Amazon Lookout for Vision

Developer Guide

Amazon Lookout for Vision: Developer Guide

Table of Contents

What is Amazon Lookout for Vision?

You can use Amazon Lookout for Vision to find visual defects in industrial products, accurately and at scale. It uses computer vision to identify missing components in an industrial product, damage to vehicles or structures, irregularities in production lines, and even minuscule defects in silicon wafers—or any other physical item where quality is important such as a missing capacitor on printed circuit boards.

Key benefits

Amazon Lookout for Vision provides the following benefits:

- **Quickly and efficiently improve processes** – You can use Amazon Lookout for Vision to implement computer vision-based inspection in industrial processes quickly and efficiently, at scale. You can provide as few as 30 baseline good images and Lookout for Vision can automatically build a custom ML model for defect detection. You can then process images from IP cameras, in batch or in real time, to quickly and accurately identify anomalies like dents, cracks, and scratches.
- **Increase production quality, fast** – With Amazon Lookout for Vision you can reduce defects in production processes, in real time. It identifies and reports visual anomalies in a dashboard so you can take action quickly to stop more defects from occurring—increasing production quality and reducing costs.
- **Reduce operational costs** – Amazon Lookout for Vision reports trends in your visual inspection data, such as identifying processes with the highest defect rate or flagging recent variations in defects. Using this information, you can determine whether to schedule maintenance on the process line or reroute production to another machine before costly, unplanned downtime occurs.

Are you a first-time Amazon Lookout for Vision end user?

If you're a first-time user of Amazon Lookout for Vision, we recommend that you read the following sections in order:

1. **Setting up Amazon Lookout for Vision (p. 2)** – In this section, you set your account details.
2. **Getting Started with the Amazon Lookout for Vision console (p. 10)** – In this section, you learn about creating your first Amazon Lookout for Vision model.

Setting up Amazon Lookout for Vision

In this section, you sign up for an AWS account and then create an IAM user, a security group, and optionally download example images that you can use to create a model.

For information about the AWS Regions that support Amazon Lookout for Vision, see Amazon Lookout for Vision Endpoints and Quotas.

> **Note**
> Amazon Lookout for Vision isn't compatible with Microsoft Internet Explorer 11. We recommend that you use a supported browser, such as Firefox or Google Chrome.

Topics

Step 1: Create an AWS account

In this section, you sign up for an AWS account. If you already have an AWS account, skip this step.

When you sign up for Amazon Web Services (AWS), your AWS account is automatically signed up for all AWS services, including IAM. You are charged only for the services that you use.

To create an AWS account

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

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

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

Step 2: Create an IAM administrator user and group

When you create an AWS account, you get a single sign-in identity that has access to all of the AWS services and resources in the account. This identity is called the AWS account *root user*. If you use the AWS root user account to sign in to the AWS Management Console , you have complete access to all of the AWS resources in your account.

We strongly recommend that you do *not* use the root user for everyday tasks, even the administrative ones. Instead, we recommend that you adhere to the best practice in Create Individual IAM Users, which is to create an AWS Identity and Access Management (IAM) administrator user. Then securely lock away the root user credentials, and use them to perform only a few account and service management tasks.

For more information, see How Amazon Lookout for Vision works with IAM (p. 109).

To create an administrator user and sign in to the console

1. Create an administrator user in your AWS account. For instructions, see Creating Your First IAM User and Administrators Group in the *IAM User Guide*.

   Note
   An IAM user with administrator permissions has unrestricted access to the AWS services in your account. You can restrict permissions as necessary. For more information, see Step 3: Set up permissions (p. 3).
2. Sign in to the AWS Management Console.

   To sign in to the AWS Management Console as a IAM user, you must use a special URL. For more information, see How Users Sign In to Your Account in the *IAM User Guide*.

Step 3: Set up permissions

The Identity and Access Management (IAM) user or group that uses Amazon Lookout for Vision console and SDK operations needs access permissions to the Lookout for Vision console, SDK operations, and the Amazon S3 bucket that you use for model training.

Note
If you only use Lookout for Vision SDK operations, you can use policies that are scoped to Lookout for Vision SDK operations. For more information, see Step 6: Set up SDK permissions (p. 6).

Setting console access with AWS managed policies

Use the following AWS managed policies to apply appropriate access permissions for the Amazon Lookout for Vision console and SDK operations.

- AmazonLookoutVisionConsoleFullAccess (p. 115) — allows full access to the Amazon Lookout for Vision console and SDK operations. You need `AmazonLookoutVisionConsoleFullAccess` permissions to create the console bucket. For more information, see Step 4: Create the console bucket (p. 4).
- AmazonLookoutVisionConsoleReadOnlyAccess (p. 117)— allows read-only access to the Amazon Lookout for Vision console and SDK operations.

To allow access to the Lookout for Vision console and SDK operations, add the desired managed policy to the IAM user or group that needs access. For more information, see Changing permissions for an IAM user.

For information about AWS managed policies, see AWS managed policies.

Setting Amazon S3 bucket permissions

Amazon Lookout for Vision uses an Amazon S3 bucket to store the following files:

- Dataset images — Images that are used to train a model. For more information, see Creating your dataset (p. 24).
- Amazon SageMaker Ground Truth format manifest files. For example, the manifest file output from SageMaker GroundTruth job. For more information, see Creating a dataset using an Amazon SageMaker Ground Truth manifest file (p. 28).
- The output from model training.

If you use the console, Lookout for Vision creates an Amazon S3 bucket (console bucket) to manage your projects. The `LookoutVisionConsoleReadOnlyAccess` and `LookoutVisionConsoleFullAccess` managed policies include Amazon S3 access permissions for the console bucket.

You can use the console bucket to store dataset images and SageMaker Ground Truth format manifest files. Alternatively, You can use a different Amazon S3 bucket. The bucket must be owned by your AWS account and must be located in the AWS Region in which you are using Lookout for Vision.

To use a different bucket, add the following inline policy to the desired IAM user or group. Replace `my-bucket` with the name of the desired bucket. For information about adding IAM policies, see Creating IAM Policies.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "LookoutVisionS3BucketAccessPermissions",
            "Effect": "Allow",
            "Action": [
                "s3:GetBucketLocation",
                "s3:ListBucket"
            ],
            "Resource": [
                "arn:aws:s3:::my-bucket"
            ]
        },
        {
            "Sid": "LookoutVisionS3ObjectAccessPermissions",
            "Effect": "Allow",
            "Action": [
                "s3:GetObject",
                "s3:GetObjectVersion",
                "s3:PutObject"
            ],
            "Resource": [
                "arn:aws:s3:::my-bucket/*"
            ]
        }
    ]
}
```

Step 4: Create the console bucket

You use an Amazon Lookout for Vision project to create and manage your models. When you first open the Amazon Lookout for Vision console in a new AWS Region, Lookout for Vision creates an Amazon S3 bucket (console bucket) to store your projects. You should note the console bucket name somewhere where you can refer to it later because you might need to use the bucket name in AWS SDK operations or console tasks, such as creating a dataset.

The format of the bucket name is `lookoutvision-<region>-<random value>`. The random value ensures that there isn't a collision between bucket names.

To create the console bucket

1. Ensure that the IAM user or group you are using has `AmazonLookoutVisionConsoleFullAccess` permission. For more information, see Step 3: Set up permissions (p. 3).
2. Open the Amazon Lookout for Vision console at https://console.aws.amazon.com/lookoutvision/.
3. Choose **Get started**.
4. If this is the first time that you've opened the console in the current AWS Region, do the following in the **First Time Set Up** dialog box:
   a. Copy down the name of the Amazon S3 bucket that's shown. You'll need this information later.
   b. Choose **Create S3 bucket** to let Amazon Lookout for Vision create an Amazon S3 bucket (console bucket) on your behalf.
5. Close the browser window.

Step 5: Set up the AWS CLI and AWS SDKs

The following steps show you how to install the AWS Command Line Interface (AWS CLI) and AWS SDKs. The examples in this documentation use the CLI and SDKs. There are a number of different ways to authenticate AWS SDK calls. The examples in this guide assume that you're using a default credentials profile for calling AWS CLI commands and AWS SDK API operations.

Installing the AWS SDKS

Follow the steps to download and configure the AWS SDKs.

To set up the AWS CLI and the AWS SDKs

1. Download and install the AWS CLI and the AWS SDKs that you want to use. This guide provides examples for the AWS CLI and Python. For information about installing AWS SDKs, see Tools for Amazon Web Services.
2. Create an access key for the user you created in Step 2: Create an IAM administrator user and group (p. 2).
   a. Sign in to the AWS Management Console and open the IAM console at https://console.aws.amazon.com/iam/.
   b. In the navigation pane, choose **Users**.
   c. Choose the name of the user you created in Step 2: Create an IAM administrator user and group (p. 2).
   d. Choose the **Security credentials** tab.
   e. Choose **Create access key**, choose **Download .csv file** to save the access key ID and secret access key to a CSV file on your computer, and then choose **Close**.

      Important
      Store the file in a secure location. You will not have access to the secret access key again after this dialog box closes.
3. If you have installed the AWS CLI, you can configure the credentials and Region for most AWS SDKs by entering `aws configure` at the command prompt. Otherwise, use the following instructions.
4. On your computer, navigate to your home directory, and create an `.aws` directory. On Unix-based systems, such as Linux or macOS, this is in the following location:

```
~/.aws
```

   On Windows, this is in the following location:

```
%HOMEPATH%\.aws
```

5. In the `.aws` directory, create a new file named `credentials`.
6. Open the credentials CSV file that you created in step 2. Copy its contents into the `credentials` file using the following format:

```
[default]
aws_access_key_id = your_access_key_id
aws_secret_access_key = your_secret_access_key
```

   Substitute your access key ID and secret access key for *your_access_key_id* and *your_secret_access_key*.
7. Save the `Credentials` file and delete the CSV file.
8. In the `.aws` directory, create a new file named `config`.
9. Open the `config` file and enter your Region in the following format.

```
[default]
region = your_aws_region
```

   Substitute your Region (for example, `us-west-2`) for *your_aws_region*.

   Note
   If you don't select a Region, us-east-1 is used by default.
10. Save the `config` file.

Step 6: Set up SDK permissions

The Identity and Access Management (IAM) user or group that uses Amazon Lookout for Vision SDK operations needs access permissions to the Lookout for Vision API and the Amazon S3 bucket that you use for model training.

Setting SDK operation access with AWS managed policies

Use the following AWS managed policies to add appropriate access permissions to Amazon Lookout for Vision SDK operations.

- AmazonLookoutVisionFullAccess (p. 114) — allows full access to Amazon Lookout for Vision SDK operations.
- AmazonLookoutVisionReadOnlyAccess (p. 113) — allows access to the read-only SDK operations.

The managed policies for the console also provide access permissions for SDK operations. For more information, see Step 3: Set up permissions (p. 3).

To allow access to Lookout for Vision SDK operations, add the desired managed policy to the IAM user or group that needs access. For more information, see Changing permissions for an IAM user.

For information about AWS managed policies, see AWS managed policies.

Setting Amazon S3 Bucket permissions

To train a model, you need an Amazon S3 bucket with appropriate permissions to store the images, manifest files and training output. The bucket must be owned by your AWS account and must be located in the AWS Region in which you are using Amazon Lookout for Vision. For more information, see Getting started with the AWS SDK (p. 13).

The SDK-only managed policies (`AmazonLookoutVisionFullAccess` and `AmazonLookoutVisionReadOnlyAccess`) don't include Amazon S3 bucket permissions and you need to apply the following permission policy to access the buckets you use, including existing console buckets.

The console managed policies (`AmazonLookoutVisionConsoleFullAccess` and `AmazonLookoutVisionConsoleReadOnlyAccess`) include access permissions to the console bucket. If you are accessing the console bucket with SDK operations and have console managed policy permissions, you don't need to use the following policy. For more information, see Step 3: Set up permissions (p. 3).

Deciding task permissions

Use the following information to decide which permissions are needed for the tasks you want to do.

Creating a dataset

To create a dataset with CreateDataset, you need the following permissions.

- `s3:GetBucketLocation` — allows Lookout for Vision to validate that your bucket is in the same region in which you are using Lookout for Vision.
- `s3:GetObject` — Allows access to the manifest file specifed in the `DatasetSource` input parameter. If you want to specify an exact S3 object version of the manifest file, you also need `s3:GetObjectVersion` on the manifest file. For more information, see Using versioning in S3 buckets.

Creating a model

To create a model with CreateModel, you need the following permissions.

- `s3:GetBucketLocation` — allows Lookout for Vision to validate that your bucket is in the same region in which you are using Lookout for Vision.
- `s3:GetObject` — allows access to the images specified in the project's training and test datasets.
- `s3:PutObject` — allows permission to store training output in the specified bucket. You specify the output bucket location in the `OutputConfig` parameter. Optionally, you can scope permissions down to only object keys specified in the `Prefix` field of the `S3Location` input field. For more information, see API_OutputConfig.

Accessing images, manifest files, and training output

Amazon S3 bucket permissions aren't required to view Amazon Lookout for Vision operation responses. You do need `s3:GetObject` permission if you want to access images, manifests files, and training output referenced in operation responses. If you are accessing a versioned Amazon S3 object, you need `s3:GetObjectVersion` permission.

Setting Amazon S3 bucket policy

You can use the following policy to specify the Amazon S3 bucket permissions needed to create a dataset (`CreateDataset`), create a model (`CreateModel`), and access images, manifest files, and training output. Change the value of *`my-bucket`* to the name of the bucket that you want use.

You can adjust the policy to your needs. For more information, see Deciding task permissions (p. 7). Add the policy to the desired IAM user or role. For more information, see Creating IAM Policies.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "LookoutVisionS3BucketAccess",
            "Effect": "Allow",
            "Action": "s3:GetBucketLocation",
            "Resource": [
                "arn:aws:s3:::my-bucket"
            ],
            "Condition": {
                "Bool": {
                    "aws:ViaAWSService": "true"
                }
            }
        },
        {
            "Sid": "LookoutVisionS3ObjectAccess",
            "Effect": "Allow",
            "Action": [
                "s3:GetObject",
                "s3:GetObjectVersion",
                "s3:PutObject"
            ],
            "Resource": [
                "arn:aws:s3:::my-bucket/*"
            ],
            "Condition": {
                "Bool": {
                    "aws:ViaAWSService": "true"
                }
            }
        }
    ]
}
```

Step 7: (Optional) Using your own AWS Key Management Service key

You can use AWS Key Management Service (KMS) to manage encryption for the input images that you store in Amazon S3 buckets.

By default your images are encrypted with a key that AWS owns and manages. You can also choose to use your own AWS Key Management Service (KMS) key. For more information, see AWS Key Management Service concepts.

