSV file with your event data, upload the file to your Amazon S3 bucket.

To upload to an Amazon S3 bucket

1. Sign in to the AWS Management Console and open the Amazon S3 console at https://console.aws.amazon.com/s3/.
2. Choose **Create bucket**.

   The **Create bucket** wizard opens.
3. In **Bucket name**, enter a DNS-compliant name for your bucket.

   The bucket name must:

   - Be unique across all of Amazon S3.
   - Be between 3 and 63 characters long.
   - Not contain uppercase characters.
   - Start with a lowercase letter or number.

   After you create the bucket, you can't change its name. For information about naming buckets, see Bucket naming rules in the *Amazon Simple Storage Service User Guide*.

   Important
   Avoid including sensitive information, such as account numbers, in the bucket name. The bucket name is visible in the URLs that point to the objects in the bucket.

4. In **Region**, choose the AWS Region where you want the bucket to reside. You must select the same Region in which you are using Amazon Fraud Detector, that is US East (N. Virginia), US East (Ohio), US West (Oregon), Europe (Ireland), Asia Pacific (Singapore) or Asia Pacific (Sydney).
5. In **Bucket settings for Block Public Access**, choose the Block Public Access settings that you want to apply to the bucket.

   We recommend that you leave all settings enabled. For more information about blocking public access, see Blocking public access to your Amazon S3 storage in the *Amazon Simple Storage Service User Guide*.
6. Choose **Create bucket**.
7. Upload training data file to your Amazon S3 bucket. Note the Amazon S3 location path for your training file (for example, s3://bucketname/object.csv).

Build a model

Amazon Fraud Detector models learn to detect fraud for a specific event type. In Amazon Fraud Detector, you first create a model, which acts as a container for your model versions. Each time you train a model, a new version is created. For details on how to create and train a model using the AWS Console see Step 2: Create model (p. 12).

Each model has a corresponding model score variable. Amazon Fraud Detector creates this variable on your behalf when you create a model. You can use this variable in your rule expressions to interpret your model scores during a fraud evaluation.

Train and deploy a model using the AWS SDK for Python (Boto3)

A model version is created by calling the `CreateModel` and `CreateModelVersion` operations. `CreateModel` initiates the model, which acts as a container for your model versions. `CreateModelVersion` starts the training process, which results in a specific version of the model. A new version of the solution is created each time you call `CreateModelVersion`.

The following example shows a sample request for the `CreateModel` API. This example creates *Online Fraud Insights* model type and assumes you have created an event type `sample_registration`. For additional details about creating an event type, see Create event type (p. 32).

```
import boto3
fraudDetector = boto3.client('frauddetector')

fraudDetector.create_model (
modelId = 'sample_fraud_detection_model',
eventTypeName = 'sample_registration',
modelType = 'ONLINE_FRAUD_INSIGHTS')
```

Train your first version using the `CreateModelVersion` API. `TrainingDataSource` and `ExternalEventsDetail` specify the source and Amazon S3 location of the training data set. `TrainingDataSchema` specifies how Amazon Fraud Detector should interpret the training data, specifically which event variables to include and how to classify the event labels.

```
import boto3
fraudDetector = boto3.client('frauddetector')

fraudDetector.create_model_version (
modelId = 'sample_fraud_detection_model',
```

```
modelType = 'ONLINE_FRAUD_INSIGHTS',
trainingDataSource = 'EXTERNAL_EVENTS',
trainingDataSchema = {
    'modelVariables' : ['ip_address', 'email_address'],
    'labelSchema' : {
        'labelMapper' : {
            'FRAUD' : ['fraud'],
            'LEGIT' : ['legit']
        }
    }
},
externalEventsDetail = {
    'dataLocation' : 's3://bucket/file.csv',
    'dataAccessRoleArn' : 'role_arn'
}
)
```

A successful request will result in a new model version with status `TRAINING_IN_PROGRESS`. At any point during the training, you can cancel the training by calling `UpdateModelVersionStatus` and updating the status to `TRAINING_CANCELLED`. Once training is complete, the model version status will update to `TRAINING_COMPLETE`. You can review model performance using the Amazon Fraud Detector console or by calling `DescribeModelVersions`. For more information on how to interpret model scores and performance, see Model scores (p. 48) and Model performance metrics (p. 49).

After reviewing the model performance, activate the model to make it available to use by Detectors in real-time fraud predictions. Amazon Fraud Detector will deploy the model in multiple availability zones for redundancy with auto-scaling turned on to ensure the model scales with the number of fraud predictions you are making. To activate the model, call the `UpdateModelVersionStatus` API and update the status to `ACTIVE`.

```
import boto3
fraudDetector = boto3.client('frauddetector')

fraudDetector.update_model_version_status (
modelId = 'sample_fraud_detection_model',
modelType = 'ONLINE_FRAUD_INSIGHTS',
modelVersionNumber = '1.00',
status = 'ACTIVE'
)
```

Model scores

Amazon Fraud Detector generates model scores between 0 and 1000, where 0 is low fraud risk and 1000 is high fraud risk. Model scores are directly related to the false positive rate (FPR). For example, a score of 600 corresponds to an estimated 10% false positive rate whereas a score of 900 corresponds to an estimated 2% false positive rate. The following table provides details of how certain model scores correlate to estimated false positive rates.

Model score	Estimated FPR
975	0.50%
950	1%
900	2%
860	3%

Model score	Estimated FPR
775	5%
700	7%
600	10%

Model performance metrics

After model training is complete, Amazon Fraud Detector validates model performance using 15% of your data that was not used to train the model. You can expect your trained Amazon Fraud Detector model to have real-world fraud detection performance that is similar to the validation performance metrics.

As a business, you must balance between detecting more fraud, and adding more friction to legitimate customers. To assist in choosing the right balance, Amazon Fraud Detector provides the following tools to assess model performance:

- **Score distribution chart** – A histogram of model score distributions assumes an example population of 100,000 events. The left Y axis represents the legitimate events and the right Y axis represents the fraud events. You can select a specific model threshold by clicking on the chart area. This will update the corresponding views in the confusion matrix and ROC chart.
- **Confusion matrix** – Summarizes the model accuracy for a given score threshold by comparing model predictions versus actual results. Amazon Fraud Detector assumes an example population of 100,000 events. The distribution of fraud and legitimate events simulates the fraud rate in your businesses.
  - **True positives** – The model predicts fraud and the event is actually fraud.
  - **False positives** – The model predicts fraud but the event is actually legitimate.
  - **True negatives** – The model predicts legitimate and the event is actually legitimate.
  - **False negatives** – The model predicts legitimate but the event is actually fraud.
  - **True positive rate (TPR)** – Percentage of total fraud the model detects. Also known as capture rate.
  - **False positive rate (FPR)** – Percentage of total legitimate events that are incorrectly predicted as fraud.
- **Receiver Operator Curve (ROC)** – Plots the true positive rate as a function of false positive rate over all possible model score thresholds. View this chart by choosing **Advanced Metrics**.
- **Area under the curve (AUC)** – Summarizes TPR and FPR across all possible model score thresholds. A model with no predictive power has an AUC of 0.5, whereas a perfect model has a score of 1.0.

To use the model performance metrics

1. Start with the **Score distribution** chart to review the distribution of model scores for your fraud and legitimate events. Ideally, you will have a clear separation between the fraud and legitimate events. This indicates the model can accurately identify which events are fraudulent and which are legitimate. Select a model threshold by clicking on the chart area. You can see how adjusting the model score threshold impacts your true positive and false positive rates.

   Note
   The score distribution chart plots the fraud and legitimate events on two different Y axis. The left Y axis represents the legitimate events and the right Y axis represents the fraud events.

2. Review the **Confusion matrix**. Depending on your selected model score threshold, you can see the simulated impact based on a sample of 100,000 events. The distribution of fraud and legitimate

events simulates the fraud rate in your businesses. Use this information to find the right balance between true positive rate and false positive rate.

3. For additional details, choose **Advanced Metrics**. Use the ROC chart to understand the relationship between true positive rate and false positive rate for any model score threshold. The ROC curve can help you fine-tune the tradeoff between true positive rate and false positive rate.

   Note
   You can also review metrics in table form by choosing **Table**.
   The table view also shows the metric **Precision**. **Precision** is the percentage of fraud events correctly predicted as fraudulent as compared to all events predicted as fraudulent.

4. Use the performance metrics to determine the optimal model thresholds for your businesses based on your goals and fraud-detection use case. For example, if you plan to use the model to classify new account registrations as either high, medium, or low risk, you need to identify two threshold scores so you can draft three rule conditions as follows:

   - Scores > X are high risk
   - Scores < X but > Y are medium risk
   - Scores < Y are low risk

Model variable importance

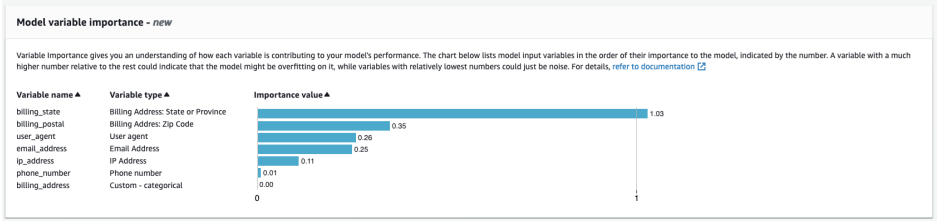
Model variable importance is a feature of Amazon Fraud Detector that ranks model variables within a model version. Each model variable is provided a value based on its relative importance to the overall performance of your model. The model variable with the highest value is more important to the model than the other model variables in the dataset for that model version, and is listed at the top by default. Likewise, the model variable with the lowest value is listed at the bottom by default and is least important compared to the other model variables. Using model variable importance values, you can gain insight into what inputs are driving your model's performance.

You can view model variable importance values for your trained model version in the Amazon Fraud Detector console or by using the DescribeModelVersion API.

Model variable importance provides the following set of values for each Variable used to train the Model Version.

- **Variable Name**: Name of the model variable that was used to train the model version (for example, `ip_address` or `email_address`)
- **Variable Type**: Type of variable (for example, IP address or Email)
- **Variable Importance Value**: A number that represents the relative importance of the variable to the model's performance. Typical range: 0–10

In the Amazon Fraud Detector console, the model variable importance values are displayed as follows. The visual chart makes it easy to see the relative importance between variables with the vertical dotted line providing reference to the importance value of the highest ranked variable.



Amazon Fraud Detector generates variable importance values for every Fraud Detector model version at no additional cost.

Important
Model versions that were created before *July 9, 2021* do not have variable importance values. You must train a new version of your model to generate the model variable importance values.

Using model variable importance values

You can use model variable importance values to gain insight into what is driving performance of your model up or down and which of variables contribute the most. And then tweak your model to improve overall performance.

More specifically, to improve your model performance, examine the variable importance values against your domain knowledge and debug issues in the training data. For example, if Account Id was used as an input to the model and it is listed at the top, take a look at its variable importance value. If the variable importance value is significantly higher than the rest of the values, then your model might be overfitting on a specific fraud pattern (for example, all the fraud events are from the same Account Id). However, it might also be the case that there is a label leakage if the variable depends on the fraud labels. Depending on the outcome of your analysis based on your domain knowledge, you might want to remove the variable and train with a more diverse dataset, or keep the model as it is.

Similarly, take a look at the variables ranked last. If the variable importance value is significantly lower than the rest of the values, then this model variable might not have any importance in training your model. You could consider removing the variable to train a simpler model version. If your model has few variables, such as only two variables, Amazon Fraud Detector still provides the variable importance values and rank the variables. However, the insights in this case will be limited.

Important

1. If you notice variables missing in the **Model variable importance** chart, it might be due to one of the following reasons. Consider modifying the variable in your dataset and retrain your model.
   - The count of unique values for the variable in the training dataset is lower than 100.
   - Greater than 0.9 of values for the variable are missing from the training data-set.
2. You need to train a new model version every time that you want to adjust your model's input variables.

Evaluating model variable importance values

We recommend that you consider the following when you evaluate model variable importance values:

- Variable importance values must always be evaluated in combination with the domain knowledge.
- Examine variable importance value of a variable relative to the variable importance value of the other variables within the model version. Do not consider variable importance value for a single variable independently.
- Compare variable importance values of the variables within the same model version. Do not compare variable importance values of the same variables across model versions because the variable importance value of a variable in a model version might differ from the value of the same variable in a different model version. If you use the same variables and dataset to train different model versions, this does not necessarily generate the same variable importance values.

Viewing model variable importance ranking

After model training is complete, you can view model variable importance ranking of your trained model version in the Amazon Fraud Detector console or by using the DescribeModelVersion API.

To view the model variable importance ranking using console,

1. Open the AWS Console and sign in to your account. Navigate to Amazon Fraud Detector.
2. In the left navigation pane, choose **Models**.
3. Choose your model and then your model version.
4. Make sure that the **Overview** tab is selected.
5. Scroll down to view the **Model variable importance** pane.

Understanding how the model variable importance value is calculated

Upon completion of each model version training, Amazon Fraud Detector automatically generates model variable importance values and model's performance metrics. For this, Amazon Fraud Detector uses SHapley Additive exPlanations (SHAP). SHAP is essentially the average expected contribution of a model variable after all possible combinations of all model variables have been considered.

SHAP first assigns contribution of each model variable for prediction of an event. Then, it aggregates these predictions to create a ranking of the variables at the model level. To assign contributions of each model variable for a prediction, SHAP considers differences in model outputs among all possible variable combinations. By including all possibilities of including or removing specific set of variables to generate a model output, SHAP can accurately access the importance of each model variable. This is particularly important when the model variables are highly correlated with one another.

ML models, in most cases, do not allow you to remove variables. You can instead replace a removed or missing variable in the model with the corresponding variable values from one or more baselines (for example, non-fraud events). Choosing proper baseline instances can be difficult, but Amazon Fraud Detector makes this easy by setting this baseline as the population average for you.

Import a SageMaker model

You can optionally import SageMaker-hosted models to Amazon Fraud Detector. Similar to models, SageMaker models can be added to detectors and generate fraud predictions using the `GetEventPrediction` API. As part of the `GetEventPrediction` request, Amazon Fraud Detector will invoke your SageMaker endpoint and pass the results to your rules.

You can configure Amazon Fraud Detector to use the event variables sent as part of the `GetEventPrediction` request. If you choose to use event variables, you must provide an input template. Amazon Fraud Detector will use this template to transform your event variables into the required input payload to invoke the SageMaker endpoint. Alternatively, you can configure your SageMaker model to use a byteBuffer that is sent as part of the `GetEventPrediction` request.

Amazon Fraud Detector supports importing SageMaker algorithms that use JSON or CSV input formats and JSON or CSV output formats. Examples of supported SageMaker algorithms include XGBoost, Linear Learner, and Random Cut Forest.

Import a SageMaker model using the AWS SDK for Python (Boto3)

To import a SageMaker model, use the `PutExternalModel` API. The following example assumes the SageMaker endpoint `sagemaker-transaction-model` has been deployed, is `InService` status, and uses the XGBoost algorithm.

The input configuration specifies that will use the event variables to construct the model input (`useEventVariables` is set to `TRUE`). The input format is TEXT_CSV, given XGBoost requires a CSV input. The csvInputTemplate specifies how to construct the CSV input from the variables sent as part of the `GetEventPrediction` request. This example assumes you have created the variables `order_amt`, `prev_amt`, `hist_amt` and `payment_type`.

The output configuration specifies the response format of the SageMaker model, and maps the appropriate CSV index to the Amazon Fraud Detector variable `sagemaker_output_score`. Once configured, you can use the output variable in rules.

Note
The output from a SageMaker model must be mapped to a variable with source `EXTERNAL_MODEL_SCORE`. You cannot create these variables in the console using **Variables**. You must instead create them when you configure your model import.

```
import boto3
fraudDetector = boto3.client('frauddetector')

fraudDetector.put_external_model (
modelSource = 'SAGEMAKER',
modelEndpoint = 'sagemaker-transaction-model',
invokeModelEndpointRoleArn = 'your_SagemakerExecutionRole_arn',
inputConfiguration = {
    'useEventVariables' : True,
    'eventTypeName' : 'sample_transaction',
    'format' : 'TEXT_CSV',
    'csvInputTemplate' : '{{order_amt}}, {{prev_amt}}, {{hist_amt}}, {{payment_type}}'
},

outputConfiguration = {
    'format' : 'TEXT_CSV',
    'csvIndexToVariableMap' : {
        '0' : 'sagemaker_output_score'
    }
},

modelEndpointStatus = 'ASSOCIATED'
)
```

Create a detector

A detector contains the detection logic, such as the models and rules, for a particular event that you want to evaluate for fraud. Each detector can evaluate one event type.

A detector can have multiple versions, with each version having a status of `DRAFT`, `ACTIVE`, or `INACTIVE`. Only one detector version can be in `ACTIVE` status at a time.

To create a detector using the AWS Console, see Step 1: Build detector (p. 14).