If you want to use your own KMS key, use the the following policy to specify the KMS key. Change *kms_key_arn* to the ARN of the KMS key (or KMS alias ARN) that you want to use. Alternatively, specify * to use any KMS key. For information about adding the policy to an IAM user or role, see Creating IAM Policies.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "LookoutVisionKmsDescribeAccess",
```

```
            "Effect": "Allow",
            "Action": "kms:DescribeKey",
            "Resource": "kms_key_arn"
        },
        {
            "Sid": "LookoutVisionKmsCreateGrantAccess",
            "Effect": "Allow",
            "Action": "kms:CreateGrant",
            "Resource": "kms_key_arn",
            "Condition": {
                "StringLike": {
                    "kms:ViaService": "lookoutvision.*.amazonaws.com"
                },
                "Bool": {
                    "kms:GrantIsForAWSResource": "true"
                }
            }
        }
    ]
}
```

Step 8: (Optional) Prepare example images

Amazon Lookout for Vision provides example images of circuit boards that AWS customers can use to learn how to create, test, and use a model.

You can copy the images from the https://github.com/aws-samples/amazon-lookout-for-vision GitHub repository. The images are in the `circuitboard` folder.

The `circuitboard` folder has the following folders.

- `train` – Images you can use in a training dataset.
- `test` – Images you can use in a test dataset.
- `extra_images` – Images you can use to run a trial detection or to try out your trained model with the DetectAnomalies operation.

The `train` and `test` folders each have a subfolder named `normal` (contains images that are normal) and a subfolder named `anomaly` (contains images with anomalies).

> **Note**
> Later, when you create a dataset with the console, Amazon Lookout for Vision can use the folder names (`normal` and `anomaly`) to automatically label the images. For more information, see the section called "Amazon S3 bucket" (p. 26).

To prepare the dataset images

1. Clone the https://github.com/aws-samples/amazon-lookout-for-vision repository to your computer. For more information, see Cloning a repository.
2. Create an Amazon S3 bucket. For more information, see How do I create an S3 Bucket?.
3. At the command prompt, enter the following command to copy the dataset images from your computer to your Amazon S3 bucket.

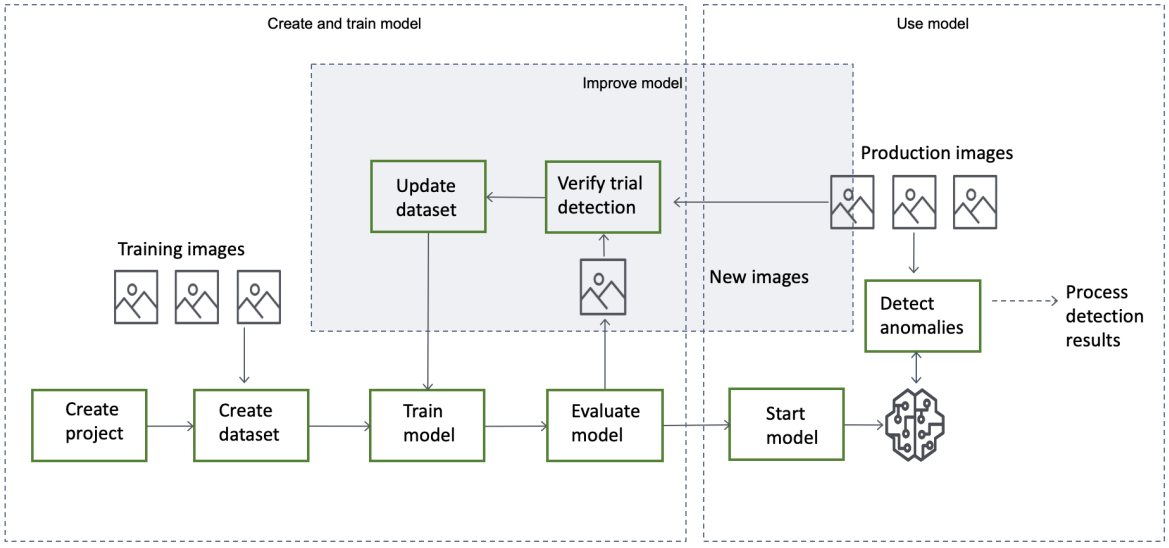
```
aws s3 cp --recursive your-repository-folder/circuitboard s3://your-bucket/circuitboard
```

For more information, see Getting Started with the Amazon Lookout for Vision console (p. 10) and Getting started with the AWS SDK (p. 13).

Getting Started with the Amazon Lookout for Vision console

The Getting Started instructions show you how to create, train, evaluate, and use a model that detects anomalies in images. To help you learn, Amazon Lookout for Vision provides example images of circuit boards (that is, `circuitboard`) that you can use.

The general workflow is as follows:



Topics

Set up Amazon Lookout for Vision

Set up Amazon Lookout for Vision and prepare the *circuitboard* example images for use in Getting Started.

Step 1: Set up Amazon Lookout for Vision (p. 2).

Be sure to do Step 8: (Optional) Prepare example images (p. 9).

Create a project

Create a project to manage the datasets and the models that you create. A project should be used for a single use case, such as detecting anomalies in a single type of machine part.

You can use the dashboard to get an overview of your projects. For more information, see Using the Amazon Lookout for Vision dashboard (p. 20).

Step 2: Create your project (p. 22).

Create a dataset

A dataset is a set of images and labels (that is, normal or anomaly) that describe those images. You use datasets to train and test the models you create. The images should represent a single type of object. For more information, see Preparing images for a dataset (p. 24). With Amazon Lookout for Vision you can have a project that uses a single dataset, or a project that has separate training and test datasets. We recommend using a single dataset project unless you want finer control over training, testing, and performance tuning. To create a dataset, you import the images in one of the following ways:

- Import images from your local computer.
- Import images from an S3 bucket. Amazon Lookout for Vision can label the images using the folder names that contain the images.
- Import an Amazon SageMaker Ground Truth manifest file.

To help you understand how Amazon Lookout for Vision works, create a project with a training dataset and a test dataset. Import your training dataset images from *s3://your bucket/circuitboard/train/*. Import your test dataset images from *s3://your bucket/circuitboard/test/*.

Step 3: Create your dataset (Import images from Amazon S3) (p. 26).

After you create the dataset, you can add more images and also label the images. For more information, see Editing your dataset (p. 34). For Getting Started, you don't need to add any more images to create your first model.

Train your model

Training creates a model and trains it to predict the presence of anomalies in images. A new version of your model is created each time you train.

At the start of training, Amazon Lookout for Vision chooses the most suitable algorithm to train your model with. The model is trained and then tested. When you train a single dataset project, the dataset is internally split to create a training dataset and a test dataset. In Getting Started, your project has separate training and test datasets. In this configuration, Amazon Lookout for Vision trains your model with the training dataset and tests the model with the test dataset.

Step 4: Train your model (p. 35).

> **Note**
> You are charged for the amount of time it takes to successfully train your model.

Evaluate your model

Evaluate the performance of your model by using the performance metrics created during testing.

Performance metrics enable you to understand the performance of your trained model, and decide if you're ready to use it in production.

Step 5: View your performance metrics (p. 42).

Step 6: Evaluate and improve your model (p. 40).

If the performance metrics indicate that improvements are needed, you can add more training data by running a trial detection task with new images. After the task completes you can verify the results and add the verified images to your training dataset. Alternatively, you can add new training images directly to the dataset. In these steps, improve the model by running a trial detection task and adding the verified images to your training dataset. Use the example images in the `extra_images` folder.

Step 7: Run a trial detection task (p. 45).

Step 8: Retrain your model (p. 35).

If the performance results indicate acceptable performance, you can verify the performance by running a trial detection. The performance results after retraining should confirm the readiness of your model.

Use your model

Before you can use your model, you start the model with the StartModel operation. You can get the `StartModel` CLI command for your model from the console.

Step 9: Start your model (p. 49).

A trained Amazon Lookout for Vision model predicts whether an input image contains normal or anomalous content.

To make a prediction with your model, call the DetectAnomalies operation and pass an input image from your local computer. For Getting Started, use one of the images in the `extra_images` folder. You can get the CLI command that calls `DetectAnomalies` from the console.

Step 10: Detect anomalies in an image (p. 55).

You are charged for the time that your model is running. If you are no longer using your model, use the StopModel operation to stop the model. You can get the CLI command from the console.

Step 11: Stop your model (p. 52).

Use your dashboard

You can use the dashboard to get an overview of all your projects and overview information for individual projects.

Step 12: Use your dashboard (p. 20).

Getting started with the AWS SDK

The Getting Started instructions show you how to use the AWS SDK to create, train, evaluate, and use a model that detects anomalies in an image. We recommend reading Getting Started with the Amazon Lookout for Vision console (p. 10) first. This section includes AWS CLI commands. You can get more information and Python examples by choosing the **More information** link at the end of each section.

The Amazon Lookout for Vision Lab GitHub repository includes a Python Notebook that you can use to create a model with the circuitboard example images (p. 9). Alternatively, you can use the Amazon Lookout for Vision example code in the AWS Code Examples Repository.

To prepare, train, evaluate, improve, and use a model, do the following:

Topics

Using Lookout for Vision with an AWS SDK

AWS software development kits (SDKs) are available for many popular programming languages. Each SDK provides an API, code examples, and documentation that make it easier for developers to build applications in their preferred language.

SDK documentation	Code examples
AWS SDK for C++	AWS SDK for C++ code examples
AWS SDK for Go	AWS SDK for Go code examples
AWS SDK for Java	AWS SDK for Java code examples
AWS SDK for JavaScript	AWS SDK for JavaScript code examples
AWS SDK for .NET	AWS SDK for .NET code examples
AWS SDK for PHP	AWS SDK for PHP code examples
AWS SDK for Python (Boto3)	AWS SDK for Python (Boto3) code examples
AWS SDK for Ruby	AWS SDK for Ruby code examples

For examples specific to Lookout for Vision, see Code examples for Lookout for Vision (p. 82).

Example availability
Can't find what you need? Request a code example by using the **Provide feedback** link at the bottom of this page.

Set up Amazon Lookout for Vision

Set up Amazon Lookout for Vision and prepare the example images for use in Getting Started.

More information: Setup Amazon Lookout for Vision (p. 2).

Be sure to do Step 8: (Optional) Prepare example images (p. 9).

Create a project

You create a project to manage the datasets and the models that you create.

Note
Amazon Lookout for Vision stores your project files in an Amazon S3 bucket created for you when you first open the console. If you haven't previously used the console, create the console bucket (p. 4).

To create a project, call the CreateProject operation and specify a project name. The following example creates a project named `my-sdk-project`.

```
aws lookoutvision create-project --project-name my-sdk-project
```

The JSON response examples use a project named `my-sdk-project`.

```
{
    "ProjectMetadata": {
        "ProjectArn": "arn:aws:lookoutvision:us-east-1:nnnnnnnnnnn:project/my-sdk-project",
        "ProjectName": "my-sdk-project",
        "CreationTimestamp": 1607277318.389
    }
}
```

More information: Create your project (p. 23).

Upload your images

You store the images used to train a model in an Amazon S3 bucket. For Getting Started, use the circuitboard images in the Amazon Lookout for Vision Lab GitHub repository. If you haven't already, do Step 8: (Optional) Prepare example images (p. 9).

Create a manifest file

To create a dataset with the AWS SDK, you need an Amazon SageMaker Ground Truth format manifest file. A manifest file provides information about the images used for training. For example, the S3 bucket location of images and labels that classify images as normal or anomalous. Each image in the manifest file is represented by a JSON Line, as shown in the following example:

```
{
    "source-ref": "s3://lookoutvision-console-us-east-1-nnnnnnnnnn/gt-job/manifest/
IMG_1133.png",
    "anomaly-label": 1,
    "anomaly-label-metadata": {
        "confidence": 0.95,
        "job-name": "labeling-job/testclconsolebucket",
        "class-name": "normal",
        "human-annotated": "yes",
        "creation-date": "2020-04-15T20:17:23.433061",
        "type": "groundtruth/image-classification"
    }
}
```

You can use an existing manifest file, such as the output from an SageMaker Ground Truth job, or you can create your own manifest file.

Tip
If you are creating your dataset using the circuitboard example images, you have two options:

1. Create the manifest file using code. The Amazon Lookout for Vision Lab Python Notebook shows how to create the manifest file for the circuitboard example images. Alternatively, use the Datasets example code in the AWS Code Examples Repository.
2. If you've already followed Getting Started with the Amazon Lookout for Vision console (p. 10) and created a dataset with the circuitboard example images, reuse the manifest files created for you by Amazon Lookout for Vision. The training and test manifest file locations are `s3://`*`bucket`*`/datasets/`*`project name`*`/`*`train or test`*`/manifests/output/output.manifest`.

Upload your completed manifest files to a folder within the bucket that contains your images.

More information: Create a manifest file (p. 28).

Create a dataset

A dataset is a set of images and labels (that is, normal or anomaly) that describe those images. You use datasets to train and test the models you create. With Amazon Lookout for Vision you can create a project that uses a single dataset, or create a project that has separate training and test datasets.

For Getting Started, create a project with a training dataset and a test dataset.

To create the training dataset, call CreateDataset and specify a dataset type of `train`.

```
aws lookoutvision create-dataset --project-name "my-sdk-project"\
  --dataset-type train\
  --dataset-source '{ "GroundTruthManifest": { "S3Object": { "Bucket": "bucket", "Key":
 "manifest file" } } }'
```

To create the test dataset, call `CreateDataset` and specify a dataset type of `test`.

```
aws lookoutvision create-dataset --project-name "my-sdk-project"\
  --dataset-type test\
  --dataset-source '{ "GroundTruthManifest": { "S3Object": { "Bucket": "bucket", "Key":
 "manifest file" } } }'
```

Depending on the number of images you have, it might take a while to create the dataset. You can use the `DescribeDataset` operation to check the current status.

```
aws lookoutvision describe-dataset --project-name "my-sdk-project"\
  --dataset-type train or test
```

If the value of `Status` is `CREATE_COMPLETE`, the dataset has been successfully created.

```
{
    "DatasetDescription": {
        "ProjectName": "my-sdk-project",
        "DatasetType": "test",
        "CreationTimestamp": 1606579439.278,
        "LastUpdatedTimestamp": 1606579453.6,
        "Status": "CREATE_COMPLETE",
        "StatusMessage": "The dataset was created successfully.",
        "ImageStats": {
            "Total": 44,
            "Labeled": 44,
            "Normal": 34,
            "Anomaly": 10
        }
    }
}
```

More information: Create your dataset (p. 31).

Train your model

At the start of training, Amazon Lookout for Vision chooses the most suitable algorithm to train the model with. The model is trained and then tested using the test dataset. A new version of your model is created each time you train it.

When you train a single dataset project, the dataset is internally split to create a training dataset and a test dataset. If your project has separate training and test datasets, they are separately used to train and test the model.

To train a model, use the CreateModel operation. You provide the project name that contains the dataset that you want to use. You also specify an output location for the training results.

```
aws lookoutvision create-model --project-name "my-sdk-project"\
  --output-config '{ "S3Location": { "Bucket": "output bucket", "Prefix":  "output
 folder" } }'
```

In the response, note the value of `ModelVersion`. If this is the first time you've trained a model in this project, the value is `1`.

```
{
    "ModelMetadata": {
        "CreationTimestamp": 1607278983.132,
        "ModelVersion": "1",
        "ModelArn": "arn:aws:lookoutvision:us-east-1:nnnnnnnnnnn:model/my-sdk-project/1",
        "Status": "TRAINING",
        "StatusMessage": "The model is being trained."
    }
}
```

Training takes a while to complete. To check the current status, call DescribeModel and pass the project name and the model version.

```
aws lookoutvision  describe-model --project-name "my-sdk-project"\
    --model-version "model version"
```

To get the current status, check the `Status` field in the response. Training has successfully completed when the value of `Status` is `TRAINED`.

```
{
    "ModelDescription": {
        "ModelVersion": "1",
        "ModelArn": "arn:aws:lookoutvision:us-east-1:nnnnnnnnnnnn:model/my-sdk-project/1",
        "CreationTimestamp": 1606581265.586,
        "Status": "TRAINING",
        "StatusMessage": "The model is being trained.",
        "OutputConfig": {
            "S3Location": {
                "Bucket": "lookoutvision-us-east-1-nnnnnnnnnn",
                "Prefix": "my-sdk-project-model-output/"
            }
        }
    }
}
```

More information: Train your model (p. 37).

Note
You are charged for the amount of time it takes to successfully train your model.

Evaluate your model

Evaluate the performance of your model by using the performance metrics created during testing.

Performance metrics enable you to understand the performance of your trained model, and decide if you're ready to use it in production. For more information, see Improving your model (p. 40).

You can view the performance metrics by calling the `DescribeModel` operation.

```
aws lookoutvision  describe-model --project-name "my-sdk-project"\
    --model-version "model version"
```

The `Performance` field in the response includes summary precision, recall, and F1 metrics for the model. `EvaluationManifest` contains the location of the evaluation manifest which includes metrics for individual test images. For more information, see Using the evaluation manifest (p. 43). `EvaluationResult` contains the location of overview performance metrics. For more information, see Reviewing the evaluation result (p. 44).

```
{
    "ModelDescription": {
        "ModelVersion": "1",
        "ModelArn": "arn:aws:lookoutvision:us-east-1:nnnnnnnnnnnn:model/my-sdk-project/1",
        "CreationTimestamp": 1606581265.586,
        "Status": "TRAINED",
        "StatusMessage": "Training completed successfully.",
        "Performance": {
            "F1Score": 1.0,
            "Recall": 1.0,
```

```
            "Precision": 1.0
        },
        "OutputConfig": {
            "S3Location": {
                "Bucket": "lookoutvision-us-east-1-nnnnnnnnnn",
                "Prefix": "my-sdk-project-model-output/"
            }
        },
        "EvaluationManifest": {
            "Bucket": "lookoutvision-us-east-1-nnnnnnnnnn",
            "Key": "my-sdk-project-model-output/EvaluationManifest-my-sdk-project-1.json"
        },
        "EvaluationResult": {
            "Bucket": "lookoutvision-us-east-1-nnnnnnnnnn",
            "Key": "my-sdk-project-model-output/EvaluationResult-my-sdk-project-1.json"
        },
        "EvaluationEndTimestamp": 1606583060.681
    }
}
```

Note
You can use the Amazon Lookout for Vision console to create a trial detection and improve your model by adding verified images to your dataset. You can't use the AWS SDK to create a trial detection.

If the performance metrics indicate that improvements are needed, add more labeled training images by calling the UpdateDatasetEntries operation. Afterwards, retrain your model.

More information:

- View performance metrics (p. 43).
- Add more images (p. 34).

Use your model

To detect anomalies in new images you first have to start your model with the StartModel operation. You pass the project name and the version of the model that you want to start.

```
aws lookoutvision start-model --project-name "my-sdk-project"\
  --model-version "model version"\
  --min-inference-units 1
```

It might take a few minutes to start your model. Check the status by calling `DescribeModel`.

```
aws lookoutvision  describe-model --project-name "my-sdk-project"\
    --model-version "model version"
```

The model is running if the Status field is `HOSTED`.

More information: Start your model (p. 49).

A trained Amazon Lookout for Vision model predicts if an input image contains normal or anomalous content.

To make a prediction with your model, call the DetectAnomalies operation and pass an input image from your local computer. You also pass the project name and the version of the model that you want to use. Use a PNG or JPEG format image.

```
aws lookoutvision detect-anomalies  --project-name "my-sdk-project"\
    --model-version "model version"\
    --content-type "image/png or image/jpeg"\
    --body "image path and file name"
```

More information: Detect anomalies in an image (p. 55).

The response includes a prediction for the presence of an anomaly and the confidence that Amazon Lookout for Vision has in the accuracy of the prediction.

```
{
    "DetectAnomalyResult": {
        "Source": {
            "Type": "direct"
        },
        "IsAnomalous": false,
        "Confidence": 0.9999964237213135
    }
}
```

You are charged for the time that your model is running. If you are no longer using your model, use the StopModel operation to stop the model.

```
aws lookoutvision stop-model --project-name "my-sdk-project"\
    --model-version "model version"
```

More information: Stop your model (p. 53).

Using the Amazon Lookout for Vision dashboard

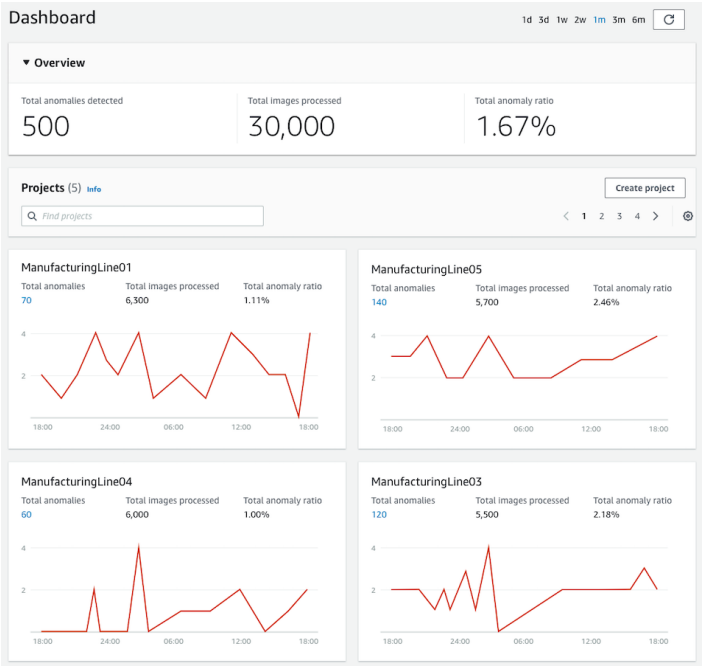
The dashboard provides an overview of metrics for your Amazon Lookout for Vision projects, such as the total number of anomalies detected over the last week. With the dashboard you get an overview for all of your projects and an overview for each individual project. You can choose the timeline over which metrics are shown. You can also use the dashboard to create a new project.

The **Overview** section shows the total number of projects, total number of images, and the total number of images detected by all of your projects.

The **Projects** section shows the following overview information for individual projects:

- The total number or anomalies detected.
- The total number of images processed.
- The total anomaly ratio (that is, the percentage of images detected with an anomaly).
- A graph shows the anomaly detections over the chosen time frame.

You can also get further information about a project.

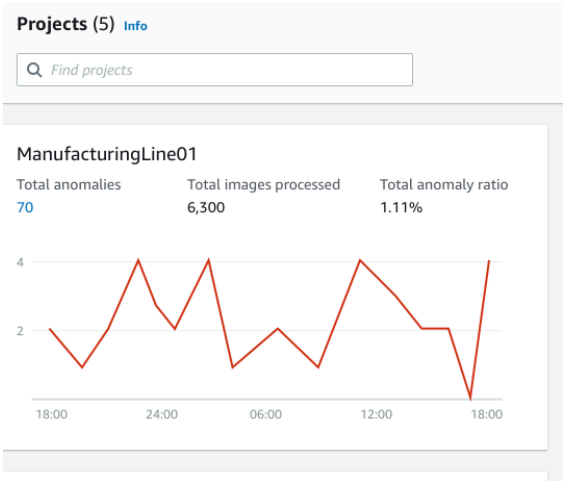


To use your dashboard

1. Open the Amazon Lookout for Vision console at https://console.aws.amazon.com/lookoutvision/.
2. Choose **Get started**.
3. In the left navigation pane, choose **Dashboard**.
4. To view metrics over a specific time frame, do the following:

a. Choose the time frame in the upper-right side of the dashboard.
b. Choose the refresh button to show the dashboard with the new timeline.

5. To get further details about a project, choose the project name in the **Projects** section (for example, `ManufacturingLine01`).



6. To create a project, choose **Create project** in the **Projects** section.

Creating an Amazon Lookout for Vision model

An Amazon Lookout for Vision model is a machine learning model that predicts the presence of anomalies in new images by finding patterns in images used to train the model. This section shows you how to create and train a model. After you train your model, you evaluate its performance. For more information, see Improving your model (p. 40).

Before you create your first model, we recommend that you read Getting Started with the Amazon Lookout for Vision console (p. 10). If you are using the AWS SDK, read Getting started with the AWS SDK (p. 13).

Topics

Creating your project

An Amazon Lookout for Vision project is a grouping of the resources needed to create and manage a Lookout for Vision model. A project manages the following:

- **Dataset** – The images and image labels used to train a model. For more information, see Creating your dataset (p. 24).
- **Model** – The software that you train to detect anomalies. You can have multiple versions of a model. For more information, see Training your model (p. 35).

We recommend that you use a project for a single use case, such as detecting anomalies in a single type of machine part.

> **Note**
> You can use AWS CloudFormation to provision and configure Amazon Lookout for Vision projects. For more information, see Creating Amazon Lookout for Vision projects with AWS CloudFormation (p. 69).

To view your projects, see Viewing your projects (p. 64) or open the Using the Amazon Lookout for Vision dashboard (p. 20). To delete a model, see Deleting a model (p. 76).

Topics

Creating a project (console)

The following procedure shows you how to create a project using the console.

To create a project (console)

1. Open the Amazon Lookout for Vision console at https://console.aws.amazon.com/lookoutvision/.
2. In the left navigation pane, choose **Projects**.
3. Choose **Create project**.
4. In **Project name**, enter a name for your project.
5. Choose **Create project**. The details page for your project is displayed.
6. Follow the steps in Creating your dataset (p. 24) to create your dataset.

Creating a project (SDK)

You use the CreateProject operation to create an Amazon Lookout for Vision project. The response from `CreateProject` includes the project name and the Amazon Resource Name (ARN) of the project. Afterwards, call CreateDataset to add a training and a test dataset to your project. For more information, see Creating a dataset with a manifest file (SDK) (p. 31).

To view the projects that you have created in a project, call `ListProjects`. For more information, see Viewing your projects (p. 64).

To create a project (SDK)

1. If you haven't already done so, do the following:
   a. Create or update an IAM user with permissions to access Amazon Lookout for Vision. For more information, see Step 3: Set up permissions (p. 3).
   b. Install and configure the AWS CLI and the AWS SDKs. For more information, see Step 5: Set up the AWS CLI and AWS SDKs (p. 5).
2. Use the following example code to create a model.

   CLI

   Change the value of `project-name` to the name that you want to use for the project.

```
aws lookoutvision create-project --project-name project name
```

   Python

   In the function `main`, change the `project_name` to the name that you want to use for the project.

```
# Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
# SPDX-License-Identifier: Apache-2.0

import boto3

from botocore.exceptions import ClientError

def create_project(project_name):
    """
    Creates a new Amazon Lookout for Vision project.
    param: project_name: The name for the new project.
    """

    try:

        client = boto3.client("lookoutvision")
```

```
        # Create a project
        print("Creating project:" + project_name)
        response = client.create_project(ProjectName=project_name)
        print("project ARN: " + response["ProjectMetadata"]["ProjectArn"])
        print("Done...")

    except ClientError as err:
        print("Service error: " + format(err))
        raise


def main():

    project_name = "my-project"  # Change to the desired name.
    create_project(project_name)


if __name__ == "__main__":
    main()
```

3. Follow the steps in Creating a dataset using an Amazon SageMaker Ground Truth manifest file (p. 28) to create your dataset.

Preparing images for a dataset

You need a collection of images to create a dataset. Your images must be PNG or JPEG format files.

If you are using a single dataset, you need at least 20 images of normal objects and at least 10 images of anomalous objects.

If you are using separate training and test datasets, your training dataset needs a minimum of 10 images of normal objects. Your test dataset needs at least 10 images of normal objects and at least 10 images of anomalous objects. To create a higher quality model, use more than the minimum number of images.

Your images should be of a single type of object. Also, you should have consistent image capture conditions, such as camera positioning, lighting, and object pose.

All images in a project must have the same dimensions.

All training and test images must be unique images, preferably of unique objects. Normal images should capture the normal variations of the object being analyzed. Anomalous images should capture a diverse sampling of anomalies.

Amazon Lookout for Vision provides example images that you can use. For more information, see the section called “Step 8: (Optional) Prepare example images” (p. 9).

For image limits, see *Quotas* (p. 128).

Creating your dataset

A dataset contain the images and assigned labels that you use to train and test a model. An Amazon Lookout for Vision project can have one of the following dataset configurations:

- **Single dataset** – A single dataset project uses a single dataset to train and test your model. Using a single dataset simplifies training by letting Amazon Lookout for Vision choose the training and test

images. During training, Amazon Lookout for Vision, internally splits the dataset into a training dataset and a test dataset. You don't have access to the split datasets. We recommend using a single dataset project for most scenarios.

- **Separate training and test datasets** – If you want finer control over training, testing, and performance tuning, you can configure your project to have separate training and test datasets. Use a separate test dataset if you want control over the images used for testing, or if you already have a benchmark set of images that you want to use.

You can add a test dataset to an existing single dataset project. The single dataset then becomes the training dataset. If you remove the test dataset from a project with separate training and test datasets, the project becomes a single dataset project. For more information, see Deleting a dataset (p. 72).

To view the existing models in a project, see Viewing your datasets (p. 70).

For information about dataset image requirements, see Preparing images for a dataset (p. 24).

You can create your project's datasets by using images in an Amazon S3 bucket, your local computer, or from an Amazon SageMaker Ground Truth format manifest file.

Topics

Creating a dataset using images stored on your local computer

You can create a dataset by using images that are loaded directly from your computer. You can upload up to 30 images at a time. In this procedure, you can create a single dataset project, or a project with separate training and test datasets.

> **Note**
> If you've just completed Creating your project (p. 22), the console should show your project dashboard and you don't need to do steps 1 - 3.

To create a dataset using images on a local computer (console)

1. Open the Amazon Lookout for Vision console at https://console.aws.amazon.com/lookoutvision/.
2. In the left navigation pane, choose **Projects**.
3. In the **Projects** page, choose the project to which you want to add a dataset.
4. On the project details page, choose **Create dataset**.
5. Choose the **Single dataset** tab or the **Separate training and test datasets** tab and follow the steps.

   Single dataset

   a. In the **Dataset configuration** section, choose **Create a single dataset**.
   b. In the **Image source configuration** section, choose **Upload images from your computer**.
   c. Choose **Create dataset**.
   d. On the dataset page, choose **Add images**.
   e. Choose the images you want to upload into the dataset from your computer files. You can drag the images or choose the images that you want to upload from your local computer.

f. Choose **Upload images**.

Separate training and test datasets

a. In the **Dataset configuration** section, choose **Create a training dataset and a test dataset**.
b. In the **Training dataset details** section, choose **Upload images from your computer**.
c. In the **Test dataset details** section, choose **Upload images from your computer**.

Note
Your training and test datasets can have different image sources.

d. Choose **Create dataset**. A dataset page appears with a **Training** tab and a **Test** tab for the respective datasets.
e. Choose **Actions** and then choose **Add images to training dataset**.
f. Choose the images you want to upload to the dataset. You can drag the images or choose the images that you want to upload from your local computer.
g. Choose **Upload images**.
h. Repeat steps 5e - 5g. For step 5e, choose **Actions** and then choose **Add images to test dataset**.

6. Follow the steps in Labeling your images (p. 34) to label your images.
7. Follow the steps in Training your model (p. 35) to train your model.

Creating a dataset using images stored in an Amazon S3 bucket

You can create a dataset using images stored in an Amazon S3 bucket. With this option, you can use the folder structure in your Amazon S3 bucket to automatically label your images. You can store the images in the console bucket or another Amazon S3 bucket in your account.

Setting up folders for automatic labeling

During dataset creation, you can choose to assign label names to images based on the name of the folder that contains the images. The folders must be a child of the Amazon S3 folder path that you specify in **S3 URI** when you create the dataset.

The following is the `train` folder for the Getting Started example images. If you specify the Amazon S3 folder location as `S3-bucket/circuitboard/train/`, the images in the folder `normal` are assigned the label `Normal`. Images in the folder `anomaly` are assigned the label `Anomaly`. The names of deeper child folders aren't used to label images.