Create a detector using the AWS SDK for Python (Boto3)

The following example shows a sample request for the `PutDetector` API. A detector acts as a container for your detector versions. The `PutDetector` API specifies what event type the detector will evaluate. The following example assumes you have created an event type `sample_registration`.

```
import boto3
fraudDetector = boto3.client('frauddetector')

fraudDetector.put_detector (
detectorId = 'sample_detector',
eventTypeName = 'sample_registration'
)
```

After you create a detector, you can Create a rule (p. 55).

Create a detector version

A detector version defines the specific models and rules that will be run as part of the `GetEventPrediction` request. You can add any of the rules defined within a detector to the detector version. You can also add any model trained on the evaluated event type.

Each detector version has a status of `DRAFT`, `ACTIVE`, or `INACTIVE`. Only one detector version can be in `ACTIVE` status at a time. During the `GetEventPrediction` request, Amazon Fraud Detector will use the `ACTIVE` detector if no `DetectorVersion` is specified.

To create a detector using the AWS Console, see Step 1: Build detector (p. 14).

Rule execution mode

Amazon Fraud Detector supports two different rule execution modes: `FIRST_MATCHED` and `ALL_MATCHED`.

- If the rule execution mode is `FIRST_MATCHED`, Amazon Fraud Detector evaluates rules sequentially, first to last, stopping at the first matched rule. Amazon Fraud Detector then provides the outcomes for that single rule. If a rule evaluates to false (not matched), the next rule in the list is evaluated.
- If the rule execution mode is `ALL_MATCHED`, then all rules in an evaluation are executed in parallel, regardless of their order. Amazon Fraud Detector executes all rules and returns the defined outcomes for every matched rule.

Create a detector version using the AWS SDK for Python (Boto3)

The following example shows a sample request for the `CreateDetectorVersion` API. The example assumes you have created the rules described in Create a rule (p. 55) and the model `sample_fraud_detection_model` as described in Build a model (p. 47). The rule execution mode is set to `FIRST_MATCHED`, therefore Amazon Fraud Detector will evaluate rules sequentially, first to last, stopping at the first matched rule. Amazon Fraud Detector then provides the outcomes for that single rule during the `GetEventPrediction response`.

```
import boto3
fraudDetector = boto3.client('frauddetector')

fraudDetector.create_detector_version(
detectorId = 'sample_detector',
rules = [{
    'detectorId' : 'sample_detector',
    'ruleId' : 'high_fraud_risk',
    'ruleVersion' : '1'
},
{
    'detectorId' : 'sample_detector',
    'ruleId' : 'medium_fraud_risk',
    'ruleVersion' : '1'
},
{
    'detectorId' : 'sample_detector',
    'ruleId' : 'low_fraud_risk',
    'ruleVersion' : '1'
}
],
modelVersions = [{
    'modelId' : 'sample_fraud_detection_model',
    'modelType': 'ONLINE_FRAUD_INSIGHTS',
    'modelVersionNumber' : '1.00'
}],
ruleExecutionMode = 'FIRST_MATCHED'
)
```

To update the status of a detector version, use the `UpdateDetectorVersionStatus` API. The following example updates the detector version status from `DRAFT` to `ACTIVE`. During a `GetEventPrediction` request, if a detector ID is not specified, Amazon Fraud Detector will use the `ACTIVE` version of the detector.

```
import boto3
fraudDetector = boto3.client('frauddetector')

fraudDetector.update_detector_version_status(
detectorId = 'sample_detector',
detectorVersionId = '1',
status = 'ACTIVE'
)
```

Create a rule

A rule is a condition that tells Amazon Fraud Detector how to interpret variable values during a fraud prediction. A rule consists of one or more variables, a logic expression, and one or more outcomes. A

detector must have at least one associated rule. Rules in a detector are evaluated as part of a fraud prediction.

Expressions

Each rule must contain a single expression that captures your business logic. All expressions must evaluate to a Boolean value (true or false) and be less than 4,000 characters in length. If-else type conditions are not supported. All variables used in the expression must be predefined in the evaluated event type.

To create a rule using the AWS Console, see Step 3: Add rules (p. 14).

Create a rule using the AWS SDK for Python (Boto3)

The following example creates three different rules (`high_fraud_risk`, `medium_fraud_risk`, and `low_fraud_risk`). The example assumes you have completed the steps outlined in Build a model (p. 47) and created the model `sample_fraud_detection_model`. The example also assumes you have created the outcomes `verify_customer`, `review`, and `approve` (see Create an outcome (p. 73) for details on how to create an outcome). The expressions compare the model score output variable `sample_fraud_detection_model_insightscore` against various thresholds to determine the level of risk for an event.

```
import boto3
fraudDetector = boto3.client('frauddetector')

fraudDetector.create_rule(
ruleId = 'high_fraud_risk',
detectorId = 'sample_detector',
expression = '$sample_fraud_detection_model_insightscore > 900',
language = 'DETECTORPL',
outcomes = ['verify_customer']
)

fraudDetector.create_rule(
ruleId = 'medium_fraud_risk',
detectorId = 'sample_detector',
expression = '$sample_fraud_detection_model_insightscore <= 900 and
 $sample_fraud_detection_model_insightscore > 700',
language = 'DETECTORPL',
outcomes = ['review']
)

fraudDetector.create_rule(
ruleId = 'low_fraud_risk',
detectorId = 'sample_detector',
expression = '$sample_fraud_detection_model_insightscore <= 700',
language = 'DETECTORPL',
outcomes = ['approve']
)
```

After you create the rules, you can Create a detector version (p. 54).

You can update a rule by calling the `UpdateRuleVersion` API. The following example updates the model score thresholds for the rules `high_fraud_risk` and `medium_fraud_risk` from 900 to 950.

```
fraudDetector.update_rule_version(
rule = {
    'detectorId' : 'sample_detector',
    'ruleId' : 'high_fraud_risk',
```

```
    'ruleVersion' : '1'
},
expression = '$sample_fraud_detection_model_insightscore > 950',
language = 'DETECTORPL',
outcomes = ['verify_customer']
)

fraudDetector.update_rule_version(
rule = {
    'detectorId' : 'sample_detector',
    'ruleId' : 'medium_fraud_risk',
    'ruleVersion' : '1'
},
expression = '$sample_fraud_detection_model_insightscore <= 950 and
 $sample_fraud_detection_model_insightscore > 700',
language = 'DETECTORPL',
outcomes = ['review']
)
```

Rule language reference

The following section outlines the expression (that is, rule writing) capabilities in Amazon Fraud Detector.

Using variables

You can use any variable defined in the evaluated event type as part of your expression. Use the dollar sign to indicate a variable:

```
$example_variable < 100
```

Comparison, membership, and identity operators

Amazon Fraud Detector includes the following comparison operators: >, >=, <, <=,!=, ==, in, not in

The following are examples:

Example: <

```
$variable < 100
```

Example: in, not in

```
$variable in [5, 10, 25, 100]
```

Example: !=

```
$variable != "US"
```

Example: ==

```
$variable == 1000
```

Operator Tables

Operator	Amazon Fraud Detector Operator
Equal to	==
Not equal to	!=
Greater than	>
Less than	<
Great than or equal to	>=
Less than or equal to	<=
In	in
And	and
Or	or
Not	!

Basic math

You can use basic math operators in your expression (for example, +, -, * ,/). A typical use case is when you need to combine variables during your evaluation.

In the rule below, we are adding the variable `$variable_1` with `$variable_2`, and checking whether the total is less than 10.

```
$variable_1 + $variable_2 < 10
```

Basic Math Table Data

Operator	Amazon Fraud Detector Operator
Plus	+
Minus	-
Multiply	*
Divide	/
Modulo	%

Regular Expression (regex)

You can use regex to search for specific patterns as part of your expression. This is particularly useful if you are looking to match a specific string or numerical value for one of your variables. Amazon Fraud Detector only supports match when working with regular expressions (for example, it returns True/False depending on whether the provided string is matched by the regular expression). Amazon Fraud Detector's regular expression support is based on .matches() in java (using the RE2J Regular Expression library). There are several helpful websites on the internet that are useful for testing different regular expression patterns.

In the first example below, we first transform the variable `email` to lowercase. We then check whether the pattern `@gmail.com` is in the `email` variable. Notice the second period is escaped so that we can explicitly check for the string `.com`.

```
regex_match(".*@gmail\.com", lowercase($email))
```

In the second example, we check whether the variable `phone_number` contains the country code `+1` to determine if the phone number is from the US. The plus symbol is escaped so that we can explicitly check for the string `+1`.

```
regex_match(".*\+1", $phone_number)
```

Regex Table

Operator	Amazon Fraud Detector Example
Match any string that starts with	regex_match("^mystring", $variable)
Match entire string exactly	regex_match("mystring", $variable)
Match any character except new line	regex_match(".", $variable)
Match any number of characters except new line prior 'mystring'	regex_match(".*mystring", $variable)
Escape special characters	\

Checking for missing values

Sometimes it is beneficial to check whether the value is missing. In Amazon Fraud Detector this is represented by null. You can do this by using the following syntax:

```
$variable != null
```

Similarly, if you wanted to check whether a value is not present, you could do the following:

```
$variable == null
```

Multiple conditions

You can combine multiple expressions together using `and` and `or`. Amazon Fraud Detector stops in an `OR` expression when a single true value is found, and it stops in an `AND` when a single false value is found.

In the example below, we are checking for two conditions using the `and` condition. In the first statement, we are checking whether variable 1 is less than 100. In the second we check whether variable 2 is not the US.

Given the rule uses an `and`, both must be TRUE for the entire condition to evaluate to TRUE.

```
$variable_1 < 100 and $variable_2 != "US"
```

You can use parenthesis to group Boolean operations, as shown following:

```
$variable_1 < 100 and $variable_2 != "US" or ($variable_1 * 100.0 > $variable_3)
```

Other expression types

String Operators

Operator	Example
Transform string to uppercase	uppercase($variable)
Transform string to lowercase	lowercase($variable)

Other

Operator	Comment
Add a comment	# my comment

Create resources

Models, rules, and detectors use resources such as variables, outcomes, labels, and entities to evaluate events for fraud risk. This section provides information about creating and managing the resources.

Topics

- Create a variable (p. 61)
- Create an outcome (p. 73)
- Create an entity type (p. 74)
- Create a label (p. 74)

Create a variable

Variables represent data elements that you want to use in a fraud prediction. These variables can be taken from the event dataset that you prepared for training your model, from your Amazon Fraud Detector model's risk score outputs, or from Amazon SageMaker models. For more information about variables taken from the event dataset, see Prepare event dataset (p. 25).

The variables you want to use in your fraud prediction must first be created and then added to the event when creating your event type. Each variable you create must be assigned a datatype, a default value, and optionally a variable type. Amazon Fraud Detector enriches some of the variables that you provide such as IP addresses, bank identification numbers (BINs), and phone numbers, to create additional inputs and boost performance for the models that use these variables.

Data types

Variables must have a data type for the data element that the variable represents and can optionally be assigned one of the predefined Variable types (p. 62). For variables that are assigned to a variable type, the data type is pre-selected. Possible data types include the following types :

Data type	Description	Default value	Example values
String	Any combination of letters, whole numbers, or both	<empty>	abc, 123, 1D3B
Integer	Positive or negative whole numbers	0	1, -1
Boolean	True or False	False	True, False
Float	Numbers with decimal points	0.0	4.01, 0.10

Default value

Variables must have a default value. When Amazon Fraud Detector generates fraud predictions, this default value is used to run a rule or model if Amazon Fraud Detector doesn't receive a value for a variable. Default values you provide must match the selected data type. In the AWS Console, Amazon Fraud Detector assigns the default value of `0` for integers, `false` for Booleans, `0.0` for floats, and (empty) for strings. You can set a custom default value for any of these data types.

Variable types

When you create a variable, you can optionally assign the variable to a variable type. Variable type represents the common data elements that are used to train models and to generate fraud predictions. Only variables with an associated variable type can be used for model training. As part of the model training process, Amazon Fraud Detector uses the variable type associated with the variable to perform variable enrichments, feature engineering, and risk scoring.

Amazon Fraud Detector has pre-defined the following varaible types that can be used to assign to your variables.

Cat	Variable type	Description	Data type	Exa
Email	EMAIL_ADDRESS	The email address that's collected during the event	String	abc@domain.com
IP address	IP_ADDRESS	The IP address that's collected during the event	String	1.1.1.1 **Note:** Amazon Fraud Detector enriches this data. For more information, see Geolocation enrichment (p. 68)
Phone number	PHONE_NUMBER	The phone number that's collected during the event	String	+1 555-0100 **Note:** Amazon Fraud Detector enriches this data. For more information, see Phone number enrichment (p. 67)
Browser and device	USERAGENT	The user agent that's collected during the event	String	Mozilla 5.0 (Windows NT 10.0,

Cat	Variable type	Description	Data type	Exa
				Win64, x64,rv:68.0) Gecko 20100101
	FINGERPRINT	The unique identifier for a device	String	sadfow987u234
Payment method	PAYMENT_TYPE	The payment method that's used for payment during the event	String	Credit card
	CARD_BIN	The first six digits of the credit card	Integer	123456
	AUTH_CODE	The alphanumerical code that's sent by a credit card issuer or issuing bank	String	00
	AVS	The address verification system (AVS) response code from the card processor	String	Y
Billing address	BILLING_NAME	The name that's associated with the billing address	String	John Doe
	BILLING_PHONE	The phone number that's associated with the billing address	String	+1 555-0100 **Note:** Amazon Fraud Detector enriches this data. For more information, see Phone number enrichment (p. 67)
	BILLING_ADDRESS_L1	The first line of the billing address	String	123 Any Street
	BILLING_ADDRESS_L2	The second line of the billing address	String	Unit 123

Cat	Variable type	Description	Data type	Exa
	BILLING_CITY	The city that's in the billing address	String	Any City
	BILLING_STATE	The state or province that's in the billing address	String	Any State
	BILLING_COUNTRY	The country that's in the billing address	String	US **Note:** Amazon Fraud Detector enriches this data. For more information, see Geolocation enrichment (p. 68)
	BILLING_ZIP	The postal code that's in the billing address	String	01234 **Note:** Amazon Fraud Detector enriches this data. For more information, see Geolocation enrichment (p. 68)
Shipping address	SHIPPING_NAME	The name that's associated with the shipping address	String	John Doe

Cat	Variable type	Description	Data type	Exa
	SHIPPING_PHONE	The phone number that's associated with the shipping address	String	+1 555-0100 **Note:** Amazon Fraud Detector enriches this data. For more information, see Phone number enrichment (p. 67)
	SHIPPING_ADDRESS_L1	The first line of the shipping address	String	123 Any Street
	SHIPPING_ADDRESS_L2	The second line of the shipping address	String	Unit 123
	SHIPPING_CITY	The city that's in the shipping address	String	Any City
	SHIPPING_STATE	The state or province that's in the shipping address	String	Any State
	SHIPPING_COUNTRY	The country that's in the billing address	String	Any Country **Note:** Amazon Fraud Detector enriches this data. For more information, see Geolocation enrichment (p. 68)

Cat	Variable type	Description	Data type	Exa
	SHIPPING_ZIP	The postal code that's in the shipping address	String	01234 **Note:** Amazon Fraud Detector enriches this data. For more information, see Geolocation enrichment (p. 68)
Order	ORDER_ID	The unique identifier for the transaction	String	LUX60
	PRODUCT_CATEGORY	The product category of order item	String	kitchen
	CURRENCY_CODE	The ISO 4217 currency code	String	USD
	PRICE	The total order price	String	560.00
Custom	NUMERIC	Any variable that can be represented as a real number	Float	1.224
	CATEGORICAL	Any variable that describes categories, segments, or groups	String	Large
	FREE_FORM_TEXT	Any free form text that's captured as part of the event (for example, a customer review or comment)	String	Example of a free form text input

Assigning variable to a variable type

If you are planning to use a variable for training your model, it is important that you choose a right variable type to assign to the variable. Incorrect variable type assignment can negatively impact your model performance. It can also become very difficult for you change the assignment later, especially if multiple models and events have used the variable.

You can assign your variable any one of the pre-defined variable types or one of the custom variable types – `FREE_FORM_TEXT`, `CATEGORICAL`, or `NUMERIC`.

Important notes for assigning variables to the right variable types

1. If the variable matches one of predefined variable types, use it. Make sure the variable type corresponds to the variable. For example, if you assign an *ip_address* variable to `EMAIL_ADDRESS` variable type, the ip_address variable will not get enriched with enrichments such as ASN, ISP, geo-location, and risk score. For more information, see Variable enrichments (p. 67).
2. If the variable doesn't match any of predefined variable types, follow the recommendations listed below to assign one of the custom variable types.
3. Assign `CATEGORICAL` variable type to variables that typically do not have natural ordering and can be put into categories, segments, or groups. The dataset you are using to train your model might have ID variables such as, *merchant_id*, *campaign_id*, or *policy_id*. These variables represent groups (for example, all customers with same policy_id represent a group). Variables that have the following data must be assigned CATEGORICAL variable type -
   - Variables that contain data such as *customer_ID*, *segment_ID*, *color_ID*, *department_code*, or *product_ID*.
   - Variables that contain Boolean data with true, false, or null values.
   - Variables that can be put into groups or categories such as company name, product category, card type, or referral medium.