```
S3-bucket
  ### circuitboard
    ### train
        ### anomaly
            ### train-anomaly_1.jpg
            ### train-anomaly_2.jpg
            ### .
            ### .
        ### normal
            ### train-normal_1.jpg
            ### train-normal_2.jpg
            ### .
            ### .
```

Creating a dataset using images from an Amazon S3 bucket

The following procedure creates a dataset using the Getting Started example images stored in an Amazon S3 bucket. To use your own images, create the folder structure described in Setting up folders for automatic labeling (p. 26).

The procedure also shows how to create a single dataset project, or a project that uses separate training and test datasets.

If you don't choose to automatically label your images, you need to label the images after the datasets is created. For more information, see Labeling your images (p. 34).

Note
If you've just completed Creating your project (p. 22), the console should show your project dashboard and you don't need to do steps 1 - 4.

To create a dataset using images stored in an Amazon S3 bucket

1. If you haven't already done so, upload the getting started images to your Amazon S3 bucket. For more information, see Step 8: (Optional) Prepare example images (p. 9).
2. Open the Amazon Lookout for Vision console at https://console.aws.amazon.com/lookoutvision/.
3. In the left navigation pane, choose **Projects**.
4. In the **Projects** page, choose the project to which you want to add a dataset. The details page for your project is displayed.
5. Choose **Create dataset**. The **Create dataset** page is shown.

   Tip
   If you're following the Getting Started instructions, choose **Create a training dataset and a test dataset**.
6. Choose the **Single dataset** tab or the **Separate training and test datasets** tab and follow the steps.

   Single dataset

   a. In the **Dataset configuration** section, choose **Create a single dataset**.
   b. Enter the information for steps 7 - 9 in the **Image source configuration** section.

   Separate training and test datasets

   a. In the **Dataset configuration** section, choose **Create a training dataset and a test dataset**.
   b. For your training dataset, enter the information for steps 7 - 9 in the **Training dataset details** section.
   c. For your test dataset, enter the information for steps 7 - 9 in the **Test dataset details** section.

      Note
      Your training and test datasets can have different image sources.
7. Choose **Import images from Amazon S3 bucket**.
8. In **S3 URI**, enter the Amazon S3 bucket location and folder path. Change `bucket` to the name of your Amazon S3 bucket.

   a. If you're creating a single dataset project or a training dataset, enter the following:

   ```
   s3://bucket/circuitboard/train/
   ```

   b. If you're creating a test dataset enter the following:

```
s3://bucket/circuitboard/test/
```

9. Choose **Automatically attach labels to images based on the folder**.
10. Choose **Create dataset**. A dataset page opens with your labeled images.
11. Follow the steps in Training your model (p. 35) to train your model.

Creating a dataset using an Amazon SageMaker Ground Truth manifest file

A manifest file contains information about the images and image labels that you can use to train and test a model. You can store a manifest file in an Amazon S3 bucket and use it to create a dataset. You can create your own manifest file or you can use an existing manifest file, such as the output from an Amazon SageMaker Ground Truth job.

Topics

- Creating a manifest file (p. 28)
- Creating a dataset with a manifest file (console) (p. 30)
- Creating a dataset with a manifest file (SDK) (p. 31)

Creating a manifest file

You can create a dataset by importing an SageMaker Ground Truth format manifest file. If your images are labeled in a format that isn't a SageMaker Ground Truth manifest file, use the following information to create an SageMaker Ground Truth format manifest file.

Manifest files are in JSON lines format where each line is a complete JSON object representing the labeling information for an image. Amazon Lookout for Vision supports SageMaker Ground Truth manifests with JSON lines in Classification Job Output format. The `class-name` field determines whether the contents of an image are normal or anomalous. For more information, see Defining JSON lines for anomaly classification (p. 28).

> **Note**
> The JSON line examples in this section are formatted for readability.

The images referenced by a manifest file must be located in the same Amazon S3 bucket. The manifest file can be in a different bucket. You specify the location of an image in the `source-ref` field of a JSON line.

You can create a manifest file by using code. The Amazon Lookout for Vision Lab Python Notebook shows how to create the manifest file for the circuitboard example images. Alternatively, you can use the Datasets example code in the AWS Code Examples Repository.

Topics

- Defining JSON lines for anomaly classification (p. 28)

Defining JSON lines for anomaly classification

You define a JSON line for each image that you want to use in an Amazon Lookout for Vision manifest file. The JSON line determines whether the image contains normal or anomalous content. A JSON line is in SageMaker Ground Truth Classification Job Output format. A manifest file is made of one or more JSON lines, one for each image that you want to import.

To create a manifest file for image-level labels

1. Create an empty text file.
2. Add a JSON line for each image the that you want to import. Each JSON line should look similar to the following:

```
{"source-ref":"s3://lookoutvision-console-us-east-1-nnnnnnnnnn/gt-job/manifest/
IMG_1133.png","anomaly-label":1,"anomaly-label-metadata":{"confidence":0.95,"job-
name":"labeling-job/testclconsolebucket","class-name":"normal","human-
annotated":"yes","creation-date":"2020-04-15T20:17:23.433061","type":"groundtruth/
image-classification"}}
```

3. Save the file.

   Note
   You can use the extension `.manifest`, but it is not required.

4. Create a dataset using the manifest file that you created. For more information, see Creating a manifest file (p. 28).

Image-level JSON lines

In this section, you learn how to create a JSON line for an image that is either normal or anomalous.

Anomaly JSON line

The following JSON line shows an image that is labeled as an anomaly. Note that the value of `class-name` is `anomaly`.

```
{
    "source-ref": "s3: //bucket/image/anomaly/abnormal-1.jpg",
    "anomaly-label-metadata": {
        "confidence": 1,
        "job-name": "labeling-job/auto-label",
        "class-name": "anomaly",
        "human-annotated": "yes",
        "creation-date": "2020-11-10T03:37:09.600",
        "type": "groundtruth/image-classification"
    },
    "anomaly-label": 1
}
```

Normal JSON line

The following JSON line shows an image labeled as normal. Note that the value of `class-name` is `normal`.

```
{
    "source-ref": "s3: //bucket/image/normal/2020-10-20_12-14-55_613.jpeg",
    "anomaly-label-metadata": {
        "confidence": 1,
        "job-name": "labeling-job/auto-label",
        "class-name": "normal",
        "human-annotated": "yes",
        "creation-date": "2020-11-10T03:37:09.603",
        "type": "groundtruth/image-classification"
    },
    "anomaly-label": 0
```

```
}
```

JSON line keys and values

The following information describes the keys and values in an Amazon Lookout for Vision JSON line.

source-ref

(Required) The Amazon S3 location of the image. The format is `"s3://BUCKET/OBJECT_PATH"`. Images in an imported dataset must be stored in the same Amazon S3 bucket.

anomaly-label

(Required) The label attribute. Use the key `anomaly-label`, or another key name that you choose. The key value (`0` in the preceding example) is required by Amazon Lookout for Vision, but it isn't used. The output manifest created by Amazon Lookout for Vision converts the value to `0` for an anomalous image and a value of `1` for a normal image. The value of `class-name` determines if the image is normal or anomalous.

There must be corresponding metadata identified by the field name with *-metadata* appended. For example, `"anomaly-label-metadata"`.

anomaly-label-metadata

(Required) Metadata about the label attribute. The field name must be the same as the label attribute with *-metadata* appended.

confidence

(Optional) Currently not used by Amazon Lookout for Vision. If you do specify a value, use a value of `1`.

job-name

(Optional) A name that you choose for the job that processes the image.

class-name

(Required) If the image contains normal content, specify `normal`, otherwise specify `anomaly`. If the value of `class-name` is any other value, the image is added to the dataset as an unlabeled image. To label an image, see Editing your dataset (p. 34).

human-annotated

(Required) Specify `"yes"`, if the annotation was completed by a human. Otherwise, specify `"no"`.

creation-date

(Optional) The Coordinated Universal Time (UTC) date and time that the label was created.

type

(Required) The type of processing that should be applied to the image. For image-level anomaly labels, the value is `"groundtruth/image-classification"`.

Creating a dataset with a manifest file (console)

The following procedure shows you how to create a training or test dataset by importing an SageMaker format manifest file that is stored in an Amazon S3 bucket.

After you create the dataset, you can add more images to the dataset, or add labels to images. For more information, see Editing your dataset (p. 34).

To create a dataset using an SageMaker Ground Truth format manifest file (console)

1. Create, or use an existing, Amazon Lookout for Vision compatible SageMaker Ground Truth format manifest file. For more information, see Creating a manifest file (p. 28).
2. Sign in to the AWS Management Console and open the Amazon S3 console at https://console.aws.amazon.com/s3/.
3. In an Amazon S3 bucket, create a folder to hold your manifest file.
4. Upload your manifest file to the folder that you just created.
5. In the Amazon S3 bucket, create a folder to store your images.
6. Upload your images to the folder you just created.

   Important
   The `source-ref` field value in each JSON line must map to images in the folder.
7. Open the Amazon Lookout for Vision console at https://console.aws.amazon.com/lookoutvision/.
8. Choose **Get started**.
9. In the left navigation pane, choose **Projects**.
10. Choose the project that you want to add to use with the manifest file.
11. In the **How it works** section, choose **Create dataset**.
12. Choose the **Single dataset** tab or the **Separate training and test datasets** tab and follow the steps.

    Single dataset

    1. Choose **Create a single dataset**.
    2. In the **Image source configuration** section, choose **Import images labeled by SageMaker Ground Truth**.
    3. For **.manifest file location**, enter the location of your manifest file.

    Separate training and test datasets

    1. Choose **Create a training dataset and a test dataset**.
    2. In the **Training dataset details** section, choose **Import images labeled by SageMaker Ground Truth**.
    3. In **.manifest file location**, enter the location of your training manifest file.
    4. In the **Test dataset details** section, choose **Import images labeled by SageMaker Ground Truth**.
    5. In **.manifest file location**, enter the location of your test manifest file.

       Note
       Your training and test datasets can have different image sources.
13. Choose **Submit**.

Amazon Lookout for Vision creates a dataset in the Amazon S3 bucket `datasets` folder. Your original `.manifest` file remains unchanged.

Creating a dataset with a manifest file (SDK)

You use the CreateDataset operation to create the datasets associated with an Amazon Lookout for Vision project.

If you want to use a single dataset for training and testing, create a single dataset with the `DatasetType` value set to `train`. During training, the dataset is internally split to make a training and test dataset. You don't have access to the split training and test datasets. If you want a separate test

dataset, make a second call to `CreateDataset` with the `DatasetType` value set `test`. During training, the training and test datasets are used to train and test the model.

You can optionally use the `DatasetSource` parameter to specify the location of a SageMaker Ground Truth format manifest file that's used to populate the dataset. In this case, the call to `CreateDataset` is asynchronous. To check the current status, call `DescribeDataset`. For more information, see Viewing your datasets (p. 70). If a validation error occurs during import, the value of `Status` is set to CREATE_FAILED and the status message (`StatusMessage`) is set.

If you don't specify `DatasetSource`, an empty dataset is created. In this case, the call to `CreateDataset` is synchronous. Later, you can labeled images to the dataset by calling UpdateDatasetEntries.

If you want to replace a dataset, first delete the existing dataset with DeleteDataset and then create a new dataset of the same dataset type by calling `CreateDataset`. For more information, see Deleting a dataset (p. 72).

After you create the datasets, you can create the model. For more information, see Training a model (SDK) (p. 37).

You can view the labeled images (JSON lines) within a dataset by calling ListDatasetEntries. You can add labeled images by calling `UpdateDatasetEntries`.

To view information about the test and training datasets, see Viewing your datasets (p. 70).

To create a dataset (SDK)

1. If you haven't already done so, do the following:
   a. Create or update an IAM user with permissions to access Amazon Lookout for Vision. For more information, see Step 3: Set up permissions (p. 3).
   b. Install and configure the AWS CLI and the AWS SDKs. For more information, see Step 5: Set up the AWS CLI and AWS SDKs (p. 5).
2. Use the following example code to create a model.

   CLI

   Change the following values:

   - `project-name` to the name of the project that you want to associate the dataset with.
   - `dataset-type` to the type of dataset that you want to create (`train` or `test`).
   - `dataset-source` to the Amazon S3 location of the manifest file.
   - `Bucket` to the name of the Amazon S3 bucket that contains the manifest file.
   - `Key` to the path and file name of the manifest file in the Amazon S3 bucket.

```
aws lookoutvision create-dataset --project-name project\
  --dataset-type train or test\
  --dataset-source '{ "GroundTruthManifest": { "S3Object": { "Bucket": "bucket",
 "Key": "manifest file" } } }'
```

   Python

   In the function `main`, change the following values:

   - `project_name` to the name of the project that you want to associate the dataset with.
   - `dataset_type` to the type of dataset that you want to create (`train` or `test`).
   - `bucket` to the name of the Amazon S3 bucket that contains the manifest file.

- `manifest_file` to the path and file name of the manifest file within `bucket`.

```
import json
import time
import boto3

from botocore.exceptions import ClientError

def create_dataset(project_name, bucket, manifest_file, dataset_type):
    """
    Creates a new Amazon Lookout for Vision dataset
    param: project_name: The name of the project in which you want to create a
 dataset.
    param: bucket: The bucket that contains the manifest file.
    param: manifest_file: The path and name of the manifest file.
    param: dataset_type: The type of the dataset (train or test).
    """

    try:

        client = boto3.client("lookoutvision")

        # Create a dataset
        print("Creating dataset...")
        dataset = json.loads(
            '{ "GroundTruthManifest": { "S3Object": { "Bucket": "'
            + bucket
            + '", "Key": "'
            + manifest_file
            + '" } } }'
        )

        response = client.create_dataset(
            ProjectName=project_name, DatasetType=dataset_type,
 DatasetSource=dataset
        )
        print("Dataset Status: " + response["DatasetMetadata"]["Status"])
        print("Dataset Status Message: " + response["DatasetMetadata"]
["StatusMessage"])
        print("Dataset Type: " + response["DatasetMetadata"]["DatasetType"])

        # Wait until either created or failed.
        while True:

            dataset_description = client.describe_dataset(
                ProjectName=project_name, DatasetType=dataset_type
            )
            status = dataset_description["DatasetDescription"]["Status"]

            if status == "CREATE_IN_PROGRESS":
                print("Dataset creation in progress...")
                time.sleep(2)
                continue

            if status == "CREATE_COMPLETE":
                print("Dataset created.")
                break

            if status == "CREATE_FAILED":
                print("Dataset creation failed.")
                break
```

```
            print("Failed. Unexpected state for dataset creation: " + status)
            break

        print("Done...")

    except ClientError as err:
        print("Service error: " + format(err))
        raise


def main():
    project_name = "my-project"  # Change to your project name.
    bucket = "bucket"  # Change to the bucket with the manifest file.
    manifest_file = "input.manifest"  # Change to your manifest file.
    dataset_type = "train"  # or 'test' to create the test dataset.

    create_dataset(project_name, bucket, manifest_file, dataset_type)


if __name__ == "__main__":
    main()
```

3. Repeat the previous step and provide values for the other dataset type.

Adding an image to a dataset

To add labeled images to a dataset, use the UpdateDatasetEntries operation.

Editing your dataset

After you create a dataset, you should make sure the images are correctly labeled. You can also enhance the quality of your model by adding more images. If you have created a test dataset, adding more images can increase the accuracy of your model's performance metrics.

> **Note**
> If you're following the Getting Started instructions, you don't need to edit your training and test datasets before creating your first model.

Train your model after editing your datasets.

Topics

Labeling your images

You use the Lookout for Vision console to label images in a dataset as **normal** or **anomaly**. This prepares the dataset for training so that the model can learn to recognize anomalies in new images. Unlabeled images aren't used to train your model.

> **Note**
> If you've just completed Creating your dataset (p. 24), the console should currently show your model dashboard and you don't need to do steps 1 - 4.

To label your images (console)

1. Open the Amazon Lookout for Vision console at https://console.aws.amazon.com/lookoutvision/.
2. In the left navigation pane, choose **Projects**.

3. In the **Projects** page, choose the project that you want to use.
4. In the left navigation pane of you project dashboard, choose **Dataset**.
5. If you have separate training and test datasets, choose the tab for the dataset that you want to use.
6. Choose **Add labels**.
7. Choose **Select all images on this page**.
8. If the images are normal, choose **Classify as normal**, otherwise choose **Classify as abnormal**. A label appears underneath each picture.
9. If you need to change the label for an image, do the following:

    a. Choose **Anomaly** or **Normal** under the image.
    b. If you can't determine the correct label for an image, magnify the image by choosing the image in the gallery.

    Note
    You can filter image labels by choosing the desired label, or label state, in the **Filters** section.
10. Repeat steps 7-9 on each page as necessary until all the images in the dataset have been labeled correctly.
11. Choose **Save changes**.

Adding more images

You can add more images to your datasets by uploading images from your local computer. To add more labeled images with the SDK, use the UpdateDatasetEntries operation.

To add more images to your dataset (console)

1. Choose **Actions** and select the dataset that you want to add images to.
2. Choose the images you want to upload to the dataset. You can drag the images or choose the images that you want to upload from your local computer. You can upload up to 30 images at a time.
3. Choose **Upload images**.
4. Choose **Save changes**.

When you are done adding more images, you need to label them so that they can be used to train the model. For more information, see Labeling your images (p. 34).

Training your model

After you have created your datasets and labeled the images, you can train your model. As part of the training process, a test dataset is used. If you have a single dataset project, the images in the dataset are automatically split into a test dataset and a training dataset as part of the training process. If your project has a training and a test dataset, they are used to separately train and test the dataset.

After training is complete, you can evaluate the performance of the model and make any necessary improvements. For more information, see Improving your model (p. 40).

To train your model, Amazon Lookout for Vision makes a copy of your source training and test images. By default the copied images are encrypted with a key that AWS owns and manages. You can also choose to use your own AWS Key Management Service (KMS) key. For more information, see AWS Key Management Service concepts. Your source images are unaffected.

You can assign metadata to your model in the form of tags. For more information, see Tagging models (p. 78).

Each time you train a model, a new version of the model is created. If you no longer need a version of a model, you can delete it. For more information, see Deleting a model (p. 76).

You are charged for the amount of time it takes to successfully train your model. For more information, see Training Hours.

To view the existing models in a project, Viewing your models (p. 74).

> **Note**
> If you've just completed Creating your dataset (p. 24) or Editing your dataset (p. 34). The console should currently show your model dashboard and you don't need to do steps 1 - 4.

Topics

Training a model (console)

The following procedure shows you how to train your model using the console.

To train your model (console)

1. Open the Amazon Lookout for Vision console at https://console.aws.amazon.com/lookoutvision/.
2. In the left navigation pane, choose **Projects**.
3. In the **Projects** page, choose the project that contains the model that you want to train.
4. On the project details page, choose **Train model**. The **Train model** button is available if you have enough labeled images to train the model. If the button isn't available, add more images (p. 35) until you have enough labeled images.
5. (Optional) If you want to use your own AWS KMS encryption key, do the following:
   a. In **Image data encryption** choose **Customize encryption settings (advanced)**.
   b. In **encryption.aws_kms_key** enter the Amazon Resource Name (ARN) of your key, or choose an existing AWS KMS key. To create a new key, choose **Create an AWS IMS key**.
6. (Optional) if you want to add tags to your model do the following:
   a. In the **Tags** section, choose **Add new tag**.
   b. Enter the following:
      i. The name of the key in **Key**.
      ii. The value of the key in **Value**.
   c. To add more tags, repeat steps 6a and 6b.
   d. (Optional) If you want to remove a tag, choose **Remove** next to the tag that you want to remove. If you are removing a previously saved tag, it is removed when you save your changes.
7. Choose **Train model**.
8. In the **Do you want to train your model?** dialog box, choose **Train model**.
9. In the **Models** view, you can see that training has started and you can check the current status by viewing the `Status` column for the model version. Training a model takes a while to complete.
10. When training is finished, you can evaluate its performance. For more information, see Improving your model (p. 40).

Training a model (SDK)

You use the CreateModel operation to start the training, testing and evaluation of a model. Amazon Lookout for Vision trains the model using the training and test dataset associated with the project. For more information, see Creating a project (SDK) (p. 23).

Each time you call `CreateModel`, a new version of the model is created. The response from `CreateModel` includes the version of the model.

You are charged for each successful model training. Use the `ClientToken` input parameter to help prevent charges due to unnecessary or accidental repeats of model training by your users. `ClientToken` is an idempotent input parameter that ensures `CreateModel` only completes once for a specific set of parameters — A repeat call to `CreateModel` with the same `ClientToken` value ensures that training isn't repeated. If you don't supply a value for `ClientToken`, the AWS SDK you are using inserts a value for you. This prevents retries after a network error from starting multiple training jobs, but you'll need to provide your own value for your own use cases. For more information, see CreateModel.

Training takes a while to complete. To check the current status, call `DescribeModel` and pass the project name (specified in the call to `CreateProject`) and the model version. The `status` field indicates the current status of the model training. For example code, see Viewing your models (SDK) (p. 74).

If training is successful, you can evaluate model. For more information, see Improving your model (p. 40).

To view the models that you have created in a project, call `ListModels`. For example code, see Viewing your models (p. 74).

To train a model (SDK)

1. If you haven't already done so, do the following:
   a. Create or update an IAM user with permissions to access Amazon Lookout for Vision. For more information, see Step 3: Set up permissions (p. 3).
   b. Install and configure the AWS CLI and the AWS SDKs. For more information, see Step 5: Set up the AWS CLI and AWS SDKs (p. 5).
2. Use the following example code to train a model.

   CLI

   Change the following values:

   - `project-name` to the name of the project that contains the model that you want to create.
   - `output-config` to the location where you want to save training results. Replace the following values:
     - `output bucket` with the name of the Amazon S3 bucket where Amazon Lookout for Vision saves the training results.
     - `output folder` with the name of the folder where you want to save the training results.
     - `Key` with the name of a tag key.
     - `Value` with a value to associate with `tag_key`.

```
aws lookoutvision create-model --project-name "project name"\
  --output-config '{ "S3Location": { "Bucket": "output bucket", "Prefix":  "output
 folder" } }'\
  --tags '[{"Key":"Key","Value":"Value"}]'
```

Python

In the function `main`, change the following values:

- `project-name` to the name of the project that contains the model that you want to create.
- `output-config` to the location where you want to save training results. Replace the following values:
  - `output_bucket` with the name of the Amazon S3 bucket where Amazon Lookout for Vision saves the training results.
  - `output_folder` with the name of the folder where you want to save the training results.
  - `tag_key` with the name of a tag key.
  - `tag_key_value` with a value to associate with `tag_key`.
  - `client_token` with an idempotent token value that you choose.

```
# Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
# SPDX-License-Identifier: Apache-2.0

import json
import time
import boto3

from botocore.exceptions import ClientError


def create_model(project_name, output_bucket, output_folder, tag_key,
 tag_key_value, client_token):
    """
    Creates a version of an Amazon Lookout for Vision model.
    param: project_name: The name of the project in which you want to create a
 model.
    param: output_bucket: The output bucket in which to place training results.
    param: output_folder: The output folder in which to place training results.
    param: tag_key: The key for a tag to add to the model.
    param: tag_key_value: A value associated with the tag_key.
    param: client_token: Idempotent token value. Valid for 8 hours.
    """

    client = boto3.client("lookoutvision")

    try:
        # Create a model
        print("Creating model...")

        output_config = json.loads(
            '{ "S3Location": { "Bucket": "'
            + output_bucket
            + '", "Prefix": "'
            + output_folder
            + '" } } '
        )
        tags = json.loads(
            '[{"Key": "' + tag_key + '" ,"Value":"' + tag_key_value + '"}]'
        )

        response = client.create_model(
            ProjectName=project_name,
            OutputConfig=output_config,
            Tags=tags,
            ClientToken=client_token,
        )
```

```
        print("ARN: " + response["ModelMetadata"]["ModelArn"])
        print("Version: " + response["ModelMetadata"]["ModelVersion"])
        print("Started training...")

        while True:
            model_description = client.describe_model(
                ProjectName=project_name,
                ModelVersion=response["ModelMetadata"]["ModelVersion"],
            )
            status = model_description["ModelDescription"]["Status"]

            if status == "TRAINING":
                print("Model training in progress...")
                time.sleep(600)
                continue

            if status == "TRAINED":
                print("Model was successfully trained.")
                break

            if status == "TRAINING_FAILED":
                print(
                    "Model training failed: "
                    + model_description["ModelDescription"]["StatusMessage"]
                )
                break

            print("Failed. Unexpected state for training: " + status)
            break

        print("Done...")

    except ClientError as err:
        print("Service error: " + format(err))
        raise


def main():
    project_name = "my-project"  # the project in which to create a model.
    output_bucket = "output-bucket"  # bucket to store training output.
    output_folder = "my-project-output-folder/"  # folder for training output.
    tag_key = "my-tag-key"  # tag key.
    tag_key_value = "my-tag-key-value"  # tag value.
    client_token = "1"  # idempotency token.

    create_model(
        project_name, output_bucket, output_folder, tag_key, tag_key_value,
 client_token
    )


if __name__ == "__main__":
    main()
```

3. When training is finished, you can evaluate its performance. For more information, see Improving your model (p. 40).

Improving your model

During training Lookout for Vision tests your model with the test dataset and uses the results to create performance metrics. You can use performance metrics to evaluate the performance of your model. If necessary, you can take steps to improve your datasets and then retrain your model.

If you're satisfied with the performance of your model, you can begin to use it. For more information, see Running your trained Amazon Lookout for Vision model (p. 48).

Topics

Step 1: Evaluate the performance of your model

You can access the performance metrics from the console and from the `DescribeModel` operation. Amazon Lookout for Vision provides summary performance metrics for the test dataset and the predicted results for all individual images.

To view the performance metrics and test image predictions in the console, see Viewing performance metrics (console) (p. 43). For information about accessing performance metrics and test image predictions with the `DescribeModel` operation, see Viewing performance metrics (SDK) (p. 43).

Amazon Lookout for Vision provides the following summary performance metrics:

- Precision (p. 40)
- Recall (p. 41)
- F1 score (p. 41)

During testing, the model predicts a result for each test image in the test dataset. The result for each prediction is compared to the label (normal or anomaly) of the corresponding test image as follows:

- Correctly predicting that an image is anomalous is considered a *true positive*.
- Incorrectly predicting that an image is anomalous is considered a *false positive*.
- Correctly predicting that an image is normal is considered a *true negative*.
- Incorrectly predicting that an image is normal is considered a *false negative*.

Amazon Lookout for Vision uses the results of the comparisons to generate the performance metrics.

Precision

The precision metric answers the question – *When the model predicts that an image contains an anomaly, how often is that prediction correct?*

Precision is a useful metric for situations where the cost of a false positive is high. For example, the cost of removing a machine part that is not defective from an assembled machine.

Amazon Lookout for Vision provides a summary precision metric value for the entire test dataset.

Precision is the fraction of correctly predicted anomalies (true positives) over all predicted anomalies (true and false positives). The formula for precision is as follows.

Precision value = true positives / (true positives + false positives)

The possible values for precision range from 0–1. The Amazon Lookout for Vision console displays precision as a percentage value (0–100).

A higher precision value indicates that more of the predicted anomalies are correct. For example, suppose your model predicts that 100 images are anomalous. If 85 of the predictions are correct (the true positives) and 15 are incorrect (the false positives), the precision is calculated as follows:

85 *true positives* / (85 *true positives* + 15 *false positives*) = **0.85** precision value

However, if the model only predicts 40 images correctly out of 100 anomaly predictions, the resulting precision value is lower at **0.40** (that is, 40 / (40 + 60) = 0.40). In this case, your model is making more incorrect predictions than correct predictions. To fix this, consider making improvements to your model. For more information, see Step 2: Improve your model (p. 42).

For more information, see Precision and recall.

Recall

The recall metric answers the question - *Of the total number of anomalous images in the test dataset, how many are correctly predicted as anomalous?*

The recall metric is useful for situations where the cost of a false negative is high. For example, when the cost of not removing a defective part is high. Amazon Lookout for Vision provides a summary recall metric value for the entire test dataset.

Recall is the fraction of the anomalous test images that were detected correctly. It is a measure of how often the model can correctly predict an anomalous image, when it's actually present in the images of your test dataset. The formula for recall is calculated as follows:

Recall value = true positives / (true positives + false negatives)

The range for recall is 0–1. The Amazon Lookout for Vision console displays recall as a percentage value (0–100).

A higher recall value indicates that more of the anomalous images are correctly identified. For example, suppose the test dataset contains 100 anomalous images. If the model correctly detects 90 of the 100 anomalous images, then the recall is as follows:

90 *true positives* / (90 *true positives* + 10 *false negatives*) = **0.90** recall value

A recall value of 0.90 indicates that your model is correctly predicting most of the anomalous images in the test dataset. If the model only predicts 20 of the anomalous images correctly, the recall is lower at **0.20** (that is, 20 / (20 + 80) = 0.20).

In this case, you should consider making improvements to your model. For more information, see Step 2: Improve your model (p. 42).

For more information, see Precision and recall.

F1 score

Amazon Lookout for Vision provides an average model performance score for the test dataset. Specifically, model performance for anomaly classification is measured by the F1 score metric, which is the harmonic mean of the precision and recall scores.

F1 score is an aggregate measure that takes into account both precision and recall. The model performance score is a value between 0 and 1. The higher the value, the better the model is performing for both recall and precision. For example, for a model with precision of 0.9 and a recall of 1.0, the F1 score is 0.947.

If the model isn't performing well, for example, with a low precision of 0.30 and a high recall of 1.0, the F1 score is 0.46. Similarly, if the precision is high (0.95) and the recall is low (0.20), the F1 score is 0.33. In both cases, the F1 score is low, which indicates problems with the model.

For more information, see F1 score.

Step 2: Improve your model

The performance metrics might show that you can improve your model. For example, if the model doesn't detect all anomalies in the test dataset, your model has low recall (that is, the recall metric has a low value). If you need to improve your model, consider the following:

- Check that the training and test dataset images are properly labeled.
- Reduce the variability of image capture conditions such as lighting and object pose, and train your model on objects of the same type.
- Ensure that your images show only the required content. For example, if your project detects anomalies in machine parts, make sure that no other objects are in your images.
- Add more labeled images to your train and test datasets. If your test dataset has excellent recall and precision but the model performs poorly when deployed, your test dataset might not be representative enough and you need to extend it.
- If your test dataset results in poor recall and precision, you might need to add more training images. For more information, see Adding more images (p. 35). An alternative way to add labeled images to your training dataset is to run a trial detection task and verify the results. You can then add the verified images to the training dataset. For more information, see Verifying your model with a trial detection task (p. 45).
- Ensure you have sufficiently diverse normal and anomalous images in your training and test dataset. The images must represent the type of normal and anomalous images that your model will encounter. For example, when analyzing circuit boards, your normal images should represent the variations in position and soldering of components, such as resistors and transistors. The anomalous images should represent the different types of anomalies that the system might encounter, such as misplaced or missing components.

After you have improved your training and test dataset, retrain and re-evaluate your model. For more information, see Training your model (p. 35).

If the metrics show that you model has acceptable performance, you can verify its performance by adding the results of a trial detection task to the test dataset. After retraining, the performance metrics should confirm the performance metrics from the previous training. For more information, see Verifying your model with a trial detection task (p. 45).

Viewing performance metrics

You can get performance metrics from the console and by calling the `DescribeModel` operation. After training is complete, the console displays the performance metrics. The following procedure shows how to get the performance metrics from a project's model list view.

The Amazon Lookout for Vision console shows the following performance metrics:

- Precision (p. 40)
- Recall (p. 41)
- F1 score (p. 41)

The test results overview section shows you the total correct and incorrect predictions for images in the test dataset. You can also see the predicted and actual label assignments for individual images in the test dataset.

Topics

- Viewing performance metrics (console) (p. 43)
- Viewing performance metrics (SDK) (p. 43)

Viewing performance metrics (console)

To view performance metrics (console)

1. Open the Amazon Lookout for Vision console at https://console.aws.amazon.com/lookoutvision/.
2. Choose **Get started**.
3. In the left navigation pane, choose **Projects**.
4. In the projects view, choose the project that contains the version of the model that you want to view.
5. In the left navigation pane, under the project name, choose **Models**.
6. In the models list view, choose the versions of the model that you want to view.
7. On the model details page, view the performance metrics on the **Performance metrics** tab.

Viewing performance metrics (SDK)

You can use the DescribeModel operation to get the summary performance metrics, the evaluation manifest, and the evaluation results for a model.

Getting the summary performance metrics

The summary performance metrics (Precision (p. 40), Recall (p. 41), and F1 score (p. 41)) are returned in the `Performance` field returned by a call to `DescribeModel`.