     Note
     `ENTITY_ID` is a reserved variable type used by Amazon Fraud Detector to assign to ENTITY_ID variable. The ENTITY_ID variable is the ID of the entity initiating the action you want to evaluate. If you are creating a Transaction Fraud Insight (TFI) model type, you are required to provide ENTITY_ID variable. You will need to decide which variable in your data uniquely identifies the entity initiating the action and pass it on as ENTITY_ID variable. Assign CATEGORICAL variable type to all the other IDs in your dataset, if they are present and if you are using them for model training. Examples of other IDs that are not an entity in your dataset can be *merchant_ID*, *policy_ID*, and *campaign_ID*.
4. Assign `FREE_FORM_TEXT` variable type to variables that contain a block of text. Examples of FREE_FORM_TEXT variable types are – *user reviews*, *comments*, *dates*, and *referral codes*. The FREE_FORM_TEXT data contains multiple tokens separated by a delimiter. The delimiters can be any character other than alpha-numeric and underscore symbol. For example, user reviews and comments can be separated by "space" delimiter, dates and referral codes can use hyphens as delimiters to separate out prefix, suffix, and intermediate parts. Amazon Fraud Detector uses the delimiters to extract data from FREE_FORM_TEXT variables.
5. Assign *NUMERIC* variable type to variables that are real numbers and have inherent ordering. Examples of NUMERIC variables include *day_of_the_week*, *incident_severity*, *customer_rating*. Although, you can assign CATEGORICAL variable type to these variables, we strongly recommend to assign all real number variables with inherent order to NUMERIC variable type.

Variable enrichments

Amazon Fraud Detector enriches some of the raw data elements that you provide such as IP addresses, bank identification numbers (BINs), and phone numbers, to create additional inputs and boost performance for the models that use these data elements. The enrichment helps identify potentially suspicious situations and help the models to capture more fraud.

Phone number enrichment

Amazon Fraud Detector enriches phone number data with additional information that relates to geolocation, the original carrier, and the validity of the phone number. Phone number enrichment is automatically enabled for all the models that are trained on or after *December 13, 2021* and have a

phone number that includes a country code (+xxx). If you have included phone number variable in your model and have trained it before *December 13, 2021*, retrain your model so it can take advantage of this enrichment.

We highly recommend that you use the following format for phone number variables to ensure that your data is enriched successfully.

Variable	Format	Description
PHONE_NUMBER	The E.164 standard	Make sure to include cou with the phone number.
BILLING_PHONE and SHIPPING_PHONE	The E.164 standard	Make sure to include cou with the phone number.

Geolocation enrichment

Starting on *February 8, 2022* Amazon Fraud Detector calculates the physical distance between the IP_ADDRESS, BILLING_ZIP, and SHIPPING_ZIP values that you provide for an event. The calculated distances are used as inputs to your fraud detection model.

To enable geolocation enrichment, your event data must include at least two of the three variables: IP_ADDRESS, BILLING_ZIP, or SHIPPING_ZIP. In addition, each BILLING_ZIP and SHIPPING_ZIP value must have a valid BILLING_COUNTRY code and SHIPPING_COUNTRY code respectively. If you have a model that was trained before *February 8, 2022* and it includes these variables, you must retrain the model to enable the geolocation enrichment.

If Amazon Fraud Detector can't determine the location that's associated with the IP_ADDRESS, BILLING_ZIP ,or SHIPPING_ZIP values for an event due to the data being not valid, a special placeholder value is used instead. For example, suppose that an event has valid IP_ADDRESS and BILLING_ZIP values, but SHIPPING_ZIP value isn't valid. In this case, enrichment is done only for IP_ADDRESS–> BILLING_ZIP. The enrichment isn't done for IP_ADDRESS–>SHIPPING_ZIP and BILLING_ZIP–>SHIPPING_ZIP . Instead, the placeholder values are used in their place. No matter if geolocation enrichment is enabled for your model or not, the performance of your model doesn't change.

You can opt out of geolocation enrichment by mapping your BILLING_ZIP and SHIPPING_ZIP variables to the CUSTOM_CATEGORICAL variable type. Changing the variable type doesn't affect your model's performance.

Geolocation variable format

We highly recommend that you use the following format for geolocation variables to ensure that your location data is enriched successfully.

Variable	Format	Description
IP_ADDRESS	IPv4 address	For example - 1.1.1.1
BILLING_ZIP and SHIPPING_ZIP	The ISO 3166-1 alpha-2 postal code for the specified country	For more information, se **and territory codes** sect
BILLING_COUNTRY and SHIPPING_COUNTRY	The ISO 3166-1 alpha-2 two-letter standard country code	For more information, se **and territory codes** secti Amazon Fraud Detector t all the common variation name to their ISO 3166-1 standard country code. H

Variable	Format	Description
		cannot guarantee they w correctly.

Country and territory codes

The following table provides a complete list of the countries and territories that are supported by Amazon Fraud Detector for geolocation enrichment. Each country and territory has an assigned country code (specifically, the ISO 3166-1 alpha-2 two-letter country code) and a postal code.

Postal code format

- 9 - number
- a - letter
- [X] - X is optional. For example, Guersney "GY9[9] 9aa" means both "GY9 9aa" and "GY99 9aa" are valid. Use one format.
- [X/XX] - either X or XX can be used. For example, Bermuda "aa[aa/99]" means both "aa aa" and "aa 99" are valid. Use either one of these formats, but *do not* use both.
- Some countries have fixed prefix. For example, the postal code for Andorra is AD999. This means the country code must start with letters *AD* followed by three numbers.

Code	Name	Postal code
AD	Andorra	AD999
AR	Netherlands Antilles	9999
AT	Austria	9999
AU	Australia	9999
AZ	Azerbaijan	AZ 9999
BD	Bangladesh	9999
BE	Belgium	9999
BG	Bulgaria	9999
BM	Bermuda	aa[aa/99]
BY	Belarus	999999
CA	Canada	a9a 9a9
CH	Switzerland	9999
CL	Chile	9999999
CO	Colombia	999999
CR	Costa Rica	99999
CY	Cyprus	9999
CZ	Czechia	999 99

Code	Name	Postal code
DE	Germany	99999
DK	Denmark	9999
DO	Dominican Republic	99999
DZ	Algeria	99999
EE	Estonia	99999
ES	Spain	99999
FI	Finland	99999
FM	Federated States of Micronesia	99999
FO	Faroe Islands	999
FR	France	99999
GB	United Kingdom	a[a]9[a/9] 9aa
GG	Guernsey	GY9[9] 9aa
GL	Greenland	9999
GP	Guadeloupe	99999
GT	Guatemala	99999
GU	Guam	99999
HR	Croatia	99999
HU	Hungary	9999
IE	Ireland	a99[a/9][a/9][a/9][a/9]
IM	Isle of Man	IM9[9]9aa
IN	India	999999
IS	Iceland	999
IT	Italy	99999
JE	Jersey	JE9[9]9aa
JP	Japan	999-9999
KR	Republic of Korea	99999
LI	Liechtenstein	9999
LK	Sri Lanka	99999
LT	Lithuania	99999
LU	Luxembourg	L-9999
LV	Latvia	LV-9999

Code	Name	Postal code
MC	Monaco	99999
MD	Republic of Moldova	9999
MH	Marshall Islands	99999
MK	North Macedonia	9999
MP	North Mariana Islands	99999
MQ	Matinique	99999
MT	Malta	aaa 9999
MX	Mexico	99999
MY	Malaysia	99999
NL	Netherlands	9999 aa
NO	Norway	9999
NZ	New Zealand	9999
PH	Philippines	9999
PK	Pakistan	99999
PL	Poland	99-999
PR	Puerto Rico	99999
PT	Portugal	9999-999
PW	Palau	99999
RE	Reunion	99999
RO	Romania	999999
RU	Russian Federation	999999
SE	Sweden	999 99
SG	Singapore	999999
SI	Slovenia	9999
SK	Slovakia	999 99
SM	San Marino	99999
TH	Thailand	99999
TR	Turkey	99999
UA	Ukraine	99999
US	United States	99999
UY	Uruguay	99999

Code	Name	Postal code
VI	Virgin Islands, US	99999
WF	Wallis and Futuna	99999
YT	Mayotte	99999
ZA	South Africa	9999

Create a variable using the Amazon Fraud Detector console

This example creates two variables, `email_address` and `ip_address`, and assigns them to the corresponding variable types (`EMAIL_ADDRESS` and `IP_ADDRESS`). These variables are used as examples. If you are creating variables to use for your model training, use the variables from your dataset that are appropriate for your use case. Make sure to read about Variable types (p. 62) and Variable enrichments (p. 67) before you create your variables.

1. Open the AWS Management Console and sign in to your account. Navigate to Amazon Fraud Detector.
2. Navigate to Amazon Fraud Detector, choose **Variables** in the left navigation, then choose **Create**.
3. In the **New variable** page, enter `email_address` as the variable name. Optionally, enter a description of the variable.
4. In the **Variable type**, choose **Email Address**.
5. Amazon Fraud Detector automatically selects the data type for this variable type because this variable type is predefined. For more information, see Variable types (p. 62).
6. If you want to provide a default value for your variable, select **Define a custom default value** and enter a default value for your variable. Skip this step if you are following this example.
7. Choose **Create**.
8. In the **email_address** overview page, confirm the details of the variable you just created.

   If you need to update, choose **Edit** and provide the updates. Choose **Save changes**.
9. Repeat the process to create another variable `ip_address` and choose **IP Address** for the variable type.
10. The **Variables** page shows the newly created variables.

Create a variable using the AWS SDK for Python (Boto3)

The following example shows requests for the CreateVariable API. The example creates two variables, `email_address` and `ip_address`, and assigns them to the corresponding variable types (`EMAIL_ADDRESS` and `IP_ADDRESS`).

These variables are used as examples. If you are creating variables to use for your model training, use the variables from your dataset that are appropriate for your use case. Make sure to read about Variable types (p. 62) and Variable enrichments (p. 67) before you create your variables.

Be sure to specify a variable source. It helps to identify where the variable value is derived. If the variable source is **EVENT**, the variable value is sent as part of the GetEventPrediction request. If the variable value

is `MODEL_SCORE`, it's populated by an Amazon Fraud Detector. If `EXTERNAL_MODEL_SCORE`, the variable value is populated by an imported SageMaker model.

```
import boto3
fraudDetector = boto3.client('frauddetector')

 #Create variable email_address
   fraudDetector.create_variable(
     name = 'email_address',
     variableType = 'EMAIL_ADDRESS',
     dataSource = 'EVENT',
     dataType = 'STRING',
     defaultValue = '<unknown>'
     )

#Create variable ip_address
   fraudDetector.create_variable(
     name = 'ip_address',
     variableType = 'IP_ADDRESS',
     dataSource = 'EVENT',
     dataType = 'STRING',
     defaultValue = '<unknown>'
     )
```

Create an outcome

An outcome is the result of a fraud prediction. You can create an outcome for each possible fraud prediction result. For example, you might want outcomes to represent risk levels (high_risk, medium_risk, and low_risk) or actions (approve, review). After an outcome is created, you can add one or more outcomes to a rule. As part of the GetEventPrediction response, Amazon Fraud Detector returns the defined outcomes for any matched rule.

To create an outcome using the AWS Console, open the AWS Console and sign in to your account. Navigate to Amazon Fraud Detector and, in the left navigation pane, choose **Outcomes**, then choose **Create**.

Create an outcome using the AWS SDK for Python (Boto3)

The following example shows requests for the `CreateOutcome` API. In the following example, three outcomes are created. They are `verify_customer`, `review`, and `approve`. You can assign these outcomes to rules.

```
import boto3
fraudDetector = boto3.client('frauddetector')

fraudDetector.put_outcome(
name = 'verify_customer',
description = 'this outcome initiates a verification workflow'
)

fraudDetector.put_outcome(
name = 'review',
description = 'this outcome sidelines event for review'
)

fraudDetector.put_outcome(
```

```
name = 'approve',
description = 'this outcome approves the event'
)
```

Create an entity type

An entity represents a person or thing that's active in an event. As part of a fraud prediction, you can pass the entity ID to indicate the specific entity who performed the event.

An entity type classifies the entity. Example classifications include customer, merchant, or account.

To create an entity type using the AWS Console, open the AWS Console and sign in to your account. Navigate to Amazon Fraud Detector and in the left navigation pane, choose **Entities**, then choose **Create**.

Create an entity type using the AWS SDK for Python (Boto3)

The following example shows a request for the `PutEntityType` API. In the example, the entity type that's called `sample_customer` is created. You can associate this entity with an event type to describe who is performing the event.

```
import boto3
fraudDetector = boto3.client('frauddetector')

fraudDetector.put_entity_type(
name = 'sample_customer',
description = 'sample customer entity type'
)
```

Create a label

A label classifies an event as fraudulent or legitimate. After you created a label, add the label to the event type by calling the PutEventType API.

You can use labels to train supervised machine learning models in Amazon Fraud Detector. As part of model version configuration, you classify the event labels as either fraudulent or legitimate. The supervised model learns to classify events using the variable values.

To create a label using the AWS Console, open the AWS Console and sign in to your account. Navigate to Amazon Fraud Detector and, in the left navigation pane, choose **Labels**, then choose **Create**.

Create a label using the AWS SDK for Python (Boto3)

The following example shows requests for the PutLabel API. In the example, two labels (fraud, legit) are created. You can add these labels to an event type to classify specific events.

```
import boto3
fraudDetector = boto3.client('frauddetector')

fraudDetector.put_label(
name = 'fraud',
description = 'label for fraud events'
```

```
)

fraudDetector.put_label(
name = 'legit',
description = 'label for legitimate events'
)
```

Delete resources

This section provides information about deleting resources.

When you delete a resource, Amazon Fraud Detector permanently deletes that resource and the data is no longer stored in Amazon Fraud Detector.

Some delete operations require you to first delete associated resources. For example, prior to deleting a model, you must first delete the model versions. See the following table for additional details.

If you want to delete this resource	You must delete this first	Notes
Detector	All associated detector versions and rules	
Detector version		You can only delete detector versions that are in `DRAFT` or `INACTIVE` status.
Model	All associated model versions	
Model version	Associated detector version	You can only delete model versions that are in `Ready to deploy` status. To change a model version from `Active` to `Ready to deploy` status, undeploy the model version after removing it from any detector version.
Amazon SageMaker model		Deleting an imported SageMaker model from Amazon Fraud Detector does not delete the model's endpoint within SageMaker.
Model output variable		Amazon Fraud Detector will automatically delete model output variables when you delete the model.
SageMaker output variable	Associated SageMaker model	
Event type	Associated detector or model, as well as any stored events	
Event variable	Associated event type	
Entity type	Associated event type	
Label	Associated event type	

If you want to delete this resource	You must delete this first	Notes
Rule version	Associated detector version	
Outcome	Associated rule version	
Event		Amazon Fraud Detector deletes all event data associated with the event, including any event data sent to `SendEvent` operation and any prediction data generated through `GetEventPrediction` operation.

Topics

- Delete a detector, detector version, or rule version (p. 77)
- Delete a model or model version (p. 78)
- Delete an event or event type (p. 79)
- Delete an entity type (p. 80)
- Delete an outcome (p. 80)
- Delete a label (p. 81)
- Delete a variable (p. 81)
- Delete stored event data (p. 81)

Delete a detector, detector version, or rule version

Before deleting a detector in Amazon Fraud Detector, you must first delete all detector versions and rule versions that are associated with the detector.

When you delete a detector, detector version, or rule version, Amazon Fraud Detector permanently deletes that resource and the data is no longer stored in Amazon Fraud Detector.

To delete a detector version

You can only delete detector versions that are in `DRAFT` or `INACTIVE` status.

1. Sign in to the AWS Management Console and open the Amazon Fraud Detector console at https://console.aws.amazon.com/frauddetector.
2. In the left navigation pane of the Amazon Fraud Detector console, choose **Detectors**.
3. Choose the detector that contains the detector version you want to delete.
4. Choose the detector version that you want to delete.
5. Choose **Actions**, and then choose **Delete**.
6. Enter **`delete`**, and then choose **Delete detector**.

To delete a rule version

You can delete a rule version only if it is not used by any `ACTIVE` or `INACTIVE` detector versions. If necessary, before deleting a rule version, first move the `ACTIVE` detector version to `INACTIVE`, then delete the `INACTIVE` detector version.

1. In the left navigation pane of the Amazon Fraud Detector console, choose **Detectors**.
2. Choose the detector that contains the rule version you want to delete.
3. Choose the **Associated rules** tab, and choose the rule that you want to delete.
4. Choose the rule version that you want to delete.
5. Choose **Actions**, and then choose **Delete rule version**.
6. Enter **`delete`**, and then choose **Delete version**.

To delete a detector

Before deleting a detector, you must first delete all detector versions and rule versions that are associated with the detector.

1. In the left navigation pane of the Amazon Fraud Detector console, choose **Detectors**.
2. Choose the detector that you want to delete.
3. Choose **Actions**, and then choose **Delete detector**.
4. Enter **`delete`**, and then choose **Delete detector**.

Delete a model or model version

You can delete models and model versions in Amazon Fraud Detector, provided that they are not associated with a detector version. When you delete a model, Amazon Fraud Detector permanently deletes that model and the data is no longer stored in Amazon Fraud Detector.