```
        "Performance": {
            "F1Score": 0.8,
            "Recall": 0.8,
            "Precision": 0.9
        },
```

For information about summary performance metrics see Step 1: Evaluate the performance of your model (p. 40). For example code, see Viewing your models (SDK) (p. 74).

Using the evaluation manifest

The evaluation manifest provides test prediction metrics for the individual images used to test a model. For each image in the test dataset, a JSON line contains the original test (ground truth) information and the model's prediction for the image. Amazon Lookout for Vision stores the evaluation manifest in an Amazon S3 bucket. You can get the location from the `EvaluationManifest` field in the response from `DescribeModel` operation.

```
"EvaluationManifest": {
            "Bucket": "lookoutvision-us-east-1-nnnnnnnnnn",
            "Key": "my-sdk-project-model-output/EvaluationManifest-my-sdk-project-1.json"
        }
```

The file name format is `EvaluationManifest-`*`project name`*`.json`. For example code, see Viewing your models (p. 74).

In the following sample JSON line, the `class-name` is the ground truth for the contents of the image. In this example the image contains an anomaly. The `confidence` field shows the confidence that Amazon Lookout for Vision has in the prediction.

```
{
    "source-ref"*: "s3://customerbucket/path/to/image.jpg",
    "source-ref-metadata": {
        "creation-date": "2020-05-22T21:33:37.201882"
    },

    // Test dataset ground truth
    "anomaly-label": 1,
    "anomaly-label-metadata": {
        "class-name": "anomaly",
        "type": "groundtruth/image-classification",
        "human-annotated": "yes",
        "creation-date": "2020-05-22T21:33:37.201882",
        "job-name": "labeling-job/anomaly-detection"
    },
    // Anomaly label detected by Lookout for Vision
    "anomaly-label-detected": 1,
    "anomaly-label-detected-metadata": {
        "class-name": "anomaly",
        "confidence": 0.9,
        "type": "groundtruth/image-classification",
        "human-annotated": "no",
        "creation-date": "2020-05-22T21:33:37.201882",
        "job-name": "training-job/anomaly-detection",
        "model-arn": "lookoutvision-some-model-arn",
        "project-name": "lookoutvision-some-project-name",
        "model-version": "lookoutvision-some-model-version"
    }
}
```

Reviewing the evaluation result

The evaluation result has the following aggregate performance metrics for the entire set of test images:

- Precision (p. 40)
- Recall (p. 41)
- ROC curve (not shown in console)
- Average precision (not shown in console)

The evaluation result also includes the number of images used for training and testing the model.

Amazon Lookout for Vision stores the evaluation result in an Amazon S3 bucket. You can get the location by checking the value of `EvaluationResult` in the response from `DescribeModel` operation.

```
"EvaluationResult": {
            "Bucket": "lookoutvision-us-east-1-nnnnnnnnnn",
```

```
            "Key": "my-sdk-project-model-output/EvaluationResult-my-sdk-project-1.json"
        }
```

The file name format is `EvaluationResult-`*`project name`*`.json`. For an example, see Viewing your models (p. 74).

The following schema shows the evaluation result.

```
{
    "Version": 1,
    "EvaluationDetails": {
        "ModelArn": "string", // The Amazon Resource Name (ARN) of the model version.
        "EvaluationEndTimestamp": "string",  // The date and time that evaluation finished.
        "NumberOfTrainingImages": "int",     // The number of images that were successfully
 used for training.
        "NumberOfTestingImages": "int",      // The number of images that were successfully
 used for testing.
    },
    "AggregatedEvaluationResults": {
        "Metrics": {
            "ROCAUC": "float",               // ROC area under the curve.
            "AveragePrecision": "float",     // The average precision of the model.
            "Precision": "float",            // The overall precision of the model.
            "Recall": "float",               // The overall recall of the model.
        }
    }

}
```

Verifying your model with a trial detection task

If you want to verify or improve the quality of your model, you can run a trial detection task. A trial detection task detects anomalies in new images that you supply.

You can verify the detection results and add the verified images to your dataset. If you have separate training and test datasets, the verified images are added to the training dataset.

You can verify images from your local computer or images located in an Amazon S3 bucket. If you want to add verified images to the dataset, images located in an S3 bucket must be in the same S3 bucket as the images in your dataset.

> **Note**
> To run a trial detection task, ensure that your S3 bucket has versioning enabled. For more information, see Using versioning. The console bucket is created with versioning enabled.

By default your images are encrypted with a key that AWS owns and manages. You can also choose to use your own AWS Key Management Service (KMS) key. For more information, see AWS Key Management Service concepts.

Topics

Running a trial detection task

Perform the following steps to run a trial detection task.

To run a trial detection (console)

1. Open the Amazon Lookout for Vision console at https://console.aws.amazon.com/lookoutvision/.
2. Choose **Get started**.
3. In the left navigation pane, choose **Projects**.
4. In the projects view, choose the project that contains the version of the model that you want to view.
5. In the left navigation pane, under the project name, choose **Trial detections**.
6. In the trial detections view, choose the **Run trial detection**.
7. On the **Run trial detection** page, enter a name for your trial detection task in **Task name**.
8. In **Choose model**, choose the version of that model that you want to use.
9. Import the images according to the source of the images as follows:
   - If you are importing your source images from an Amazon S3 bucket, enter the **S3 URI**.

     Tip
     If you're using the Getting Started example images, use the *extra_images* folder. The Amazon S3 URI is `s3://your bucket/circuitboard/extra_images`.
   - If you are uploading images from your computer, add the images after you choose **Detect anomalies**.
10. (Optional) If you want to use your own AWS KMS encryption key, do the following:
    a. For **Image data encryption**, choose **Customize encryption settings (advanced)**.
    b. In **encryption.aws_kms_key**, enter the Amazon Resource Name (ARN) of your key, or choose an existing AWS KMS key. To create a new key, choose **Create an AWS IMS key**.
11. Choose **Detect anomalies** and then choose **Run trial detection** to start the trial detection task.
12. Check the current status in the trial detections view. The trial detection might take a while to complete.

Verifying trial detection results

Verifying the results of a trial detection can help you improve your model.

If the performance metrics are poor, improve your model by running a trial detection and then add verified images to the dataset (training dataset, if you have a separate datasets).

If the model's performance metrics are good, but the results of a trial detection are poor, you can improve your model by adding verified images to the dataset (training dataset). If you have a separate test dataset, consider adding more images to the test dataset.

After you add verified images to your dataset, retrain and re-evaluate your model. For more information, see Training your model (p. 35).

To verify the results of a trial detection

1. Open the Amazon Lookout for Vision console at https://console.aws.amazon.com/lookoutvision/.
2. In the left navigation pane, choose **Projects**.
3. In the **Projects** page, choose the project that you want to use. The dashboard for your project is displayed.
4. In the left navigation pane, choose **Trial detections**.
5. Choose the trial detection that you want to verify.
6. On the trial detection page, choose **Verify machine predictions**.
7. Choose **Select all images on this page**.

8. If the predictions are correct, choose **Verify as correct**. Otherwise, choose **Verify as incorrect**. The prediction and prediction confidence score is shown under each image.
9. If you need to change the label for an image, do the following:

    a. Choose **Correct** or **Incorrect** under the image.

    b. If you can't determine the correct label for an image, magnify the image by choosing the image in the gallery.

    Note
    You can filter image labels by choosing the desired label, or label state, in the **Filters** section. You can sort by confidence score in the **Sorting options** section.
10. Repeat steps 7-9 on each page as necessary until all the images have been verified.
11. Choose **Add verified images to dataset**. If you have separate datasets, the images are added to the training dataset.
12. Retrain your model. For more information, see the section called “Training your model” (p. 35).

Running your trained Amazon Lookout for Vision model

To detect anomalies in images with your model, you must first start your model with the StartModel operation.

After your model starts, you can use the `DetectAnomalies` operation to detect anomalies in an image. For more information, see Detecting anomalies in an image (p. 55).

When you start your model, Amazon Lookout for Vision provisions a minimum of one compute resource, known as an inference unit. You specify the number of inference units to use in the `MinInferenceUnits` input parameter to the `StartModel` API. The default allocation for a model is 1 inference unit.

You can increase or decrease the throughput of your model depending on the demands of your manufacturing line. To increase capacity, use additional inference units. Each additional inference unit increases your processing speed by one inference unit.

The transactions per second (TPS) that a single inference unit supports is affected by the following:

- The algorithm that Lookout for Vision uses to train the model. When you train a model, multiple models are trained. Lookout for Vision selects the model with the best performance based on the size of the dataset and its composition of normal and anomalous images.
- Higher resolution images require more time for analysis.
- Smaller sized images (measured in MBs) are analyzed faster than larger images.

Amazon Lookout for Vision distributes inference units across multiple Availability Zones within an AWS Region to provide increased availability. For more information, see Availability Zones. To help protect your production models from Availability Zone outages and inference unit failures, we strongly recommend that you start your production models with at least 2 inference units.

If an Availability Zone outage occurs, all inference units in the Availability Zone are unavailable and model capacity is reduced. Calls to DetectAnomalies are redistributed across the remaining inference units and succeed if the calls don't exceed the supported Transactions Per Seconds (TPS) of the remaining inference units. After AWS repairs the Availability Zone, the inference units are restarted, and full capacity is restored.

If a single inference unit fails, Lookout for Vision automatically starts a new inference unit in the same Availability Zone. Model capacity is reduced until the new inference unit starts.

You are charged for the number of hours that your model is running and for the number of inference units that your model uses whilst it is running. For example, if you start the model with 2 inference units and you use the model for 8 hours, you are charged for 16 inference hours (8 hours running time * 2 inference units).

> **Warning**
> If you don't explicitly stop your model by calling StopModel, you are charged even if you are not actively detecting anomalies with your model.

The Amazon Lookout for Vision console provides AWS CLI commands that you can use to start and stop your model. This section includes example Python code that you can use.

Topics

Starting your Amazon Lookout for Vision model

Before you can use an Amazon Lookout for Vision model to detect anomalies, you must first start the model. You start a model by calling the StartModel API and passing the following:

- **ProjectName** – The name of the project that contains the model that you want to start.
- **ModelVersion** – The version of the model that you want to start.
- **MinInferenceUnits** – The minimum number of inference units. For more information, see Running your trained Amazon Lookout for Vision model (p. 48).

The Amazon Lookout for Vision console provides example code that you can use to start and stop a model.

> **Note**
> You are charged for the amount of the time that your model is running. To stop a running model, see Stopping your Amazon Lookout for Vision model (p. 52).

Topics

Starting your model (console)

The Amazon Lookout for Vision console provides a AWS CLI command that you can use to start a model. After the model starts, you can start detecting anomalies in images. For more information, see Detecting anomalies in an image (p. 55).

To start a model (console)

1. If you haven't already done so, do the following:
   a. Create or update an IAM user with permissions to access Amazon Lookout for Vision. For more information, see Step 3: Set up permissions (p. 3).
   b. Install and configure the AWS CLI and the AWS SDKs. For more information, see Step 5: Set up the AWS CLI and AWS SDKs (p. 5).
2. Open the Amazon Lookout for Vision console at https://console.aws.amazon.com/lookoutvision/.
3. Choose **Get started**.
4. In the left navigation pane, choose **Projects**.
5. On the **Projects** resources page, choose the project that contains the trained model that you want to start.
6. In the **Models** section, choose the model that you want to start.
7. On the model's details page, choose the **Use model** tab.
8. Under **Code snippets** choose **AWS CLI commands**.
9. Copy the AWS CLI command that calls `start-model`.

10. At the command prompt, enter the `start-model` command that you copied in the previous step.
11. In the console, choose **Models** in the left navigation page.
12. Check the **Status** column for the current status of the model, When the status is **Hosted**, you can use the model to detect anomalies in images. For more information, see Detecting anomalies in an image (p. 55).

Starting your Amazon Lookout for Vision model (SDK)

You start a model by calling the StartModel operation.

A model might take a while to start. You can check the current status by calling DescribeModel. For more information, see Viewing your models (p. 74).

To start your model (SDK)

1. If you haven't already done so, do the following:
   a. Create or update an IAM user with permissions to access Amazon Lookout for Vision. For more information, see Step 3: Set up permissions (p. 3).
   b. Install and configure the AWS CLI and the AWS SDKs. For more information, see Step 5: Set up the AWS CLI and AWS SDKs (p. 5).
2. Use the following example code to start a model.

   CLI

   Change the following values:

   - `project-name` to the name of the project that contains the model that you want to start.
   - `model-version` to the version of the model that you want to start.
   - `--min-inference-units` to the number of anomaly detection units that you want to use.

```
aws lookoutvision start-model --project-name "project name"\
  --model-version model version\
  --min-inference-units number of units
```

   Python

   In the function `main`, change the following values:

   - `project` to the name of the project that contains the model that you want to start.
   - `model_version` to the version of the model that you want to start. Note that the console prepends the model version with the text `Model`. You only need to specify the version number. For example `12`.
   - `version_name` to the name of the model that you want to start.
   - `min_inference_units` to the number of inference units that you want to use.

```
# Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
# SPDX-License-Identifier: Apache-2.0

import time
import boto3
```

```
from botocore.exceptions import ClientError

def start_model(project_name, model_version, min_inference_units):
    """
    Starts the hosting of an Amazon Lookout for Vision model.
    param: project_name: The name of the project that contains the version
     of the model that you want to start hosting.
    param: model_version  The version of the model that you want to start hosting.
    param: min_inference_units The number of inference units to use for hosting.
    """

    try:

        client = boto3.client("lookoutvision")

        # Start the model
        print(
            "Starting model version " + model_version + " for project " +
 project_name
        )
        client.start_model(
            ProjectName=project_name,
            ModelVersion=model_version,
            MinInferenceUnits=min_inference_units,
        )

        print("Starting hosting...")

        # Wait until either hosted or failed.
        while True:

            model_description = client.describe_model(
                ProjectName=project_name, ModelVersion=model_version
            )
            status = model_description["ModelDescription"]["Status"]

            if status == "STARTING_HOSTING":
                print("Host starting in progress...")
                time.sleep(10)
                continue

            if status == "HOSTED":
                print("Model is hosted and ready for use.")
                break

            if status == "HOSTING_FAILED":
                print("Model hosting failed and the model can't be used.")
                break

            print("Failed. Unexpected state for hosting: " + status)
            break

    except ClientError as err:
        print("Service error: " + format(err))
        raise

    print("Done...")


def main():
    project = "my-project"  # Change to your project name
    model_version = "1"  # Change to the version of your model
    min_inference_units = (
        1  # Change to the number of inference units that you want to use for
 hosting.
```

```
    )

    start_model(project, model_version, min_inference_units)


if __name__ == "__main__":
    main()
```

3. If the output of the code is `Model is hosted and ready for use`, you can use the model to detect anomalies in images. For more information, see Detecting anomalies in an image (p. 55).

Stopping your Amazon Lookout for Vision model

To stop a running model, you call the `StopModel` operation and pass the following:

- **Project** – The name of the project that contains the model that you want to stop.
- **ModelVersion** – The version of the model that you want to stop.

The Amazon Lookout for Vision console provides example code that you can use to stop a model.

Note
You are charged for the amount of the time that your model is running.

Topics

Stopping your model (console)

Perform the steps in the following procedure to stop your model using the console.

To stop your model (console)

1. If you haven't already done so, do the following:
   a. Create or update an IAM user with permissions to access Amazon Lookout for Vision. For more information, see Step 3: Set up permissions (p. 3).
   b. Install and configure the AWS CLI and the AWS SDKs. For more information, see Step 5: Set up the AWS CLI and AWS SDKs (p. 5).
2. Open the Amazon Lookout for Vision console at https://console.aws.amazon.com/lookoutvision/.
3. Choose **Get started**.
4. In the left navigation pane, choose **Projects**.
5. On the **Projects** resources page, choose the project that contains the running model that you want to stop.
6. In the **Models** section, choose the model that you want to stop.
7. On the model's detail page, choose the **Use model** tab.
8. Under **Code snippets** choose **AWS CLI commands**.
9. Copy the AWS CLI command that calls `stop-model`.
10. At the command prompt, enter the `stop-model` command that you copied in the previous step.
11. At the console, choose **Models** in the left navigation page.
12. Check the **Status** column for the current status of the model. The model has stopped when the **Status** column value is **Training complete**.

Stopping your Amazon Lookout for Vision model (SDK)

You stop a model by calling the StopModel operation.

A model might take a while to stop. To check the current status, use `DescribeModel`.

To stop your model (SDK)

1. If you haven't already done so, do the following:
   a. Create or update an IAM user with permissions to access Amazon Lookout for Vision. For more information, see Step 3: Set up permissions (p. 3).
   b. Install and configure the AWS CLI and the AWS SDKs. For more information, see Step 5: Set up the AWS CLI and AWS SDKs (p. 5).
2. Use the following example code to stop a running model.

   CLI

   Change the following values:

   - `project-name` to the name of the project that contains the model that you want to stop.
   - `model-version` to the version of the model that you want to stop.

```
aws lookoutvision stop-model --project-name "project name"\
    --model-version model version
```

   Python

   The following example stops a model that is already running. In the function `main`, change the following values:

   - `project` to the name of the project that contains the model that you want to stop.
   - `model_version` to the version of the model that you want to stop.

```
# Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
# SPDX-License-Identifier: Apache-2.0

import time
import boto3

from botocore.exceptions import ClientError

def stop_model(project_name, model_version):
    """
    Stops a running Amazon Lookout for Vision Model
    param: project_Name: The name of the project that contains the
         version of the model that you want to stop hosting.
    param: model_version: The version of the nodel that you want to stop hosting.
    """

    try:

        client = boto3.client("lookoutvision")

        # Stop the model
```

```
        print(
            "Stopping model version " + model_version + " for project " +
 project_name
        )
        response = client.stop_model(
            ProjectName=project_name, ModelVersion=model_version
        )
        print("Stopping...")
        status = response["Status"]

        # Breaks when hosting has stopped.
        while True:
            model_description = client.describe_model(
                ProjectName=project_name, ModelVersion=model_version
            )
            status = model_description["ModelDescription"]["Status"]

            if status == "STOPPING_HOSTING":
                print("Host stopping in progress...")
                time.sleep(10)
                continue

            if status == "TRAINED":
                print("Model is no longer hosted.")
                break

            print("Failed. Unxexpected state for stopping model: " + status)
            break
    except ClientError as err:
        print("Service error: " + format(err))
        raise

    print("Done...")


def main():
    project = "my-project"  # Change to your project name.
    model_version = "1"  # Change to the version of your model.

    stop_model(project, model_version)


if __name__ == "__main__":
    main()
```

Detecting anomalies in an image

To detect anomalies in an image with a trained Amazon Lookout for Vision model, you call the DetectAnomalies operation. The result from `DetectAnomalies` includes a boolean prediction that the image contains one or more anomalies and a confidence value for the prediction.

Images analyzed with `DetectAnomalies` must have the same dimensions as the images used to train the model.

> **Note**
> Before calling `DetectAnomalies`, you must first start your model with the `StartModel` operation. For more information, see Starting your Amazon Lookout for Vision model (p. 49). You are charged for the amount of time, in minutes, that a model runs and for the number of anomaly detection units that your model uses. If you are not using a model, use the `StopModel` operation to stop your model. For more information, see Stopping your Amazon Lookout for Vision model (p. 52).

To call `DetectAnomalies`, specify the following:

- **Project** – The name of the project that contains the model that you want to use.
- **ModelVersion** – The version of the model that you want to use.
- **ContentType** – The type of image that you want analyze. Valid values are `image/png` (PNG format images) and `image/jpeg` (JPG format images).
- **Body** – The unencoded binary bytes that represent the image.

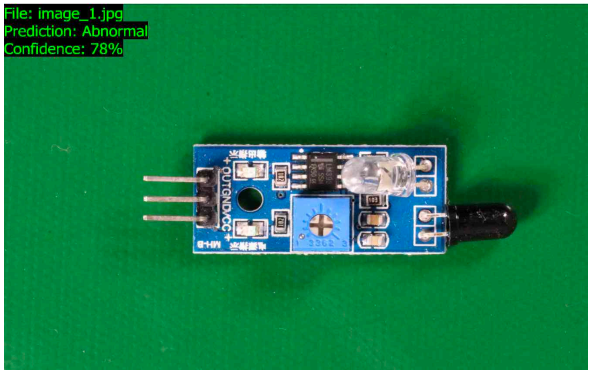
The response from `DetectAnomalies` contains the following:

- **IsAnomalous**– A boolean indicator that the image contains one or more anomalies.
- **Confidence**– The confidence that Amazon Lookout for Vision has in the accuracy of the anomaly prediction (`IsAnomalous`). `Confidence` is a floating point value between 0 and 1. A higher value indicates a higher confidence.
- **Source** – Information about the image passed to `DetectAnomalies`.

```
{
    "DetectAnomalyResult": {
        "Source": {
            "Type": "direct"
        },
        "IsAnomalous": true,
        "Confidence": 0.9996867775917053
    }
}
```

If you're finding the confidence values returned by `DetectAnomalies` are too low, consider retraining the model.

The Python and Java examples display the image and overlay the analysis results, as shown in the following image.



To detect anomalies in an image (API)

1. If you haven't already done so, do the following:
   a. Create or update an IAM user. For more information, see Step 2: Create an IAM administrator user and group (p. 2).
   b. Install and configure the AWS CLI and the AWS SDKs. For more information, see Step 5: Set up the AWS CLI and AWS SDKs (p. 5).
   c. Train your model (p. 22).
   d. Start your model (p. 49).
2. Ensure the IAM user calling `DetectAnomalies` has access to the model version that you want to use.
3. Use the following examples to call the `DetectAnomalies` operation.

   Tip
   If you're using the Getting Started images, use an image from the `extra_images` folder.

   AWS CLI

   This AWS CLI command displays the JSON output for the `DetectAnomalies` CLI operation. Change the values of the following input parameters:

   - `project name` with the name of the project that you want to use.
   - `model version` with the version of the model that you want to use.
   - `content type` with the type of the image that you want to use. Valid values are `image/png` (PNG format images) and `image/jpeg` (JPG format images).
   - `file name` with the path and file name of the image that you want to use. Ensure the file type matches the value of `content-type`.

```
aws lookoutvision detect-anomalies  --project-name project name\
    --model-version model version\
    --content-type content type\
    --body file name
```

   Python

   The following example code detects anomalies in an image that you supply. The example takes the following command line options:

   - `project` – the name of the project that you want to use.
   - `version` – the version of the model, within the project, that you want to use.

- `image` – the path and file of a local image file (JPEG or PNG format).

```
# Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
# SPDX-License-Identifier: Apache-2.0
"""
Uses a trained Amazon Lookout for Vision model to detect anomalies
in an image. The image can be local or in an S3 bucket.
"""

import argparse
import logging
import io
import boto3
from PIL import Image, ImageDraw, ImageFont

from botocore.exceptions import ClientError

logger = logging.getLogger(__name__)


class Inference:
    """
    Shows how to detect anomalies in an image using a trained Amazon Lookout
    for Vision model.
    """

    @staticmethod
    def show_anomaly_prediction(lookoutvision_client,
 project_name,model_version,photo):
        """
        Detects anomalies in an image (jpg/png) by using your Amazon Lookout for
 Vision
        model. Displays the image and overlays prediction information text.
        :param lookoutvision_client: An Amazon Lookout for Vision Boto3 client.
        :param project_name: The name of the project that contains the model that
        you want to use.
        :param model_version: The version of the model that you want to use.
        :param photo: The path and name of the image in which you want to detect
        anomalies.
        """
        try:

            image=Image.open(photo)
            image_type=Image.MIME[image.format]

            if (image_type == "image/jpeg" or image_type== "image/png") == False:
                logger.info("Invalid image type for %s", photo)
                raise ValueError(
                    f"Invalid file format. Supply a jpeg or png format file:
 {photo}"
                )

            # get images bytes for call to detect_anomalies
            image_bytes = io.BytesIO()
            image.save(image_bytes, format=image.format)
            image_bytes = image_bytes.getvalue()

            # analyze the image
            response = lookoutvision_client.detect_anomalies(
                    ProjectName=project_name,
                    ContentType=image_type,
                    Body=image_bytes,
                    ModelVersion=model_version,
```

```
                )

            logger.info("Detecting anomalies in %s", photo)

            prediction = "Normal"
            if (response["DetectAnomalyResult"]["IsAnomalous"]==False):
                    prediction="Abnormal"

            confidence=response["DetectAnomalyResult"]["Confidence"]

            logger.info("Prediction: %s", format(prediction))
            logger.info("Confidence: %s", format(confidence))

            draw = ImageDraw.Draw(image)

            font_size=100

            fnt = ImageFont.truetype('/Library/Fonts/Tahoma.ttf', font_size)
            y1=0

            # Draw black box and overlay text
            for text in [f"File: {photo}", f"Prediction: {prediction}",
 f"Confidence: {confidence:.2%}"]:

                text_width, text_height = draw.textsize(text,fnt)

                y2 = y1 + text_height

                draw.rectangle([(10 , y1), (text_width + 10, y2)],fill="black")
                draw.text((10,y1), text, fill="lime", font=fnt)

                y1+=text_height


            image.show()

        except ClientError as err:
            logger.info(format(err))
            raise



def add_arguments(parser):
    """
    Adds command line arguments to the parser.
    :param parser: The command line parser.
    """

    parser.add_argument(
        "project", help="The project containing the model that you want to use."
    )
    parser.add_argument(
        "version", help="The version of the model that you want to use."
    )
    parser.add_argument(
        "image",
        help="The file that you want to analyze. "
        "Supply a local file path.",
    )

def main():
    """
    Entrypoint for anomaly detection example.
    """

    try:
```

```
        logging.basicConfig(level=logging.INFO, format="%(levelname)s:
 %(message)s")
        lookoutvision_client = boto3.client("lookoutvision")

        parser = argparse.ArgumentParser(usage=argparse.SUPPRESS)

        add_arguments(parser)

        args = parser.parse_args()

        # analyze image
        Inference.show_anomaly_prediction(
            lookoutvision_client, args.project, args.version, args.image
        )

    except ClientError as err:
        print("A service error occured: " + format(err.response["Error"]
["Message"]))
    except FileNotFoundError as err:
        print("The supplied file couldn't be found: " + err.filename)
    except ValueError as err:
        print("A value error occured. " + format(err))
    else:
        print("Successfully completed analysis.")

if __name__ == "__main__":
    main().
```

Java 2

The following example displays the input image and overlays the file name, the anomaly prediction, and the prediction confidence. The example takes the following command line options:

- `project` – the name of the project that you want to use.
- `version` – the version of the model, within the project, that you want to use.
- `image` – the path and file of a local image file (JPEG or PNG format).

```
//Copyright 2021 Amazon.com, Inc. or its affiliates. All Rights Reserved.
//PDX-License-Identifier: MIT-0 (For details, see https://github.com/awsdocs/
amazon-lookout-for-vision-developer-guide/blob/main/LICENSE-SAMPLECODE.)

package com.example.lookoutvision;

import software.amazon.awssdk.core.SdkBytes;
import software.amazon.awssdk.core.sync.RequestBody;
import software.amazon.awssdk.services.lookoutvision.LookoutVisionClient;
import software.amazon.awssdk.services.lookoutvision.model.DetectAnomaliesRequest;
import software.amazon.awssdk.services.lookoutvision.model.DetectAnomaliesResponse;
import software.amazon.awssdk.services.lookoutvision.model.DetectAnomalyResult;
import software.amazon.awssdk.services.lookoutvision.model.LookoutVisionException;

import java.io.BufferedInputStream;
import java.io.ByteArrayInputStream;
import java.io.File;
import java.io.FileInputStream;
import java.io.FileNotFoundException;
import java.io.IOException;
import java.io.InputStream;
import java.net.URLConnection;

import java.text.NumberFormat;
```

```
import java.awt.*;
import java.awt.font.LineMetrics;
import java.awt.image.BufferedImage;
import javax.imageio.ImageIO;
import javax.swing.*;

import java.util.logging.Level;
import java.util.logging.Logger;

// Finds anomalies on a supplied image
public class DetectAnomalies extends JPanel {

    private static final long serialVersionUID = 1L;
    private transient BufferedImage image;
    private transient Dimension dimension;
    public static final Logger logger =
 Logger.getLogger(DetectAnomalies.class.getName());

    // Constructor. Finds anomalies in a local image file.
    public DetectAnomalies(LookoutVisionClient lfvClient, String projectName,
 String modelVersion,
            String photo) throws IOException, LookoutVisionException {

        logger.log(Level.INFO, "Processing local file: {0}", photo);


        // Get image bytes and buffered image
        InputStream sourceStream = new FileInputStream(new File(photo));
        SdkBytes imageSDKBytes = SdkBytes.fromInputStream(sourceStream);
        byte[] imageBytes = imageSDKBytes.asByteArray();
        ByteArrayInputStream inputStream = new
 ByteArrayInputStream(imageSDKBytes.asByteArray());
        image = ImageIO.read(inputStream);

        //Get the image type. Can be image/jpeg or image/png.
        String contentType=getImageType(imageBytes);


        // Set the size of the window that shows the image
        setWindowDimensions();

        //Detect anomalies in the supplied image.
        DetectAnomaliesRequest request =
 DetectAnomaliesRequest.builder().projectName(projectName)
                .modelVersion(modelVersion).contentType(contentType).build();


        DetectAnomaliesResponse response = lfvClient.detectAnomalies(request,
             RequestBody.fromBytes(imageBytes));

        /* Tip: You can also use the following to analyze a local file
        Path path = Paths.get(photo);
        DetectAnomaliesResponse response = lfvClient.detectAnomalies(request,
 path);
        */
        DetectAnomalyResult result = response.detectAnomalyResult();