You can also remove Amazon SageMaker models if they are not associated with a detector version. Removing a SageMaker model disconnects it from Amazon Fraud Detector, but the model remains available in SageMaker.

To delete a model version

You can only delete model versions that are in the `Ready to deploy` status. To change a model version from `ACTIVE` to `Ready to deploy` status, undeploy the model version.

1. Sign in to the AWS Management Console and open the Amazon Fraud Detector console at https://console.aws.amazon.com/frauddetector.
2. In the left navigation pane of the Amazon Fraud Detector console, choose **Models**.
3. Choose the model that contains the model version you want to delete.
4. Choose the model version that you want to delete.
5. Choose **Actions**, and then choose **Delete**.
6. Enter the model version name, and then choose **Delete model version**.

To undeploy a model version

You can't undeploy a model version that is in use by any detector version (`ACTIVE`, `INACTIVE`, `DRAFT`). Therefore, to undeploy a model version that is in use by a detector version, first remove the model version from the detector version.

1. In the left navigation pane of the Amazon Fraud Detector console, choose **Models**.
2. Choose the model that contains the model version you want to undeploy.
3. Choose the model version that you want to delete.
4. Choose **Actions**, and then choose **Undeploy model version**.

To delete a model

Before deleting a model, you must first delete all model versions and are associated with the model.

1. In the left navigation pane of the Amazon Fraud Detector console, choose **Models**.
2. Choose the model that you want to delete.
3. Choose **Actions**, and then choose **Delete**.
4. Enter the model name, and then choose **Delete model**.

To remove an Amazon SageMaker model

1. In the left navigation pane of the Amazon Fraud Detector console, choose **Models**.
2. Choose the SageMaker model that you want to remove.
3. Choose **Actions**, and then choose **Remove model**.
4. Enter the model name and then choose **Remove SageMaker model**.

Delete an event or event type

When you delete an event, Amazon Fraud Detector permanently deletes that event and the data associated with the event is no longer stored in Amazon Fraud Detector.

To delete an event that Amazon Fraud Detector has evaluated through `GetEventPrediction` API

1. Sign in to the AWS Management Console and open the Amazon Fraud Detector console at https://console.aws.amazon.com/frauddetector.
2. In the left navigation pane of the console, choose **Search past predictions**.
3. Choose the event that you want to delete.
4. Choose **Actions**, and then choose **Delete event**.
5. Enter **`delete`**, and then choose **Delete event**.

> **Note**
> This deletes all records that are associated to that Event ID, including any event data sent to the `SendEvent` operation and any prediction data generated through the `GetEventPrediction` operation.

To delete an event that is stored in Amazon Fraud Detector but has not been evaluated (that is, it was stored via the `SendEvent` operation), you must make a `DeleteEvent` request and specify the Event ID and Event Type ID. If you want to delete both the event and any prediction history associated with the event, set the value of the `deleteAuditHistory` parameter to "*true*".

To delete all events associated with an event type

1. In the left navigation pane of the console, choose **Event types**
2. Choose the event type for which you want all events deleted.
3. Navigate to **Stored events** tab and choose **Delete stored events**

Depending on the number of stored events for the event type, it might take some time to delete all stored events. For example, a 1 GB dataset (approximately 1-2 million events for the average customer) takes around 2 hours to delete. During this time, new events that you send to Amazon Fraud

Detector of this event type are not stored, but you can continue to generate fraud predictions via the `GetEventPrediction` operation.

To delete an event type

You cannot delete an event type that is used in a detector or a model, or has associated stored events. Before deleting an event type, you must delete all events that are associated with that event type.

When you delete an event type, Amazon Fraud Detector permanently deletes that event type and the data is no longer stored in Amazon Fraud Detector.

1. In the left navigation pane of the Amazon Fraud Detector console, choose **Resources**, and then choose **Events**.
2. Choose the event type that you want to delete.
3. Choose **Actions**, and then choose **Delete event type**.
4. Enter the event type name, and then choose **Delete event type**.

Delete an entity type

In Amazon Fraud Detector, you cannot delete an entity type that is included in an event type.

When you delete an entity type, Amazon Fraud Detector permanently deletes that entity type and the data is no longer stored in Amazon Fraud Detector.

To delete an entity type

1. Sign in to the AWS Management Console and open the Amazon Fraud Detector console at https://console.aws.amazon.com/frauddetector.
2. In the left navigation pane of the Amazon Fraud Detector console, choose **Resources**, then choose **Entities**.
3. Choose the entity type that you want to delete.
4. Choose **Actions**, and then choose **Delete**.
5. Enter the entity type name, and then choose **Delete entity type**.

Delete an outcome

You cannot delete an outcome that is used in a rule version.

When you delete an outcome, Amazon Fraud Detector permanently deletes that outcome and the data is no longer stored in Amazon Fraud Detector.

To delete an outcome

1. Sign in to the AWS Management Console and open the Amazon Fraud Detector console at https://console.aws.amazon.com/frauddetector.
2. In the left navigation pane of the Amazon Fraud Detector console, choose **Resources**, and then choose **Outcomes**.
3. Choose the outcome that you want to delete.
4. Choose **Actions**, and then choose **Delete**.
5. Enter the outcome name, and then choose **Delete outcome**.

Delete a label

You cannot delete labels that are included in an event type in Amazon Fraud Detector.

When you delete a label, Amazon Fraud Detector permanently deletes that label and the data is no longer stored in Amazon Fraud Detector.

To delete a label

1. Sign in to the AWS Management Console and open the Amazon Fraud Detector console at https://console.aws.amazon.com/frauddetector.
2. In the left navigation pane of the Amazon Fraud Detector console, choose **Resources**, then choose **Labels**.
3. Choose the label that you want to delete.
4. Choose **Actions**, and then choose **Delete**.
5. Enter the label name, and then choose **Delete label**.

Delete a variable

You can't delete variables that are included in an event type in Amazon Fraud Detector.

Amazon Fraud Detector automatically deletes model output variables and SageMaker model output variables when you delete the model. You can't delete these variables manually.

When you delete a variable, Amazon Fraud Detector permanently deletes that variable and the data is no longer stored in Amazon Fraud Detector.

To delete a variable

1. Sign in to the AWS Management Console and open the Amazon Fraud Detector console at https://console.aws.amazon.com/frauddetector.
2. In the left navigation pane of the Amazon Fraud Detector console, choose **Resources**, then choose **Variables**.
3. Choose the variable that you want to delete.
4. Choose **Actions**, and then choose **Delete**.
5. Enter the variable name, and then choose **Delete variable**.

Delete stored event data

You can delete individual stored events data one at a time, or you can delete all stored events data associated with an event type. If you choose to delete all of the events data, Amazon Fraud Detector creates a batch delete job. While this job is in progress, new events data of that event type can’t be stored. If you want to delete an event type, you must first delete all events that are stored to that event type. For instructions on how to delete stored events, see Delete an event or event type .

Delete individual stored event data

When you delete an event, Amazon Fraud Detector permanently deletes that event and the event data is no longer stored in Amazon Fraud Detector.

To delete individual event data in console,

1. Sign in to the AWS Management Console and open the Amazon Fraud Detector console at https://console.aws.amazon.com/frauddetector.
2. In the left navigation pane of the Amazon Fraud Detector console, choose **Search past predictions**
3. Choose the event that you want to delete.
4. Choose **Actions**, and then choose **Delete event**.

Delete imported event data

You can delete data that has already been imported at any time in the Amazon Fraud Detector console, using the API, or AWS SDK.

To delete imported data in console

1. Sign in to the AWS Management Console and open the Amazon Fraud Detector console at https://console.aws.amazon.com/frauddetector.
2. In the left navigation pane of the Amazon Fraud Detector console, choose **Events**
3. Choose your event type.
4. Select **Stored events** tab.
5. In the **Import events data** pane, choose the job Id of an import job you want to delete.
6. In the event job page, click **Actions** and select **Delete import data**.
7. In the **Delete imported events** confirmation box, choose **Delete events import** to delete the imported event data.

Delete imported event data using the AWS SDK for Python (Boto3)

The following example shows a sample request for the `DeleteBatchImportJob` API. The delete batch import job must include the job ID of an imported job.

```
import boto3
                        fraudDetector = boto3.client('frauddetector')
fraudDetector.delete_batch_import_job (
    jobId = 'sample_batch'

)
```

Manage Amazon Fraud Detector resources using AWS CloudFormation

Amazon Fraud Detector is integrated with AWS CloudFormation, a service that helps you to model and set up your Amazon Fraud Detector resources so that you can spend less time creating and managing your resources and infrastructure. You create a template that describes all the Amazon Fraud Detector resources that you want (such as Detector, Variables, EntityType, EventType, Outcome, and Label), and AWS CloudFormation provisions and configures those resources for you. You can reuse the template to provision and configure the resources consistently and repeatedly in multiple AWS accounts and Regions.

There is no additional charge for using AWS CloudFormation.

Creating Amazon Fraud Detector templates

To provision and configure resources for Amazon Fraud Detector 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*.

You can also create, update, and delete your Amazon Fraud Detector resources using AWS CloudFormation templates. For more information, including examples of JSON and YAML templates for your resources, see the Amazon Fraud Detector resource type reference in the *AWS CloudFormation User Guide*.

If you are already using CloudFormation, there is no need to manage additional IAM policies or CloudTrail logging.

Managing Amazon Fraud Detector stacks

You can create, update, and delete your Amazon Fraud Detector stacks through the CloudFormation console or through the AWS CLI.

To create a stack, you must have a template that describes what resources AWS CloudFormation will include in your stack. You can also bring Amazon Fraud Detector resources that you have already created into CloudFormation management by importing them into a new or existing stack.

For detailed instructions for managing your stacks, see the *AWS CloudFormation User Guide* to learn how to create, update, and delete stacks.

Organizing your Amazon Fraud Detector stacks

The way you organize your AWS CloudFormation stacks is entirely up to you. It is generally a best practice is to organize stacks by lifecycle and ownership. This means grouping resources by how frequently they change or by teams that are responsible for updating them.

You can choose to organize your stacks by creating a stack for each detector and its detection logic (for example, rules, variables, etc.). If you are using other services, you should consider whether you want to stack together Amazon Fraud Detector resources with resources from other services. For example, you could create a stack that includes Kinesis resources that help gather data and Amazon Fraud Detector resources that process the data. This can be an effective way to ensure that all of your fraud team's products are working together.

Understanding Amazon Fraud Detector CloudFormation parameters

In addition to the standard parameters that are available in all CloudFormation templates, Amazon Fraud Detector introduces two additional parameters that will help you manage deployment behavior. If you do not include one or both of these parameters, CloudFormation will use the default value shown below.

Parameter	Values	Default Value
DetectorVersionStatus	**ACTIVE:** Set the new/updated detector version to Active status **DRAFT:** Set the new/updated detector version to Draft status	DRAFT
Inline	**TRUE:** Allow CloudFormation to create/update/delete the resource when creating/updating/deleting the stack. **FALSE:** Allow CloudFormation to validate that the object exists but not make any changes to the object.	TRUE

Sample AWS CloudFormation template for Amazon Fraud Detector resources

The following is a sample AWS CloudFormation YAML template for managing a detector and associated detector versions.

```
# Simple Detector resource containing inline Rule, EventType, Variable, EntityType and
 Label resource definitions
Resources:
  TestDetectorLogicalId:
    Type: AWS::FraudDetector::Detector
    Properties:
      DetectorId: "sample_cfn_created_detector"
      DetectorVersionStatus: "DRAFT"
      Description: "A detector defined and created in a CloudFormation stack!"

      Rules:
        - RuleId: "over_threshold_investigate"
          Description: "Automatically sends transactions of $10000 or more to an
 investigation queue"
          DetectorId: "sample_cfn_created_detector"
          Expression: "$amount >= 10000"
```

```
        Language: "DETECTORPL"
        Outcomes:
          - Name: "investigate"
            Inline: true
      - RuleId: "under_threshold_approve"
        Description: "Automatically approves transactions of less than $10000"
        DetectorId: "sample_cfn_created_detector"
        Expression: "$amount <10000"
        Language: "DETECTORPL"
        Outcomes:
          - Name: "approve"
            Inline: true
    EventType:
      Inline: "true"
      Name: "online_transaction"
      EventVariables:
        - Name: "amount"
          DataSource: 'EVENT'
          DataType: 'FLOAT'
          DefaultValue: '0'
          VariableType: "PRICE"
          Inline: 'true'
      EntityTypes:
        - Name: "customer"
          Inline: 'true'
      Labels:
        - Name: "legitimate"
          Inline: 'true'
        - Name: "fraudulent"
          Inline: 'true'
```

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

Get fraud predictions

You can use Amazon Fraud Detector to get fraud predictions for single event in real time or get fraud predictions offline for a set of events. To generate fraud predictions for either a single event or a set of events, you will need to provide Amazon Fraud Detector with the following information:

- Fraud prediction logic
- Event metadata

Fraud detection logic

Fraud prediction logic uses one or more rules to evaluate data associated with an event and then provides result and a fraud prediction score. You create your fraud prediction logic using the following components:

- Event types - Defines structure of the event
- Models - Defines algorithm and data requirements for predicting fraud
- Variables - Represents a data element associated with the event
- Rules - Tells Amazon Fraud Detector how to interpret the variable values during fraud prediction
- Outcomes - Results generated from a fraud prediction
- Detector version - Contains fraud prediction logic for a particular event

For more information about the components used to create fraud detection logic, see Amazon Fraud Detector concepts. Before you start generating fraud predictions, make sure you have created and published the detector version that contains your fraud prediction logic. You can create and publish detector version using the Fraud Detector Console or API. For instructions on using the console, see Get started (console). For instructions on using the API, see Create a detector version.

Event metadata

Event metadata provides details of the event being evaluated. Each event you want to evaluate must include value for each variable in the event type associated with your detector version. In addition, your event metadata must include the following:

- EVENT_ID – An identifier for the event. For example, if your event is an online transaction the EVENT_ID might be the transaction reference number provided to your customer.

  Important notes about EVENT_ID
  - Must be unique for that event
  - Should represent information that is meaningful to your business
  - Must satisy the regular expression pattern : `^[0-9a-z_-]+$`.
  - Must be saved. EVENT_ID is the reference for the event and is used to perform operations on the event such as deleting the event.
  - Appending timestamp to the EVENT_ID is not recommended as it might cause issues when later you want to update the event, since you will need to provide the exact same EVENT_ID.
- ENTITY_TYPE – The entity that performs the event, such as a merchant or a customer.
- ENTITY_ID - An identifier for the entity performing the event. The ENTITY_ID must satisfy the following regular expression pattern: `^[0-9a-z_-]+$`. If the ENTITY_ID is not available at the time of evaluation, pass the string unknown.

- EVENT_TIMESTAMP - The timestamp when the event occurred. The timestamp must be in ISO 8601 standard in UTC.

Real time prediction

You can evaluate online activities for fraud in real time by calling `GetEventPrediction` API. You provide information about a single event in each request and synchronously receive a model score and an outcome based on the fraud prediction logic associated with the specified detector.

How real time fraud prediction works

The `GetEventPrediction` API uses a specified detector version to evaluate the event metadata provided for the event. During the evaluation, Amazon Fraud Detector first generates model scores for models that are added to the detector version, then passes the results to the rules for evaluation. The rules are executed as specified by the rule execution mode (see Create a detector version). As part of the response, Amazon Fraud Detector provides model scores as well as any outcomes associated to the matched rules.

Getting real time fraud prediction

To get real time fraud predictions, make sure you have created and published a detector that contains your fraud prediction model and rules, or simply a ruleset.

You can get fraud prediction for an event in real time by calling the GetEventPrediction API operation using the AWS Command Line Interface (AWS CLI) or one of the Amazon Fraud Detector SDKs.

To use the API, supply information of a single event with each request. As part of the request you must specify `detectorId` that Amazon Fraud Detector will use to evaluate the event. You can optionally specify a `detectorVersionId`. If a `detectorVersionId` is not specified, Amazon Fraud Detector will use the `ACTIVE` version of the detector.

You can optionally send data to invoke an SageMaker model by passing the data in the field `externalModelEndpointBlobs`.

Get a fraud prediction using the AWS SDK for Python (Boto3)

To generate a fraud prediction, call the `GetEventPrediction` API. The example below assumes you have completed Part B: Generate fraud predictions (p. 13). As part of the response, you will receive a model score as well as any matched rules and corresponding outcomes. You can find additional examples of `GetEventPrediction` requests on the aws-fraud-detector-samples GitHub repository.

```
import boto3
fraudDetector = boto3.client('frauddetector')

fraudDetector.get_event_prediction(
detectorId = 'sample_detector',
eventId = '802454d3-f7d8-482d-97e8-c4b6db9a0428',
eventTypeName = 'sample_registration',
eventTimestamp = '2020-07-13T23:18:21Z',
entities = [{'entityType':'sample_customer', 'entityId':'12345'}],
eventVariables = {
    'email_address' : 'johndoe@exampledomain.com',
    'ip_address' : '1.2.3.4'
}
```

```
)
```

Batch predictions

You can use a *batch predictions* job in Amazon Fraud Detector to get predictions for a set of events that do not require real-time scoring. For example, you could create a batch predictions job to perform an offline proof-of-concept, or to retrospectively evaluate the risk of events on an hourly, daily, or weekly basis.

You can create a batch prediction job using the Amazon Fraud Detector console, or by calling the CreateBatchPredictionJobAPI operation using the AWS Command Line Interface (AWS CLI) or one of the Amazon Fraud Detector SDKs.

Topics

- How batch predictions work (p. 88)
- Input and output files (p. 88)
- Getting batch predictions (p. 89)
- Guidance on IAM roles (p. 89)
- Get batch fraud predictions using the AWS SDK for Python (Boto3) (p. 90)

How batch predictions work

The `CreateBatchPredictionJob` API operation uses a specified detector version to make predictions based on data provided in an input CSV file that is located in an Amazon S3 bucket. The API then returns the resulting CSV file to an S3 bucket.

Batch prediction jobs calculate model scores and prediction outcomes in the same way as the `GetEventPrediction` operation. Similar to `GetEventPrediction`, to create a batch predictions job, you first create an event type, optionally train a model, and then create a detector version that evaluates the events in your batch job.

The pricing for event risk scores evaluated by batch prediction jobs is the same as the pricing for scores created by the `GetEventPrediction` API. For details, see Amazon Fraud Detector pricing.

You can only run one batch prediction job at a time.

Input and output files

The input CSV file should contain headers that match the event type that is associated with the selected detector version. The maximum size of the input data file is 50 megabytes (MB), or about 50,000 events. The number of events will vary by your event size.

Amazon Fraud Detector creates the output file in the same bucket as the input file, unless you specify a separate location for the output data. The output file contains the original data from the input file and the following appended columns:

- `MODEL_SCORES` — Details the model scores for the event from each model associated with the selected detector version.
- `OUTCOMES` — Details the event outcomes as evaluated by the selected detector version and its rules.
- `STATUS` — Indicates whether the event was evaluated successfully. If the event was not evaluated successfully, this column shows a reason code for the failure.
- `RULE_RESULTS` — A list of all the rules that matched, based on the rule execution mode.

Getting batch predictions

The following steps assume that you have already created an event type, trained a model using that event type (optional), and created a detector version for that event type.

To get a batch prediction

1. Sign in to the AWS Management Console and open the Amazon Fraud Detector console at https://console.aws.amazon.com/frauddetector.
2. In the left navigation pane of the Amazon Fraud Detector console, choose **Batch Predictions**, and then choose **New batch prediction**.
3. In **Job name**, specify a name for your batch prediction job. If you don't specify a name, Amazon Fraud Detector randomly generates a job name.
4. In **Detector**, choose the detector for this batch prediction.
5. In **Detector version**, choose the detector version for this batch prediction. You can choose a detector version in any status. If your detector has a detector version in `Active` status, that version is automatically selected, but you can also change this selection if needed.
6. In **IAM role**, choose or create a role that has read and write access to your input and output Amazon S3 buckets. See Guidance on IAM roles (p. 89) for more information.

   To get batch predictions, the IAM user role that calls the `CreateBatchPredictionJob` operation must have read permissions to your input S3 bucket and write permissions to your output S3 bucket. For more information about bucket permissions, see User policy examples in the *Amazon S3 User Guide*.
7. In **Input data location**, specify the Amazon S3 location of your input data. If you want the output file in a different S3 bucket, select **Separate data location for output** and provide the Amazon S3 location for your output data.
8. (Optional) Create tags for your batch prediction job.
9. Choose **Start**.

   Amazon Fraud Detector creates the batch prediction job, and the job's status is `In progress`. Batch prediction job processing times vary depending on the number of events and your detector version configuration.

To stop a batch prediction job that is in progress, go to the batch prediction job detail page, choose **Actions**, and then choose **Stop batch prediction**. If you stop a batch prediction job, you won't receive any results for the job.

When the batch prediction job's status changes to `Complete`, you can retrieve the job's output from the designated output Amazon S3 bucket. The output file's name is in the format `batch prediction job name_file creation timestamp_output.csv`. For example, the output file from a job named `mybatchjob` is `mybatchjob_ 1611170650_output.csv`.

To search for specific events evaluated by a batch prediction job, in the left navigation pane of the Amazon Fraud Detector console, choose **Search past predictions**.

To delete a batch prediction job that has completed, go to the batch prediction job detail page, choose **Actions** and then choose **Delete batch prediction**.

Guidance on IAM roles

To get batch predictions, the IAM user role that calls the CreateBatchPredictionJob operation must have read permissions to your input S3 bucket and write permissions to your output S3 bucket. For more information about bucket permissions, see User policy examples in the Amazon S3 User Guide. On the Amazon Fraud Detector console, you have three options for selecting an IAM role for Batch Predictions:

1. Create a role when creating a new Batch Prediction job.
2. Select an existing IAM role that you have previously created in Amazon Fraud Detector console. Make sure to add the `S3:PutObject` permission to the role before you do this step.
3. Enter a custom ARN for a previously created IAM role.

If you receive an error related to your IAM role, verify the following:

1. Your S3 input and output buckets are in the same region as your detector.
2. The IAM role you are using has the `s3:GetObject` permission for your input S3 bucket and the `s3:PutObject` permission for your output S3 bucket.
3. The IAM role you are using has a trust policy for service principal `frauddetector.amazonaws.com`.

Get batch fraud predictions using the AWS SDK for Python (Boto3)

The following example shows a sample request for the CreateBatchPredictionJob API. A batch prediction job must include the following existing resources: detector, detector version, and event type name. The following example assumes you have created an event type `sample_registration`, a detector `sample_detector`, and a detector version `1`.

```
import boto3
fraudDetector = boto3.client('frauddetector')

fraudDetector.create_batch_prediction_job (
   jobId = 'sample_batch',
   inputPath = 's3://bucket_name/input_file_name.csv',
   outputPath = 's3://bucket_name/',
   eventTypeName = 'sample_registration',
   detectorName = 'sample_detector',
   detectorVersion = '1',
   iamRoleArn = 'arn:aws:iam::**:role/service-role/AmazonFraudDetector-DataAccessRole-**'
)
```

Prediction explanations

Prediction explanations provide insight into how each event variable impacted your model's fraud prediction score, and are automatically generated as part of the fraud prediction. Each fraud prediction comes with a risk score between 1 and 1000. Prediction explanations give you details of the influence of each event variable on the risk scores in terms of magnitude (0-5, 5 being highest) and direction (drove score higher or lower). You can also use prediction explanations for the following tasks:

- To identify top risk indicators during manual inverstigations when an event is flagged for review.
- To narrow down root causes that lead to false positive predictions (for example, high risk scores for legitimate events).
- To analyze fraud patterns across event data and detect bias, if any, in your dataset.

Important
Prediction explanations are automatically generated and available only for models trained on or after *June 30, 2021*. To receive prediction explanations for models trained before *June 30, 2021*, retrain those models.

Prediction explanations provide the following set of values for each event variable that was used to train the model.

Relative impact

Provides a visual reference of the variable's impact in terms of magnitude on the fraud prediction scores. The relative impact values consist of a star rating (0-5, 5 being the highest) and direction (increased/decreased) impact of the fraud risk.

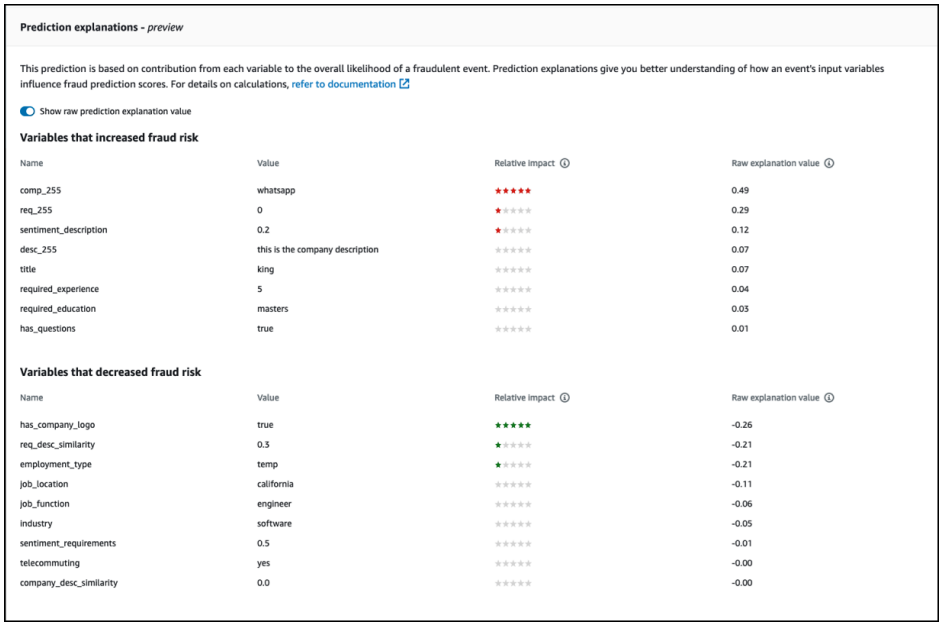
- Variables that increased fraud risk are indicated by red colored stars. The higher the number of red colored stars, the more the variable drove up the fraud score and increased likelihood of fraud.
- Variables that decreased fraud risk are indicated by green colored stars. The higher the number of green colored starts, the more the variable drove down the fraud risk score and decreased likelihood of fraud.
- Zero stars for all variables indicate that none of the variables on their own significantly changed the fraud risk.

Raw explanation value

Provides raw, uninterpreted value represented as log-odds of the fraud. These values are usually between -10 to +10, but range from - infinity to + infinity.

- A positive value indicates that the variable drove the risk score up.
- A negative value indicates that the variable drove the risk score down.

In the Amazon Fraud Detector console, the prediction explanation values are displayed as follows. The colored star ratings and the corresponding raw numerical values make it easy to see the relative influence between variables.



Viewing prediction explanations

After you generate fraud predictions, you can view prediction explanations in the Amazon Fraud Detector console. To view the prediction explanations using APIs from the AWS SDK, you must first call

the `ListEventPrediction` API to obtain the prediction timestamp for the event, and then call the `GetEventPredictionMetadata` API to get the prediction explanations.

View prediction explanations using Amazon Fraud Detector console

To view the prediction explanations using the console,

1. Open the AWS Console and sign in to your account. Navigate to Amazon Fraud Detector.
2. In the left navigation pane, choose **Search past predictions**.
3. Use the **Property**, **Operator**, and **Value** filters to select the prediction you want review.
4. In the top **Filter** pane, make sure to select the time period for when the prediction you want to review was generated.
5. The **Results** pane displays a list of all the predictions generated during the specified time period. Click the Event ID of the prediction to view the prediction explanations.
6. Scroll down to the **Prediction explanations** pane.
7. Set the **Show raw prediction explanation value** button **on** to view raw prediction explanation value of all the variables.

View prediction explanations using the AWS SDK for Python (Boto3)

The following examples show sample requests for viewing prediction explanations using `ListEventPredictions` and `GetEventPredictionMetadata` APIs from the AWS SDK.

Example 1: Get a list of most recent predictions using `ListEventPredictions` API

```
import boto3
fraudDetector = boto3.client('frauddetector')
fraudDetector.list_event_predictions(
  maxResults = 10,
  predictionTimeRange = {
     end_time: '2022-01-13T23:18:21Z',
     start_time: '2022-01-13T20:18:21Z'
    }
 )
```

Example 2; Get a list of past predictions for event type "registration" using `ListEventPredictions` API

```
import boto3
fraudDetector = boto3.client('frauddetector')
fraudDetector.list_event_predictions(
   eventType = {
      value = 'registration'
    }
   maxResults = 70,
   nextToken = "10",
   predictionTimeRange = {
     end_time: '2021-07-13T23:18:21Z',
     start_time: '2021-07-13T20:18:21Z'
    }
 )
```

Example 3: Get details of a past prediction for a specified event ID, event type, detector ID, and detector version ID that was generated in the specified time period using `GetEventPredictionMetadata` API.

The `predictionTimestamp` specified for this request is obtained by first calling the `ListEventPredictions` API.

```
import boto3
fraudDetector = boto3.client('frauddetector')
fraudDetector.get_event_prediction_metadata (
   detectorId = 'sample_detector',
   detectorVersionId = '1',
   eventId = '802454d3-f7d8-482d-97e8-c4b6db9a0428',
   eventTypeName = 'sample_registration',
   predictionTimestamp = '2021-07-13T21:18:21Z'
 )
```

Understanding how prediction explanations are calculated

Amazon Fraud Detector uses SHAP (SHapeley Additive exPlanations) to explain individual event predictions by computing the **raw explanation values** of each event variable used for model training. The raw explanation values are computed by the model as part of the classification algorithm when generating predictions. These raw explanation values represent the contribution of each input to the logarithm of the odds of fraud. The raw explanation values (from -infinity to +infinity) are converted to a **relative impact value** (-5 to +5) using a mapping. The relative impact value derived from raw explanation value represents the number of times increase in odds of fraud (positive) or legit (negative), making it easier to understand the prediction explanations.

Security in Amazon Fraud Detector

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

Topics

- Data Protection in Amazon Fraud Detector (p. 94)
- Identity and access management for Amazon Fraud Detector (p. 98)
- Logging and monitoring in Amazon Fraud Detector (p. 114)
- Compliance validation for Amazon Fraud Detector (p. 114)
- Resilience in Amazon Fraud Detector (p. 115)
- Infrastructure Security in Amazon Fraud Detector (p. 115)

Data Protection in Amazon Fraud Detector

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

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

- Use multi-factor authentication (MFA) with each account.
- Use SSL/TLS to communicate with AWS resources. We recommend TLS 1.2 or later.
- Set up API and user activity logging with AWS CloudTrail.
- Use AWS encryption solutions, along with all default security controls within AWS services.
- Use advanced managed security services such as Amazon Macie, which assists in discovering and securing personal data that is stored in Amazon S3.

- If you require FIPS 140-2 validated cryptographic modules when accessing AWS through a command line interface or an API, use a FIPS endpoint. For more information about the available FIPS endpoints, see Federal Information Processing Standard (FIPS) 140-2.

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

Encrypting data at rest

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

- An AWS owned KMS key. If you don't specify an encryption key your data is encrypted with this key by default.
- A customer managed KMS key. You can control access to your customer managed KMS key using key policies. For information on creating and managing customer managed KMS key, see Key management (p. 95).

Encrypting data in transit

Amazon Fraud Detector copies data out of your account and processes it in an internal AWS system. By default, Amazon Fraud Detector uses TLS 1.2 with AWS certificates to encrypt data in transit.

Key management

Amazon Fraud Detector encrypts your data using one of two types of keys:

- An AWS owned KMS key. This is the default.
- A customer managed KMS key.

Creating customer managed KMS key

You can create customer managed KMS key using either the AWS KMS console or the CreateKey API. When creating the key make sure you,

- Select a symmetric encryption customer managed KMS key, Amazon Fraud Detector does not support asymmetric KMS keys. For more information, see Asymmetric Keys in AWSKMS in the AWS Key Management Service Developer Guide.
- Provide the following key policy to grant permissions to Amazon Fraud Detector to use the key.

```
    {
"Effect": "Allow",
"Principal": {
    "Service": "frauddetector.amazonaws.com"
},
"Action": [
    "kms:Encrypt",
    "kms:Decrypt",
    "kms:ReEncrypt*",
```

```
        "kms:GenerateDataKey*",
        "kms:DescribeKey",
        "kms:CreateGrant",
        "kms:RetireGrant"
    ],
    "Resource": "*"
}
```

For information on key policies, see Using Key Policies in AWS KMS in the AWS Key Management Service Developer Guide.

Encrypting data using customer managed KMS key

Use Amazon Fraud Detector's PutKMSEncryptionKey API to encrypt your Amazon Fraud Detector data at rest using the customer managed KMS key. You can change the encryption configuration at any time using `PutKMSEncryptionKey` API.

Important notes about encrypted data

- Data generated after setting up the customer managed KMS key is encrypted. Data generated before setting up the customer managed KMS key will remain unencrypted.
- If customer managed KMS key is changed, the data that was encrypted using the previous encryption configuration will not be re-encrypted.

View data

When you use customer managed KMS key to encrypt your Amazon Fraud Detector data, the data encrypted using this method is not searchable using filters in the **Search Past Predictions** area of the Amazon Fraud Detector console. To ensure complete search results, use one or more of the following properties to filter results:

- Event ID
- Evaluation timestamp
- Detector status
- Detector version
- Model version
- Model type
- Rule evaluation status
- Rule execution mode
- Rule match status
- Rule version
- Variable data source

If customer managed KMS key was either deleted or is scheduled for deletion, your data might not be available. For more information, see Deleting KMS key.

Amazon Fraud Detector and interface VPC endpoints (AWS PrivateLink)

You can establish a private connection between your VPC and Amazon Fraud Detector by creating an *interface VPC endpoint*. Interface endpoints are powered by AWS PrivateLink, a technology that enables

you to privately access Amazon Fraud Detector APIs without an internet gateway, NAT device, VPN connection, or AWS Direct Connect connection. Instances in your VPC don't need public IP addresses to communicate with Amazon Fraud Detector APIs. Traffic between your VPC and Amazon Fraud Detector 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 Amazon Fraud Detector VPC endpoints

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

Amazon Fraud Detector supports making calls to all of its API actions from your VPC.

VPC endpoint policies are supported for Amazon Fraud Detector. By default, full access to Amazon Fraud Detector is allowed through the endpoint. For more information, see Controlling access to services with VPC endpoints in the *Amazon VPC User Guide*.

Creating an interface VPC endpoint for Amazon Fraud Detector

You can create a VPC endpoint for the Amazon Fraud Detector 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 Amazon Fraud Detector using the following service name:

- com.amazonaws.*region*.frauddetector

If you enable private DNS for the endpoint, you can make API requests to Amazon Fraud Detector using its default DNS name for the Region, for example, `frauddetector.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 Amazon Fraud Detector

You can create a policy for interface VPC endpoints for Amazon Fraud Detector to specify the following:

- 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*.

The following example VPC endpoint policy specifies that all users who have access to the VPC interface endpoint are allowed to access the Amazon Fraud Detector detector named `my_detector`.

```
{
  "Statement": [
      {
          "Action": "frauddetector:*Detector",
          "Effect": "Allow",
```

```
            "Resource": "arn:aws:frauddetector:us-east-1:123456789012:detector/my_detector",
            "Principal": "*"
        }
    ]
}
```

In this example, the following are denied:

- Other Amazon Fraud Detector API actions
- Invoking Amazon Fraud Detector `GetEventPrediction` API

> **Note**
> In this example, users can still take other Amazon Fraud Detector API actions from outside the VPC. For information about how to restrict API calls to those from within the VPC, see Amazon Fraud Detector identity-based policies (p. 103).

Opting out of using your data for service improvement

Historical event data you provide to train models and generate predictions is used solely to provide and maintain your service. This data might also be used to improve the quality of Amazon Fraud Detector. Your trust, privacy, and the security of your content are our highest priority and ensure that our use complies with our commitments to you. See Data Privacy FAQ for more information

You can choose to opt out of having your event data used to develop or improve the quality of Amazon Fraud Detector by visiting the AI services opt-out policies page in the *AWS Organizations User Guide* and following the process explained there.

> **Note**
> Your AWS accounts will need to be centrally managed by AWS Organizations for you to be able to use the opt-out policy. If you have not already created an organization for your AWS accounts, visit Creating and managing an organization page and follow the process explained there.

Identity and access management for Amazon Fraud Detector

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

Topics

- Audience (p. 99)
- Authenticating with identities (p. 99)
- Managing access using policies (p. 101)
- How Amazon Fraud Detector works with IAM (p. 102)
- Amazon Fraud Detector identity-based policy examples (p. 105)
- Confused deputy prevention (p. 110)
- Troubleshooting Amazon Fraud Detector identity and access (p. 111)

Audience

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

Service user – If you use the Amazon Fraud Detector service to do your job, then your administrator provides you with the credentials and permissions that you need. As you use more Amazon Fraud Detector 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 Fraud Detector, see Troubleshooting Amazon Fraud Detector identity and access (p. 111).

Service administrator – If you're in charge of Amazon Fraud Detector resources at your company, you probably have full access to Amazon Fraud Detector. It's your job to determine which Amazon Fraud Detector features and resources your employees should access. You must then submit requests to your IAM administrator to change the permissions of your service users. Review the information on this page to understand the basic concepts of IAM. To learn more about how your company can use IAM with Amazon Fraud Detector, see How Amazon Fraud Detector works with IAM (p. 102).

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 Fraud Detector. To view example Amazon Fraud Detector identity-based policies that you can use in IAM, see Amazon Fraud Detector identity-based policy examples (p. 105).

Authenticating with identities

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

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

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

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

AWS account root user

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

IAM Users and groups

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

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

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

IAM roles

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

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

- **Temporary IAM user permissions** – An IAM user can assume an IAM role to temporarily take on different permissions for a specific task.
- **Federated user access** – Instead of creating an IAM user, you can use existing identities from AWS Directory Service, your enterprise user directory, or a web identity provider. These are known as *federated users*. AWS assigns a role to a federated user when access is requested through an identity provider. For more information about federated users, see Federated users and roles in the *IAM User Guide*.
- **Cross-account access** – You can use an IAM role to allow someone (a trusted principal) in a different account to access resources in your account. Roles are the primary way to grant cross-account access. However, with some AWS services, you can attach a policy directly to a resource (instead of using a role as a proxy). To learn the difference between roles and resource-based policies for cross-account access, see How IAM roles differ from resource-based policies in the *IAM User Guide*.
- **Cross-service access** – Some AWS services use features in other AWS services. For example, when you make a call in a service, it's common for that service to run applications in Amazon EC2 or store objects in Amazon S3. A service might do this using the calling principal's permissions, using a service role, or using a service-linked role.
  - **Principal permissions** – When you use an IAM user or role to perform actions in AWS, you are considered a principal. Policies grant permissions to a principal. When you use some services, you might perform an action that then triggers another action in a different service. In this case, you must have permissions to perform both actions. To see whether an action requires additional dependent actions in a policy, see Actions, Resources, and Condition Keys for Amazon Fraud Detector in the *Service Authorization Reference*.
  - **Service role** – A service role is an IAM role that a service assumes to perform actions on your behalf. An IAM administrator can create, modify, and delete a service role from within IAM. For more information, see Creating a role to delegate permissions to an AWS service in the *IAM User Guide*.
  - **Service-linked role** – A service-linked role is a type of service role that is linked to an AWS service. The service can assume the role to perform an action on your behalf. Service-linked roles appear

in your IAM account and are owned by the service. An IAM administrator can view, but not edit the permissions for service-linked roles.

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

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

Managing access using policies

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

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

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

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

Identity-based policies

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

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

Resource-based policies

Resource-based policies are JSON policy documents that you attach to a resource. Examples of resource-based policies are IAM *role trust policies* and Amazon S3 *bucket policies*. In services that support resource-based policies, service administrators can use them to control access to a specific resource. For the resource where the policy is attached, the policy defines what actions a specified principal can perform

on that resource and under what conditions. You must specify a principal in a resource-based policy. Principals can include accounts, users, roles, federated users, or AWS services.

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

Access Control Lists (ACLs)

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

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

Other policy types

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

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

Multiple policy types

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

How Amazon Fraud Detector works with IAM

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

Topics

- Amazon Fraud Detector identity-based policies (p. 103)

- Amazon Fraud Detector resource-based policies (p. 105)
- Authorization Based on Amazon Fraud Detector Tags (p. 105)
- Amazon Fraud Detector IAM roles (p. 105)

Amazon Fraud Detector identity-based policies

With IAM identity-based policies, you can specify allowed or denied actions and resources as well as the conditions under which actions are allowed or denied. Amazon Fraud Detector supports specific actions, resources, and condition keys. To learn about all of the elements that you use in a JSON policy, see IAM JSON Policy Elements Reference in the *IAM User Guide*.

To get started with Amazon Fraud Detector, we recommend creating an IAM user with access restricted to Amazon Fraud Detector operations and required permissions. You can add other permissions as needed. The following policies provide the required permission to use Amazon Fraud Detector: `AmazonFraudDetectorFullAccessPolicy` and `AmazonS3FullAccess`. For more information on setting up Amazon Fraud Detector using these policies see Set up for Amazon Fraud Detector (p. 7).

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 Amazon Fraud Detector use the following prefix before the action: `frauddetector:`. For example, to create a rule with the Amazon Fraud Detector `CreateRule` API operation, you include the `frauddetector:CreateRule` action in the policy. Policy statements must include either an `Action` or `NotAction` element. Amazon Fraud Detector 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": [
      "frauddetector:action1",
      "frauddetector: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": "frauddetector:Describe*"
```

To see a list of Amazon Fraud Detector actions, see Actions Defined by Amazon Fraud Detector in the *IAM User Guide*.

Resources

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

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

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": "*"
```

Resource Types Defined by Amazon Fraud Detector lists all Amazon Fraud Detector resource ARNs.

For example, to specify the `my_detector` detector in your statement, use the following ARN:

```
"Resource": "arn:aws:frauddetector:us-east-1:123456789012:detector/my_detector"
```

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

To specify all detectors that belong to a specific account, use the wildcard (*):

```
"Resource": "arn:aws:frauddetector:us-east-1:123456789012:detector/*"
```

Some Amazon Fraud Detector 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 Amazon Fraud Detector resource types and their ARNs, see Resources Defined by Amazon Fraud Detector in the *IAM User Guide*. To learn which actions you can specify the ARN of each resource, see Actions Defined by Amazon Fraud Detector.

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*.

Amazon Fraud Detector defines its own set of condition keys and also supports using some global condition keys. To see all AWS global condition keys, see AWS Global Condition Context Keys in the *IAM User Guide*.

To see a list of Amazon Fraud Detector condition keys, see Condition Keys for Amazon Fraud Detector in the *IAM User Guide*. To learn which actions and resources you can use a condition key, see Actions Defined by Amazon Fraud Detector.

Examples

To view examples of Amazon Fraud Detector identity-based policies, see Amazon Fraud Detector identity-based policy examples (p. 105).

Amazon Fraud Detector resource-based policies

Amazon Fraud Detector does not support resource-based policies.

Authorization Based on Amazon Fraud Detector Tags

You can attach tags to Amazon Fraud Detector resources or pass tags in a request to Amazon Fraud Detector. 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.

Amazon Fraud Detector IAM roles

An IAM role is an entity within your AWS account that has specific permissions.

Using temporary credentials with Amazon Fraud Detector

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

Amazon Fraud Detector supports using temporary credentials.

Service-linked roles

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

Amazon Fraud Detector does not support service-linked roles.

Service roles

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

Amazon Fraud Detector supports service roles.

Amazon Fraud Detector identity-based policy examples

By default, IAM users and roles don't have permission to create or modify Amazon Fraud Detector resources. They also can't perform tasks using the AWS Management Console, AWS CLI, or AWS API. An

IAM administrator must create IAM policies that grant users and roles permission to perform specific API operations on the specified resources they need. The administrator must then attach those policies to the IAM users or groups that require those permissions.

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

Topics

- Policy best practices (p. 106)
- AWS-managed (predefined) policy for Amazon Fraud Detector (p. 106)
- Allow users to view their own permissions (p. 107)
- Allowing full access to Amazon Fraud Detector resources (p. 107)
- Allowing read-only access to Amazon Fraud Detector resources (p. 108)
- Allowing access to a specific resource (p. 108)
- Limiting access based on tags (p. 109)

Policy best practices

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

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

AWS-managed (predefined) policy for Amazon Fraud Detector

AWS addresses many common use cases by providing standalone IAM policies that are created and administered by AWS. These AWS managed policies grant necessary permissions for common use cases so that you can avoid having to investigate which permissions are needed. For more information, see AWS Managed Policies in the IAM User Guide.

The following AWS managed policy, which you can attach to users in your account, is specific to Amazon Fraud Detector:

`AmazonFraudDetectorFullAccess`: Grants full access to Amazon Fraud Detector resources, actions and the supported operations including:

- List and describe all model endpoints in Amazon SageMaker

- List all IAM roles in the account
- List all Amazon S3 buckets
- Allow IAM Pass Role to pass a role to Amazon Fraud Detector

This policy does not provide unrestricted S3 access. If you need to upload model training datasets to S3, the `AmazonS3FullAccess` managed policy (or scoped-down custom Amazon S3 access policy) is also required.

You can review the policy's permissions by signing in to the IAM console and searching by the policy name. You can also create your own custom IAM policies to allow permissions for Amazon Fraud Detector actions and resources as you need them. You can attach these custom policies to the IAM users or groups that require them.

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": "*"
        }
    ]
}
```

Allowing full access to Amazon Fraud Detector resources

The following example gives an IAM user in your AWS account full access to all Amazon Fraud Detector resources and actions.

```
{
```

```
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "frauddetector:*"
            ],
            "Resource": "*"
        }
    ]
}
```

Allowing read-only access to Amazon Fraud Detector resources

In this example, you grant an IAM user in your AWS account read-only access to your Amazon Fraud Detector resources.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
          "frauddetector:GetEventTypes",
                "frauddetector:BatchGetVariable",
                "frauddetector:DescribeDetector",
                "frauddetector:GetModelVersion",
                "frauddetector:GetEventPrediction",
                "frauddetector:GetExternalModels",
                "frauddetector:GetLabels",
                "frauddetector:GetVariables",
                "frauddetector:GetDetectors",
                "frauddetector:GetRules",
                "frauddetector:ListTagsForResource",
                "frauddetector:GetKMSEncryptionKey",
                "frauddetector:DescribeModelVersions",
                "frauddetector:GetDetectorVersion",
                "frauddetector:GetPrediction",
                "frauddetector:GetOutcomes",
                "frauddetector:GetEntityTypes",
                "frauddetector:GetModels"
            ],
            "Resource": "*"
        }
    ]
}
```

Allowing access to a specific resource

In this example of a resource-level policy, you grant an IAM user in your AWS account access to all actions and resources except for one particular Detector resource.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "frauddetector:*"
            ],
            "Resource": "*"
```

```
        },
        {
            "Effect": "Deny",
            "Action": [
                "frauddetector:*Detector"
            ],
            "Resource": "arn:${Partition}:frauddetector:${Region}:${Account}:detector/
${detector-name}"
        }
    ]
}
```

Limiting access based on tags

This example policy demonstrates how to limit access to Amazon Fraud Detector based on resource tags. This example assumes that:

- In your AWS account you have defined two different IAM groups, named Team1 and Team2
- You have created four detectors
- You want to allow members of Team1 to make API calls on 2 detectors
- You want to allow members of Team2 to make API calls on the other 2 detectors

To control access to API calls (example)

1. Add a tag with the key `Project` and value `A` to the detectors used by Team1.
2. Add a tag with the key `Project` and value `B` to the detectors used by Team2.
3. Create an IAM policy with a `ResourceTag` condition that denies access to detectors that have tags with key `Project` and value `B`, and attach that policy to Team1.
4. Create an IAM policy with a `ResourceTag` condition that denies access to detectors that have tags with key `Project` and value `A`, and attach that policy to Team2.

The following is an example of a policy that denies specific actions on any Amazon Fraud Detector resource that has a tag with a key of `Project` and a value of `B`:

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": "frauddetector:*",
      "Resource": "*"
    },
    {
      "Effect": "Deny",

      "Action": [

        "frauddetector:CreateModel",
        "frauddetector:CancelBatchPredictionJob",
        "frauddetector:CreateBatchPredictionJob",
        "frauddetector:DeleteBatchPredictionJob",
        "frauddetector:DeleteDetector"
      ],

      "Resource": "*",
      "Condition": {
        "StringEquals": {
```

```
          "aws:ResourceTag/Project": "B"
        }
      }
    }
  ]
}
```

Confused deputy prevention

The confused deputy problem occurs when an entity that doesn't have permission to perform an action can coerce a more-privileged entity to perform the action. AWS provides tools that help you protect your account if you provide third-parties (called *cross-account*) or other AWS services (called *cross-service*) access to resources in your account.

Cross-service confused deputy problem can occur when one service (the *calling service*) calls another service (the *called service*). The calling service can be manipulated to use its permissions to act on another customer's resources in a way it should not otherwise have permission to access. To prevent this, you can create policies that help you protect your data for all services with service principals that have been given access to your service's resources.

Amazon Fraud Detector supports using service roles in your permission policies to allow a service to access another service's resources on your behalf. A role requires two policies: a role trust policy that specifies the principal that is allowed to assume the role and a permissions policy that specifies what can be done with the role. When a service assumes a role on your behalf, the service principal must be allowed to perform the `sts:AssumeRole` action in the role trust policy. When a service calls `sts:AssumeRole`, AWS STS returns a set of temporary security credentials that the service principal uses to access the resources that are permitted by the role's permissions policy.

To prevent cross-service confused deputy problem, Amazon Fraud Detector recommends using the `aws:SourceArn` and `aws:SourceAccount` global condition context keys in your role trust policy to limit access to the role to only those requests that are generated by expected resources.

The `aws:SourceAccount` specifies the Account ID and the `aws:SourceArn` specifies the ARN of the resource associated with cross-service access. The `aws:SourceArn` must be specified using the ARN format. Ensure that both `aws:SourceAccount` and `aws:SourceArn` are using the same Account ID when used in the same policy statement.

The most effective way to protect against the confused deputy problem is to use the `aws:SourceArn` global condition context key with the full ARN of the resource. If you don't know the full ARN of the resource or if you are specifying multiple resources, use the `aws:SourceArn` global context condition key with a wildcard (*) for the unknown portions of the ARN. For example, `arn:aws:`*`servicename`*`:*:`*`123456789012`*`:*`. For information about Amazon Fraud Detector resources and actions you can use in your permission policies, see Actions, resources, and condition keys for Amazon Fraud Detector.

The following role trust policy example uses wildcard(*) in the `aws:SourceArn` condition key to allow Amazon Fraud Detector access to multiple resources associated with the Account Id.

```
{
        "Version": "2012-10-17",
        "Statement": [
             {
               "Effect": "Allow",
               "Principal": {
               "Service": [
                   "frauddetector.amazonaws.com"
                   ]
```

```
                },
                "Action": "sts:AssumeRole",
                "Condition": {
                "StringEquals": {
                    "aws:SourceAccount": "123456789012"
                },
                  "StringLike": {
                     "aws:SourceArn": "arn:aws:frauddetector:us-west-2:123456789012:*"
        }
      }
    }
  ]
}
```

The following role trust policy allows Amazon Fraud Detector access to only `external-model` resource. Notice the `aws:SourceArn` param in Condition block. The resource qualifier is built using the model endpoint that is provided for making the `PutExternalModel` API call.

```
   {
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "Service": [
          "frauddetector.amazonaws.com"
        ]
      },
      "Action": "sts:AssumeRole",
      "Condition": {
        "StringEquals": {
          "aws:SourceAccount": "123456789012"
        },
        "StringLike": {
          "aws:SourceArn": "arn:aws:frauddetector:us-west-2:123456789012:external-model/
MyExternalModeldoNotDelete-ReadOnly"
        }
      }
    }
  ]
}
```

Troubleshooting Amazon Fraud Detector identity and access

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

Topics

- I am not authorized to perform an action in Amazon Fraud Detector (p. 112)
- I am not authorized to perform iam:PassRole (p. 112)
- I want to view my access keys (p. 112)
- I'm an administrator and want to allow others to access Amazon Fraud Detector (p. 113)
- I want to allow people outside of my AWS account to access my Amazon Fraud Detector resources (p. 113)
- Amazon Fraud Detector could not assume the given role (p. 113)

I am not authorized to perform an action in Amazon Fraud Detector

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

The following example error occurs when the `mateojackson` IAM user tries to use the console to view details about a *`detector`* but does not have `frauddetector:`*`GetDetectors`* permissions.

```
User: arn:aws:iam::123456789012:user/mateojackson is not authorized to perform:
 frauddetector:GetDetectors on resource: my-example-detector
```

In this case, Mateo asks his administrator to update his policies to allow him to access the *`my-example-detector`* resource using the `frauddetector:`*`GetDetectors`* action.

I am not authorized to perform iam:PassRole

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

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 Fraud Detector. However, the action requires the service to have permissions granted by a service role. Mary does not have permissions to pass the role to the service.

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

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

I want to view my access keys

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

Access keys consist of two parts: an access key ID (for example, `AKIAIOSFODNN7EXAMPLE`) and a secret access key (for example, `wJalrXUtnFEMI/K7MDENG/bPxRfiCYEXAMPLEKEY`). Like a user name and password, you must use both the access key ID and secret access key together to authenticate your requests. Manage your access keys as securely as you do your user name and password.

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

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

I'm an administrator and want to allow others to access Amazon Fraud Detector

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

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

I want to allow people outside of my AWS account to access my Amazon Fraud Detector 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 Fraud Detector supports these features, see How Amazon Fraud Detector works with IAM (p. 102).
- To learn how to provide access to your resources across AWS accounts that you own, see Providing access to an IAM user in another AWS account that you own in the *IAM User Guide*.
- To learn how to provide access to your resources to third-party AWS accounts, see Providing access to AWS accounts owned by third parties in the *IAM User Guide*.
- To learn how to provide access through identity federation, see Providing access to externally authenticated users (identity federation) in the *IAM User Guide*.
- To learn the difference between using roles and resource-based policies for cross-account access, see How IAM roles differ from resource-based policies in the *IAM User Guide*.

Amazon Fraud Detector could not assume the given role

If you receive an error that Amazon Fraud Detector could not assume the given role, then you must update the trust relationship for the specified role. By specifying Amazon Fraud Detector as a trusted entity, the service can assume the role. When you use Amazon Fraud Detector to create a role, this trust relationship is automatically set. You only need to establish this trust relationship for IAM roles that are not created by Amazon Fraud Detector.

To establish a trust relationship for an existing role to Amazon Fraud Detector

1. Open the IAM console at https://console.aws.amazon.com/iam/
2. In the navigation pane choose **Roles**.
3. Choose the name of the role that you want to modify, and choose the **Trust relationships** tab.
4. Choose **Edit trust relationship**.
5. Under **Policy Document**, paste the following, and then choose **Update Trust Policy**.

```
{
    "Version": "2012-10-17",
    "Statement": [ {
        "Effect": "Allow",
        "Principal": {
            "Service": "frauddetector.amazonaws.com"
```

```
            },
            "Action": "sts:AssumeRole"
        } ]
    }
```

Logging and monitoring in Amazon Fraud Detector

AWS provides the following monitoring tools to watch Amazon Fraud Detector, 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. For more information on CloudWatch, see the Amazon CloudWatch 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. For more information an CloudTrail, see the AWS CloudTrail User Guide.

For more information on monitoring Amazon Fraud Detector, see Monitor Amazon Fraud Detector (p. 116).

Compliance validation for Amazon Fraud Detector

Third-party auditors assess the security and compliance of AWS services as part of multiple AWS compliance programs, such as SOC, PCI, FedRAMP, and HIPAA.

To learn whether Amazon Fraud Detector or other AWS services are 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 AWS services is determined by the sensitivity of your data, your company's compliance objectives, and applicable laws and regulations. AWS provides the following resources to help with compliance:

- Security and Compliance Quick Start Guides – These deployment guides discuss architectural considerations and provide steps for deploying baseline environments on AWS that are security and compliance focused.
- Architecting for HIPAA Security and Compliance Whitepaper – This whitepaper describes how companies can use AWS to create HIPAA-eligible applications.

  Note
  Not all AWS services are HIPAA eligible. For more information, see the HIPAA Eligible Services Reference.
- 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.
- AWS Security Hub – This AWS service provides a comprehensive view of your security state within AWS that helps you check your compliance with security industry standards and best practices.

- AWS Audit Manager – This AWS service helps you continuously audit your AWS usage to simplify how you manage risk and compliance with regulations and industry standards.

Resilience in Amazon Fraud Detector

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 Fraud Detector

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

You use AWS published API calls to access Amazon Fraud Detector 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.

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

Monitor Amazon Fraud Detector

Monitoring is an important part of maintaining the reliability, availability, and performance of Amazon Fraud Detector and your other AWS solutions. AWS provides the following monitoring tools to watch Amazon Fraud Detector, 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 more information, see the Amazon CloudWatch User Guide.
- *AWS CloudTrail* captures API calls and related events made by or on behalf of your AWS account and delivers the log files to an Amazon S3 bucket that you specify. You can identify which users and accounts called AWS, the source IP address from which the calls were made, and when the calls occurred. For more information, see the AWS CloudTrail User Guide.

Topics

- Monitoring Amazon Fraud Detector with Amazon CloudWatch (p. 116)
- Logging Amazon Fraud Detector API Calls with AWS CloudTrail (p. 120)

Monitoring Amazon Fraud Detector with Amazon CloudWatch

You can monitor Amazon Fraud Detector 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.

Topics

- Using CloudWatch Metrics for Amazon Fraud Detector (p. 116)
- Amazon Fraud Detector Metrics (p. 118)

Using CloudWatch Metrics for Amazon Fraud Detector

To use metrics, you must specify the following information:

- The metric dimension or no dimension. A *dimension* is a name-value pair that helps you to uniquely identify a metric.
- The metric name, such as `GetEventPrediction`.

You can get monitoring data for Amazon Fraud Detector by using the AWS Management Console, the AWS CLI, or the CloudWatch API. You can also use the CloudWatch API through one of the Amazon AWS Software Development Kits (SDKs) or the CloudWatch API tools. The console displays a series of graphs based on the raw data from the CloudWatch API. Depending on your needs, you might prefer to use either the graphs displayed in the console or retrieved from the API.

The following list shows some common uses for the metrics. These are suggestions to get you started, not a comprehensive list.

How Do I?	Relevant Metrics
How do I track the number of predictions that have been performed?	Monitor the `GetEventPrediction` metric.
How can I monitor `GetEventPrediction` errors?	Use the `GetEventPrediction5xxError` and the `GetEventPrediction4xxError` metrics.
How can I monitor the latency of `GetEventPrediction` calls?	Use the `GetEventPredictionLatency` metric.

You must have the appropriate CloudWatch permissions to monitor Amazon Fraud Detector with CloudWatch. For more information, see Authentication and Access Control for Amazon CloudWatch.

Access Amazon Fraud Detector Metrics

The following steps show how to access Amazon Fraud Detector metrics using the CloudWatch console.

To view metrics (console)

1. Open the CloudWatch console at https://console.aws.amazon.com/cloudwatch/.
2. Choose **Metrics**, choose the **All Metrics** tab, and then choose **Fraud Detector**.
3. Choose the metric dimension.
4. Choose the desired metric from the list, and choose a time period for the graph.

Create an Alarm

You can create a CloudWatch alarm that sends an Amazon Simple Notification Service (Amazon SNS) message when the alarm changes state. An alarm watches a single metric over a time period that you specify. It performs one or more actions based on the value of the metric relative to a given threshold over a number of time periods. The action is a notification sent to an Amazon SNS topic or an Auto Scaling policy.

Alarms invoke actions for sustained state changes only. CloudWatch alarms don't invoke actions simply because they are in a particular state. The state must have changed and have been maintained for a specified number of time periods.

To set an alarm (console)

1. Sign in to the AWS Management Console and open the CloudWatch console at https://console.aws.amazon.com/cloudwatch/.
2. In the navigation pane, choose **Alarms**, and choose **Create Alarm**. This opens the **Create Alarm Wizard**.
3. Choose **Select metric**.
4. In the **All metrics** tab, choose **Fraud Detector**.
5. Choose **By Detector ID**, and then choose the **GetEventPrediction** metric.
6. Choose the **Graphed metrics** tab.
7. For **Statistic**, choose **Sum**.
8. Choose **Select metric**.

9. For **Conditions**, choose **Static** for **Threshold type** and **Greater** for **Whenever...**, and then enter a maximum value of your choice. Choose **Next**.
10. To send alarms to an existing Amazon SNS topic, for **Send notification to:**, choose an existing SNS topic. To set the name and email addresses for a new email subscription list, choose **New list**. CloudWatch saves the list and displays it in the field so you can use it to set future alarms.

    Note
    If you use **New list** to create a new Amazon SNS topic, the email addresses must be verified before the intended recipients receive notifications. Amazon SNS sends email only when the alarm enters an alarm state. If this alarm state change happens before the email addresses are verified, intended recipients don't receive a notification.
11. Choose **Next**. Add a name and optional description for your alarm. Choose **Next**.
12. Choose **Create Alarm**.

Amazon Fraud Detector Metrics

Amazon Fraud Detector sends the following metrics to CloudWatch. All metrics support these statistics: `Average`, `Minimum`, `Maximum`, `Sum`.

Metric	Description
`GetEventPrediction`	The number of GetEventPrediction API requests. Valid Dimensions: `DetectorID`
`GetEventPredictionLatency`	The interval of time taken to respond to a client request from the GetEventPrediction request. Valid Dimensions: `DetectorID` Unit: Milliseconds
`GetEventPrediction4XXError`	The number of GetEventPrediction requests where Amazon Fraud Detector returned a 4xx HTTP response code. For each 4xx response, 1 is sent. Valid Dimensions: `DetectorID`
`GetEventPrediction5XXError`	The number of GetEventPrediction requests where Amazon Fraud Detector returned a 5xx HTTP response code. For each 5xx response, 1 is sent. Valid Dimensions: `DetectorID`
`Prediction`	The number of predictions. 1 is sent if successful. Valid Dimensions: `DetectorID`, `DetectorVersionID`
`PredictionLatency`	The interval of time taken for a prediction operation. Valid Dimensions: `DetectorID`, `DetectorVersionID` Unit: Milliseconds
`PredictionError`	The number of predictions where Amazon Fraud Detector encountered an error. 1 is sent if an error is encountered.

Metric	Description
	Valid Dimensions: `DetectorID`, `DetectorVersionID`
`VariableUsed`	The number of GetEventPrediction requests where the variable was used as part of the evaluation. Valid Dimensions: `DetectorID`, `DetectorVersionID`, `VariableName`
`VariableDefaultReturned`	The number of GetEventPrediction requests where the variable was not present as part of the Event Attributes and therefore the default value for the variable was used during evaluation. Valid Dimensions: `DetectorID`, `DetectorVersionID`, `VariableName`
`RuleNotEvaluated`	The number of GetEventPrediction requests where the rule was not evaluated because a prior rule matched. Valid Dimensions: `DetectorID`, `DetectorVersionID`, `RuleID`
`RuleEvaluateTrue`	The number of GetEventPrediction requests where the rule triggered as True and the rule outcome was returned. Valid Dimensions: `DetectorID`, `DetectorVersionID`, `RuleID`
`RuleEvaluateFalse`	The number of GetEventPrediction requests where the rule evaluated to False. Valid Dimensions: `DetectorID`, `DetectorVersionID`, `RuleID`
`RuleEvaluateError`	The number of GetEventPrediction requests where the rule evaluates in error Valid Dimensions: `DetectorID`, `DetectorVersionID`, `RuleID`
`OutcomeReturned`	The number of GetEventPrediction calls where the specified outcome was returned. Valid Dimensions: `DetectorID`, `DetectorVersionID`, `OutcomeName`
`ModelInvocation (Amazon SageMaker model endpoint)`	The number of GetEventPrediction requests where the SageMaker model endpoint was invoked as part of the evaluation. Valid Dimensions: `DetectorID`, `DetectorVersionID`, `ModelEndpoint`

Metric	Description
`ModelInvocationError (Amazon SageMaker model endpoint)`	The number of GetEventPrediction requests where the invoked SageMaker model endpoint returned an error during evaluation. Valid Dimensions: `DetectorID`, `DetectorVersionID`, `ModelEndpoint`
`ModelInvocationLatency (Amazon SageMaker model endpoint)`	The interval of time taken by the Imported Model to respond as viewed from Amazon Fraud Detector. This interval includes only the model invocation. Valid Dimensions: `DetectorID`, `DetectorVersionID`, `ModelEndpoint` Unit: Milliseconds
`ModelInvocation`	The number of GetEventPrediction requests where the model was invoked as part of the evaluation. Valid Dimensions: `DetectorID`, `DetectorVersionID`, `ModelType`, `ModelID`
`ModelInvocationError`	The number of GetEventPrediction requests where the Amazon Fraud Detector model returned an error during evaluation. Valid Dimensions: `DetectorID`, `DetectorVersionID`, `ModelType`, `ModelID`
`ModelInvocationLatency`	The interval of time taken by the Amazon Fraud Detector Model to respond as viewed from Amazon Fraud Detector. This interval includes only the model invocation. Valid Dimensions: `DetectorID`, `DetectorVersionID`, `ModelType`, `ModelID` Unit: Milliseconds

Logging Amazon Fraud Detector API Calls with AWS CloudTrail

Amazon Fraud Detector is integrated with AWS CloudTrail, a service that provides a record of actions taken by a user, role, or an AWS service in Amazon Fraud Detector. CloudTrail captures all API calls for Amazon Fraud Detector as events, including calls from the Amazon Fraud Detector console and calls from code to the Amazon Fraud Detector APIs.

If you create a trail, you can enable continuous delivery of CloudTrail events to an Amazon S3 bucket, including events for Amazon Fraud Detector. If you don't configure a trail, you can still view the most recent events in the CloudTrail console in **Event history**. Using the information collected by CloudTrail, you can determine the request that was made to Amazon Fraud Detector, the IP address that the request was made from, who made the request, when it was made, and additional details.

To learn more about CloudTrail, see the AWS CloudTrail User Guide.

Amazon Fraud Detector Information in CloudTrail

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

For an ongoing record of events in your AWS account, including events for Amazon Fraud Detector, create a trail. A *trail* enables CloudTrail to deliver log files to an Amazon S3 bucket. By default, when you create a trail in the console, the trail applies to all AWS Regions. The trail logs events from all Regions in the AWS partition and delivers the log files to the Amazon S3 bucket that you specify. Additionally, you can configure other AWS services to further analyze and act upon the event data that's collected in CloudTrail logs. For more information, see the following:

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

Amazon Fraud Detector supports logging every action (API operation) as an event in CloudTrail log files. For more information, see Actions.

Every event or log entry contains information about who generated the request. The identity information helps you determine the following:

- Whether the request was made with root or AWS Identity and Access Management (IAM) user credentials.
- Whether the request was made with temporary security credentials for a role or federated user.
- Whether the request was made by another AWS service.

For more information, see the CloudTrail userIdentity Element.

Understanding Amazon Fraud Detector Log File Entries

A trail is a configuration that enables delivery of events as log files to an Amazon S3 bucket that you specify. CloudTrail log files contain one or more log entries. An event represents a single request from any source and includes information about the requested operation, the date and time of the operation, request parameters, and so on. CloudTrail log files aren't an ordered stack trace of the public API calls, so they don't appear in any specific order.

The following example shows a CloudTrail log entry that demonstrates the `GetDetectors` operation.

```
{
    "eventVersion": "1.05",
    "userIdentity": {
        "type": "IAMUser",
        "principalId": "principal-id",
        "arn": "arn:aws:iam::user-arn",
        "accountId": "account-id",
        "accessKeyId": "access-key",
        "userName": "user-name"
    },
```

```
    "eventTime": "2019-11-22T02:18:03Z",
    "eventSource": "frauddetector.amazonaws.com",
    "eventName": "GetDetectors",
    "awsRegion": "us-east-1",
    "sourceIPAddress": "source-ip-address",
    "userAgent": "aws-cli/1.11.16 Python/2.7.11 Darwin/15.6.0 botocore/1.4.73",
    "requestParameters": null,
    "responseElements": null,
    "requestID": "request-id",
    "eventID": "event-id",
    "eventType": "AwsApiCall",
    "recipientAccountId": "recipient-account-id"
}
```

Troubleshoot

The following sections help you troubleshoot issues that you might encounter when working with Amazon Fraud Detector

Troubleshoot training data issues

Use information in this section to help diagnose and resolve issues you might see in the **Model training diagnostic** pane in the Amazon Fraud Detector console when you train your model.

The issues displayed in the **Model training diagnostic** pane are categorized as follows. The requirement to address the issue is dependent on the category of the issue.

- **Error**- causes the model training to fail. These issues must be addressed for the model to train successfully.
- **Warning**- causes the model training to continue, however, some of the variables might be getting excluded in the training process. Check for the relevant guidance in this section to improve the quality of your dataset.
- **Information (Info)**- has no impact on model training and all the variables are used for training. We recommend that you check the relevant guidance in this section to further improve the quality of your dataset and model performance.

Topics

- Unstable fraud rate in the given dataset (p. 123)
- Insufficient data (p. 124)
- Missing or different EVENT_LABEL values (p. 125)
- Missing or incorrect EVENT_TIMESTAMP values (p. 126)
- Data not ingested (p. 127)
- Insufficient variables (p. 127)
- Missing or incorrect variable type (p. 128)
- Missing variable values (p. 128)
- Insufficient unique variable values (p. 128)
- Incorrect variable expression (p. 129)
- Insufficient unique entities (p. 130)

Unstable fraud rate in the given dataset

Issue type : Error

Description

Fraud rate in the given data is too unstable through time. Please make sure your fraud and legitimate events are sampled uniformly over time.

Cause

This error occurs if the fraud and legitimate events in your dataset are distributed unevenly and are taken from different time slots. Amazon Fraud Detector model training process samples and partitions your dataset based on EVENT_TIMESTAMP. For example, if your dataset consists of fraud events pulled from last 6 months, but only the last month of legitimate events are included, the dataset is considered unstable. An unstable dataset might lead to biases in model performance evaluation.

Solution

Make sure to provide the fraudulent and legitimate events data from same time slot and the fraud rate does not change dramatically over time.

Insufficient data

1. **Issue type : Error**

   Description

   Fewer than 400 rows are labeled as fraudulent events. Ensure that both fraudulent and legitimate events exceed the minimum count of 400 and re-train the model.

   Cause

   This error occurs if your dataset has fewer events labeled as fraudulent than required for model training. Amazon Fraud Detector requires at least 400 fraudulent events to train your model.

   Solution

   Make sure that your dataset includes a minimum of 400 fraudulent events. You can ensure this by covering a longer time period, if needed.
2. **Issue type : Error**

   Description

   Fewer than 400 rows are labeled as legitimate events. Ensure that both fraudulent and legitimate events exceed the minimum count of $threshold and re-train the model.

   Cause

   This error occurs if your dataset has fewer events labeled as legitimate than required for model training. Amazon Fraud Detector requires at least 400 legitimate events to train your model.

   Solution

   Make sure that your dataset includes a minimum of 400 legitimate events. You can ensure this by covering a longer time period, if needed.
3. **Issue type : Error**

   Description

   The number of unique entities associated with fraud is less than 100. Consider including more examples of fraudulent entities to improve performance.

   Cause

   This error occurs if your dataset has fewer entities with fraudulent events than required for model training. The Transaction Fraud Insights (TFI) model requires at least 100 entities with fraud events to ensure maximum coverage of the fraud space. The model may not generalize well if all fraud events are performed by a small group of entities.

Solution

Make sure that your dataset includes at least 100 entities with fraudulent events. You can ensure this be covering a longer time period, if needed.

4. **Issue type : Error**

   Description

   The number of unique entities associated with legitimate is less than 100. Consider including more examples of legitimate entities to improve performance.

   Cause

   This error occurs if your dataset has fewer entities with legitimate events than required for model training. The Transaction Fraud Insights (TFI) model requires at least 100 entities with legitimate events to ensure maximum coverage of the fraud space. The model may not generalize well if all legitimate events are performed by a small group of entities.

   Solution

   Make sure that your dataset includes at least 100 entities with legitimate events. You can ensure this be covering a longer time period, if needed.

5. **Issue type : Error**

   Description

   Less than 10,000 rows are in the dataset. Ensure there are more than 10,000 rows in the total dataset and both fraudulent and legitimate events exceed the minimum count of 400.

   Cause

   This error occurs if your dataset contains fewer than 10,000 records. Amazon Fraud Detector requires data from at least 10,000 events (records) in your dataset for model training.

   Solution

   Make sure that you have data from more than 10,000 events in your dataset.

Missing or different EVENT_LABEL values

1. **Issue type : Error**

   Description

   Greater than 1% of your EVENT_LABEL column are null or are values other than those defined in the model configuration **`$label_values`**. Ensure you have less than 1% of missing values in your EVENT_LABEL column and the values are those defined in the model configuration **`$label_values`**.

   Cause

   This error occurs because of one of the following reasons:

   - More than 1% of the records in the CSV file containing your training data have missing values in the EVENT_LABEL column.
   - More than 1% of the records in the CSV file containing your training data have values in the EVENT_LABEL column that are different than those associated with your event type.

Online Fraud Insights (OFI) model requires that the EVENT_LABEL column in each record be populated with one of the labels that's associated with your event type (or, mapped in `CreateModelVersion`).

Solution

If this error is due to the missing EVENT_LABEL values, consider assigning proper labels to those records or dropping those records from your dataset. If this error is because labels of some records are not among **`label_values`**, make sure to add all the values in EVENT_LABEL column to labels of the event type and mapped to either fraudulent or legitimate (fraud, legit) in model creation.

2. **Issue type : Information**

   Description

   Your EVENT_LABEL column contains null values or label values other than those defined in the model configuration **`$label_values`**. These inconsistent values were converted to 'not fraud' prior to training.

   Cause

   You get this information because of one of the following reasons:

   - Less than 1% of the records in the CSV file containing your training data have missing values in the EVENT_LABEL column
   - Less than 1% of the records in the CSV file containing your training data have values in the EVENT_LABEL column that are different than those associated with your event type.

   The model training in both the cases will succeed. However, the label values of those events that have missing or unmapped label values are converted to legitimate. If you consider this to be an issue, follow solution provided below.

   Solution

   If there are missing EVENT_LABEL values in your dataset, consider dropping those records from your dataset. If the values provided for those EVENT_LABELS are not mapped, make sure that all those values are mapped to either fraudulent or legitimate (fraud, legit) for each event.

Missing or incorrect EVENT_TIMESTAMP values

1. **Issue type : Error**

   Description

   Your training data set contains EVENT_TIMESTAMP with timestamps that do not conform to accepted formats. Ensure the format is one of the accepted date/timestamp formats.

   Cause

   This error occurs if the EVENT_TIMESTAMP column contains value that doesn't comply with the timestamp formats that are supported by Amazon Fraud Detector.

   Solution

   Ensure that the values provided for the EVENT_TIMESTAMP column is compliant with the supported timestamp formats. If you have missing values in the EVENT_TIMESTAMP column, you can either backfill those with values using the supported timestamp format or consider dropping the event completely instead of entering strings such as `none`, `null`, or `missing`.

2. **Issue type : Error**

Your training data set contains EVENT_TIMESTAMP with missing values. Ensure you have no missing values.

Cause

This error occurs if the EVENT_TIMESTAMP column in your dataset has missing values. Amazon Fraud Detector requires that the EVENT_TIMESTAMP column in your dataset have values.

Solution

Ensure that the EVENT_TIMESTAMP column in your dataset has values and those values are compliant with the supported timestamp formats. If you have missing values in the EVENT_TIMESTAMP column, you can either backfill those with values using the supported timestamp format or consider dropping the event completely instead of entering strings such as `none`, `null`, or `missing`.

Data not ingested

Issue type : Error

Description

No ingested events found for training, please check your training configuration.

Cause

This error occurs if you are creating a model with event data stored with Amazon Fraud Detector but did not import your dataset to Amazon Fraud Detector before you started to train your model.

Solution

Use the `SendEvent` API operation, the `CreateBatchImportJob` API operation, or batch import feature in the Amazon Fraud Detector console, to first import your event data and then train your model. See Stored event datasets for more information.

> **Note**
> We recommend waiting 10 minutes after you have finished importing your data before using it to train your model.

You can use Amazon Fraud Detector console to check number of events already stored for each event type. See Viewing metrics of your stored events for more information.

Insufficient variables

Issue type : Error

Description

Dataset must contain at least 2 variables suitable for training.

Cause

This error occurs if your dataset contains less than 2 variables that are suitable for model training. Amazon Fraud Detector considers a variable suitable for model training only if it passes all validations. If a variable fails validation, it is excluded in model training and you will see a message in **Model training diagnostic**.

Solution

Ensure that your dataset has at least two variables populated with values and passed all data validations. Note that the event metadata row where you have provided your column headers (EVENT_TIMESTAMP, EVENT_ID, ENTITY_ID, EVENT_LABEL, etc.) aren't considered as variable.

Missing or incorrect variable type

Issue type : Warning

Description

The expected data type for **`$variable_name`** is NUMERIC. Review and update **`$variable_name`** in your dataset and re-train the model.

Cause

You get this warning if a variable is defined as a NUMERIC variable, but in the dataset, it has values that can't be converted to NUMERIC. As a result, that variable is excluded in model training.

Solution

If you want to keep it as a NUMERIC variable, make sure that values you provide can be converted to float number. Note that if the variable contains missing values, don't fill them with strings such as `nonene`, `null`, or `missing`. If the variable does contain non-numeric values, re-create it as a CATEGORICAL or FREE_FORM_TEXT variable type.

Missing variable values

Issue type : Warning

Description

Greater than **`$threshold`** values for **`$variable_name`** are missing from your training dataset. Consider modifying **`$variable_name`** in your dataset and re-training to improve performance.

Cause

You get this warning if the specified variable is being dropped due to too many missing values. Amazon Fraud Detector allows missing values for a variable. However, if one variable has too many missing values, it doesn't contribute much to the model and that variable is dropped in model training.

Solution

First, verify that those missing values aren't due to mistakes in data collection and preparation. If they are mistakes, then you can consider dropping them from your model training. However, if you do believe those missing values are valuable and still want to keep that variable, you can manually fill missing values with a constant in both model training and real-time inference.

Insufficient unique variable values

Issue type : Warning

Description

The count of unique values of **`$variable_name`** is lower than 100. Review and update **`$variable_name`** in your dataset and re-train the model.

Cause

You get this warning if the number of unique values of the specified variable is less than the 100. The thresholds differ depending on the variable type. With very few unique values, there's a risk that the

dataset isn't general enough to cover the feature space of that variable. As a result, the model might not generalize well on real-time predictions.

Solution

First, make sure the variable distribution is representative of the real business traffic. Then, you can either adopt more fine-trained variables with higher cardinality, such as using `full_customer_name` instead of `first_name` and `last_name` separately or change the variable type to CATEGORICAL, which allows lower cardinality.

Incorrect variable expression

1. **Issue type : Information**

   Description

   Greater than 50% of **`$email_variable_name`** values do not match the expected regular expression http://emailregex.com. Consider modifying **`$email_variable_name`** in your dataset and re-training to improve performance.

   Cause

   This information is displayed if more than 50% records in your dataset has email values that do not comply with a regular email expression and are therefore failing validation.

   Solution

   Format the email variable values to comply with the regular expression. If there are missing email values, we recommend to leave them empty instead of filling them with strings such as `none`, `null`, or `missing`.
2. **Issue type : Information**

   Description

   Greater than 50% of **`$IP_variable_name`** values do not match regular expression for IPv4 or IPv6 addresses https://digitalfortress.tech/tricks/top-15-commonly-used-regex/. Consider modifying **`$IP_variable_name`** in your dataset and re-training to improve performance.

   Cause

   This information is displayed if more than 50% records in your dataset has IP values that do not comply with a regular IP expression and are therefore failing validation.

   Solution

   Format the IP values to comply with the regular expression. If there are missing IP values, we recommend to leave them empty instead of filling them with strings such as `none`, `null`, or `missing`.
3. **Issue type : Information**

   Description

   Greater than 50% of **`$phone_variable_name`** values do not match basic phone regular expression / $pattern/. Consider modifying **`$phone_variable_name`** in your dataset and re-training to improve performance .

   Cause

   This information is displayed if more than 50% records in your dataset has phone numbers that do not comply with a regular phone number expression and are therefore failing validation.

Solution

Format the phone numbers to comply with the regular expression. If there are missing phone numbers, we recommend to leave them empty instead of filling them with strings such as `none`, `null`, or `missing`.

Insufficient unique entities

Issue type : Information

Description

The number of unique entities is less than 1500. Consider including more data to improve performance.

Cause

This information is displayed if your dataset has a smaller number of unique entities than the recommended number. The Transaction Fraud Insights (TFI) model uses both time-series aggregates and generic transaction features to provide the best performance. If your dataset has too few unique entities, then most of your generic data such as IP_ADDRESS, EMAIL_ADDRESS, might not have unique values. Then, there's also a risk that this dataset isn't general enough to cover the feature space of that variable. As a result, the model might not generalize well on transactions from fresh new entities.

Solution

Include more entities. Extend your training data time range, if needed.

Quotas

Your AWS account has default quotas, formerly referred to as *limits*, for each Amazon Web Service. Unless otherwise noted, each quota is Region-specific. You can request a quota increase for all adjustable quotas mentioned in the tables below. For more information, see Requesting a quota increase

The following tables outline Amazon Fraud Detector quotas by component.

Amazon Fraud Detector models

Quota name	Default quota	Adjustable
Training data size	5 GB	No
Models per account	50	No
Versions per model	200	No
Deployed model versions per account	5	No
Concurrent training jobs per account	3	No
Concurrent training jobs per model	1	No

Amazon Fraud Detector detectors / variables / outcomes / rules

Quota name	Default quota	Adjustable
Variables per account	5000	No
Rules per account	5000	No
Outcomes per account	5000	No
Detectors per account	100	No
Draft versions per detector	100	No
Models per detector version	10	No
Labels per account	100	No
Event types per account	100	No
Entity types per account	100	No

Amazon Fraud Detector API

Quota name	Default quota	Adjustable
GetEventPrediction API calls per second	200 TPS	Yes
Size of payload per GetEventPrediction API call	256 KB	No
Number of inputs per GetEventPrediction API call	5000	No

Document history

The following table describes important changes in Amazon Fraud Detector User Guide. We also update the Amazon Fraud Detector User Guide frequently to address the feedback that you send us.

update-history-change	update-history-description	update-history-date
Chapter update	Updated the introductory chapter with additional information about Amazon Fraud Detector	April 11, 2022
Variable enrichment	Enable enrichment of some of the raw data you provide to boost performance for the models that use these data elements and that were trained before February 8, 2022.	February 8, 2022
Opt-out policies	Use **opt-out policies** to opt out of having your event data used to develop or improve the quality of Amazon Fraud Detector.	January 6, 2022
Confused deputy prevention	Create policies to prevent a third-party or a cross-service entity from manipulating an entity with permissions to act on its behalf to gain access to resources in your account.	December 6, 2021
Create event dataset	Use the guidance provided in **Create event dataset** to prepare and gather data for training your model.	November 22, 2021
Prediction explanations	Use **Prediction explanations** to get insight into how each event variable impacted your model's fraud prediction scores.	November 10, 2021
Troubleshoot	Use information in **Troubleshoot training data issues** to help diagnose and resolve issues you might see in Amazon Fraud Detector console when you train your model.	October 11, 2021
Transaction fraud insights model	Use **Transaction fraud insights (TFI)** model to detect online or card-not-present transaction fraud.	October 11, 2021

Stored events	Store your event data in Amazon Fraud Detector and use the stored data to later train your models. By storing event data in Amazon Fraud Detector, you can train models that use auto-computed variables to improve performance, simplify model retraining, and update fraud labels to close the machine learning feedback loop.	October 11, 2021
Model variable importance	Use **Model variable importance** to gain insight into what is driving performance of your model up or down and which of your model variables contribute the most. And then tweak your model to improve overall performance.	July 9, 2021
Integration with AWS CloudFormation	Use AWS CloudFormation to manage your Amazon Fraud Detector resources.	May 10, 2021
Batch predictions	Use **Batch predictions** to get predictions for a set of events that do not require real-time scoring.	March 31, 2021
Chapter rework	Rework of Get started and other sections	July 17, 2020
Initial release	Initial release	December 2, 2019