        String prediction = "Prediction: Normal";

        if (Boolean.TRUE.equals(result.isAnomalous())) {
            prediction = "Prediction: Abnormal";
        }

        // Convert prediction to percentage
        NumberFormat defaultFormat = NumberFormat.getPercentInstance();
        defaultFormat.setMinimumFractionDigits(1);
```

```
        String confidence = String.format("Confidence: %s",
defaultFormat.format(result.confidence()));

        // Draw file name, prediction, and confidence on image
        String photoPath = "File: " + photo;
        String[] imageLines = { photoPath, prediction, confidence };
        drawImageInfo(imageLines);

        logger.log(Level.INFO, "Image: {0}\nAnomalous: {1}\nConfidence {2}",
imageLines);

    }

    // Sets window dimensions to 1/2 screen size, unless image is smaller
    public void setWindowDimensions() {
        dimension = java.awt.Toolkit.getDefaultToolkit().getScreenSize();

        dimension.width = (int) dimension.getWidth() / 2;
        if (image.getWidth() < dimension.width) {
            dimension.width = image.getWidth();
        }
        dimension.height = (int) dimension.getHeight() / 2;

        if (image.getHeight() < dimension.height) {
            dimension.height = image.getHeight();
        }

        setPreferredSize(dimension);

    }

    // Draws the file name, prediction, and confidence on the image.
    public void drawImageInfo(String[] imageLines) {

        int indent = 10;

        // Set up drawing
        Graphics2D g2d = image.createGraphics();
        g2d.setColor(Color.GREEN);
        g2d.setFont(new Font("Tahoma", Font.PLAIN, 80));
        Font font = g2d.getFont();
        FontMetrics metrics = g2d.getFontMetrics(font);

        int y1=0;

          for (int i = 0; i < imageLines.length; i++) {

            // Get text height, width, and descent
            int textWidth=metrics.stringWidth(imageLines[i]);
            LineMetrics lm=metrics.getLineMetrics(imageLines[i], g2d);
            int textHeight = (int)lm.getHeight();
            int descent=(int)lm.getDescent();

            int y2=(y1+textHeight)-descent;

            // Draw black rectangle
            g2d.setColor(Color.BLACK);
            g2d.fillRect(indent, y1, textWidth, textHeight);

            // Draw text
            g2d.setColor(Color.GREEN);
            g2d.drawString(imageLines[i], indent, y2);

            y1+=textHeight;

        }
```

```
        g2d.dispose();

    }

    // Draws the image containing the bounding boxes and labels.
    @Override
    public void paintComponent(Graphics g) {

        Graphics2D g2d = (Graphics2D) g; // Create a Java2D version of g.

        // Draw the image.
        g2d.drawImage(image, 0, 0, dimension.width, dimension.height, this);

    }


    // Gets the image mime type. Supported formats are image/jpeg and image/png.
    private String getImageType(byte[] image) throws IOException{


        InputStream is = new BufferedInputStream(new ByteArrayInputStream(image));
        String mimeType = URLConnection.guessContentTypeFromStream(is);

        logger.log(Level.INFO, "Image type: {0}", mimeType);


        if (mimeType.equals("image/jpeg") || mimeType.equals("image/png")){
            return mimeType;
        }
        // not a supported file type.
        logger.log(Level.SEVERE, "Unsupported image type: {0}", mimeType);
        throw new IOException(String.format("Wrong image type. %s format isn't
supported.",mimeType));
        }

    public static void main(String args[]) throws Exception {

        String photo = null;
        String projectName = null;
        String modelVersion = null;

        final String USAGE = "\n" +
        "Usage:\n" +
        "    DetectAnomalies <project> <version> <image> \n\n" +
        "Where:\n" +
        "    project - the Lookout for Vision project.\n\n"+
        "    version - the version of the model within the project.\n\n" +
        "    image - the path and filename of a local image. \n\n";


        try {

            if (args.length != 3) {
                System.out.println(USAGE);
                System.exit(1);
            }

            projectName = args[0];
            modelVersion = args[1];
            photo = args[2];
            DetectAnomalies panel = null;


            // Get the Lookout for Vision client
            LookoutVisionClient lfvClient = LookoutVisionClient.builder().build();
```

```
            // Create frame and panel.
            JFrame frame = new JFrame("Anomaly Detection");
            frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

            panel = new DetectAnomalies(lfvClient, projectName, modelVersion,
 photo);

            frame.setContentPane(panel);
            frame.pack();
            frame.setVisible(true);

        } catch (LookoutVisionException lfvError) {
            logger.log(Level.SEVERE, "lookout for vision client error: {0}",
 lfvError.getMessage());
            System.out.println(String.format("lookout for vision client error: %s",
 lfvError.getMessage()));
            System.exit(1);
        }
        catch(FileNotFoundException fileError)
        {
            logger.log(Level.SEVERE, "Could not find file: {0}",
 fileError.getMessage());
            System.out.println(String.format("Could not find file: %s",
 fileError.getMessage()));
            System.exit(1);
        }
        catch(IOException ioError)
        {
            logger.log(Level.SEVERE, "IO error {0}", ioError.getMessage());
            System.out.println(String.format("IO error: %s",
 ioError.getMessage()));
            System.exit(1);
        }


    }
}
```

4. If you aren't planning to continue using your model, stop your model (p. 52).

Managing your Amazon Lookout for Vision resources

You can manage your Amazon Lookout for Vision resources by using the console or the AWS SDK. Amazon Lookout for Vision has the following resources:

- Projects
- Datasets
- Models
- Trial detections

> **Note**
> You can't delete a trial detection task. Also, you can't manage trial detections by using the AWS SDK.

Topics

Viewing your projects

You can get a list of Amazon Lookout for Vision projects and information about individual projects from the console or by using the AWS SDK.

> **Note**
> The list of projects is eventually consistent. If you create or delete a project, you might have to wait a short while before the projects list is up to date.

Viewing your projects (console)

Perform the steps in the following procedure to view your projects in the console.

To view your projects

1. Open the Amazon Lookout for Vision console at https://console.aws.amazon.com/lookoutvision/.
2. Choose **Get started**.
3. In the left navigation pane, choose **Projects**. The projects view is shown.

4. Choose a project name to see the project's details.

Viewing your projects (SDK)

A project manages the datasets and models for a single use case. For example, detecting anomalies in machine parts. The following example calls `ListProjects` to get a list of your projects. The example then calls `DescribeProject` to get the current status of each project.

To view your projects (SDK)

1. If you haven't already done so, do the following:
   a. Create or update an IAM user with permissions to access Amazon Lookout for Vision. For more information, see Step 3: Set up permissions (p. 3).
   b. Install and configure the AWS CLI and the AWS SDKs. For more information, see Step 5: Set up the AWS CLI and AWS SDKs (p. 5).
2. Use the following example code to view your projects.

   CLI

   Use the `list-projects` command to list the projects in your account.

```
aws lookoutvision list-projects
```

   Use the `describe-project` command to get information about a project.

   Change the value of `project-name` to the name of the project that you want to describe.

```
aws lookoutvision describe-project --project-name project_name
```

   Python

```
# Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
# SPDX-License-Identifier: Apache-2.0


import boto3

from botocore.exceptions import ClientError


def list_projects():
    """
    Lists the projects, models and datasets in your account.
    """

    try:
        client = boto3.client("lookoutvision")

        response = client.list_projects()

        for project in response["Projects"]:

            print("Project: " + project["ProjectName"])
            print("\tARN: " + project["ProjectArn"])
            print("\tCreated: " + str(["CreationTimestamp"]))
```

```
            print("Datasets")
            project_description = client.describe_project(
                ProjectName=project["ProjectName"]
            )
            if len(project_description["ProjectDescription"]["Datasets"]) == 0:
                print("\tNo datasets")
            else:
                for dataset in project_description["ProjectDescription"]
["Datasets"]:
                    print("\ttype: " + dataset["DatasetType"])
                    print("\tStatus: " + dataset["StatusMessage"])

            print("Models")
            # list models
            response_models =
 client.list_models(ProjectName=project["ProjectName"])
            if len(response_models["Models"]) == 0:
                print("\tNo models")
            else:
                for model in response_models["Models"]:
                    print("\tVersion: " + model["ModelVersion"])
                    print("\tARN: " + model["ModelArn"])
                    if "Description" in model:
                        print("\tDescription: " + model["Description"])

                    # Get model description
                    model_description = client.describe_model(
                        ProjectName=project["ProjectName"],
                        ModelVersion=model["ModelVersion"],
                    )
                    print(
                        "\tStatus: " + model_description["ModelDescription"]
["Status"]
                    )
                    print(
                        "\tMessage: "
                        + model_description["ModelDescription"]["StatusMessage"]
                    )
                    print(
                        "\tCreated: "
                        + str(
                            model_description["ModelDescription"]
["CreationTimestamp"]
                        )
                    )

                    if model_description["ModelDescription"]["Status"] ==
 "TRAINED":
                        print(
                            "\tTraining duration: "
                            + str(
                                model_description["ModelDescription"][
                                    "EvaluationEndTimestamp"
                                ]
                                - model_description["ModelDescription"][
                                    "CreationTimestamp"
                                ]
                            )
                        )
                        print(
                            "\tRecall: "
                            + str(
                                model_description["ModelDescription"]
["Performance"][
                                    "Recall"
```

```
                                ]
                            )
                        )
                        print(
                            "\tPrecision: "
                            + str(
                                model_description["ModelDescription"]
["Performance"][
                                    "Precision"
                                ]
                            )
                        )
                        print(
                            "\tF1: "
                            + str(
                                model_description["ModelDescription"]
["Performance"][
                                    "F1Score"
                                ]
                            )
                        )
                        print(
                            "\tTraining output : s3://"
                            + str(
                                model_description["ModelDescription"]
["OutputConfig"][
                                    "S3Location"
                                ]["Bucket"]
                            )
                            + "/"
                            + str(
                                model_description["ModelDescription"]
["OutputConfig"][
                                    "S3Location"
                                ]["Prefix"]
                            )
                        )

                    print()

            print()

    except ClientError as err:
        print("Service error: " + format(err))
        raise

    print("Done...")


def main():
    list_projects()


if __name__ == "__main__":
    main()
```

Deleting a project

You can delete a project from the projects view page in the console or by using the `DeleteProject` operation.

The images referenced by a project's datasets aren't deleted.

Deleting a project (console)

Use the following procedure to delete a project. If you use the console procedure, associated model versions and datasets are deleted for you.

To delete a project

1. Open the Amazon Lookout for Vision console at https://console.aws.amazon.com/lookoutvision/.
2. Choose **Get started**.
3. In the left navigation pane, choose **Projects**.
4. On the **Projects** page, select the project that you want to delete.
5. Choose **Delete** at the top of the page.
6. In the **Delete** dialog box, enter **delete** to confirm that you want to delete the project.
7. If necessary, choose to delete any associated datasets and models.
8. Choose **Delete project**.

Deleting a project (SDK)

You delete an Amazon Lookout for Vision project by calling DeleteProject and supplying the name of the project that you want to delete.

Before you can delete a project, you must first delete all models in the project. For more information, see Deleting a model (SDK) (p. 77). You also have to delete the datasets associated with the model. For more information, see Deleting a dataset (p. 72).

The project might take a few moments to delete. During that time, the status of the project is `DELETING`. The project is deleted if a subsequent call to `DeleteProject` doesn't include the project that you deleted.

To delete a project (SDK)

1. If you haven't already done so, do the following:
   a. Create or update an IAM user with permissions to access Amazon Lookout for Vision. For more information, see Step 3: Set up permissions (p. 3).
   b. Install and configure the AWS CLI and the AWS SDKs. For more information, see Step 5: Set up the AWS CLI and AWS SDKs (p. 5).
2. Use the following code to delete a project.

   AWS CLI

   Change the value of `project-name` to the name of the project that you want to delete.

```
aws lookoutvision delete-project --project-name project_name
```

   Python

   In the function `main`, change the value of `project_name` to the name of the project you want to delete.

```
# Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
# SPDX-License-Identifier: Apache-2.0
```

```
import boto3

from botocore.exceptions import ClientError


def delete_project(project_name):
    """
    Deletes an Amazon Lookout for Vision Model
    param: project_name: The name of the project that you want to delete.
    """

    try:
        client = boto3.client("lookoutvision")

        # Delete a project
        print("Deleting project:" + project_name)
        response = client.delete_project(ProjectName=project_name)
        print("Deleted project ARN: " + response["ProjectArn"])
        print("Done...")

    except ClientError as err:
        print("Service error: " + format(err))
        raise


def main():
    project_name = (
        "my-project"  # Change to the name of the project that you want to delete.
    )

    delete_project(project_name)


if __name__ == "__main__":
    main()
```

Creating Amazon Lookout for Vision projects with AWS CloudFormation

Amazon Lookout for Vision is integrated with AWS CloudFormation, a service that helps you model and set up your AWS resources so that you can spend less time creating and managing your resources and infrastructure. You create a template that describes all the AWS resources that you want, and AWS CloudFormation takes care of provisioning and configuring those resources for you.

You can use AWS CloudFormation to provision and configure Amazon Lookout for Vision projects.

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

Lookout for Vision and AWS CloudFormation templates

To provision and configure projects for Lookout for Vision 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*.

For reference information about Lookout for Vision projects, including examples of JSON and YAML templates, see LookoutVision resource type reference.

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

Viewing your datasets

A project can have a single dataset that's used for training and testing your model. Alternatively, You can have separate training and test datasets. You can use the console to view your datasets. You can also use the `DescribeDataset` operation to get information about a dataset (training or test).

Viewing the datasets in a project (console)

Perform the steps in the following procedure to view your project's datasets in the console.

To view your datasets (console)

1. Open the Amazon Lookout for Vision console at https://console.aws.amazon.com/lookoutvision/.
2. Choose **Get started**.
3. In the left navigation pane, choose **Projects**.
4. On the **Projects** page, select the project that contains the datasets that you want to view.
5. In the left navigation pane, choose **Dataset** to view the dataset details. If you have a training and a test dataset, a tab for each dataset is shown.

Viewing the datasets in a project (SDK)

You can get use the `DescribeDataset` operation to get information about the training or test dataset associated with a project.

To view your datasets (SDK)

1. If you haven't already done so, do the following:
   a. Create or update an IAM user with permissions to access Amazon Lookout for Vision. For more information, see Step 3: Set up permissions (p. 3).
   b. Install and configure the AWS CLI and the AWS SDKs. For more information, see Step 5: Set up the AWS CLI and AWS SDKs (p. 5).
2. Use the following example code to view a dataset.

   CLI

   Change the following values:

- `project-name` to the name of the project that contains the model that you want to view.
- `dataset-type` to the type of dataset that you want to view (`train` or `test`).

```
aws lookoutvision describe-dataset --project-name project name\
  --dataset-type train or test
```

Python

In the function `main`, change the following values:

- `project_name` to the name of the project that contains the model that you want to view.
- `dataset_type` to the type of dataset that you want to view (`train` or `test`).

```
# Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
# SPDX-License-Identifier: Apache-2.0

import boto3

from botocore.exceptions import ClientError

def describe_dataset(project_name, dataset_type):
    """
    Gets information about an Amazon Lookout for Vision dataset.
    param: project_name: The name of the project that contains the
     dataset that you want to describe.
    param: dataset_type: The type (train or test) of the dataset that you want to
 describe.
    """

    try:

        client = boto3.client("lookoutvision")

        # Describe a dataset

        response = client.describe_dataset(
            ProjectName=project_name, DatasetType=dataset_type
        )
        print("Name: " + response["DatasetDescription"]["ProjectName"])
        print("Type: " + response["DatasetDescription"]["DatasetType"])
        print("Status: " + response["DatasetDescription"]["Status"])
        print("Message: " + response["DatasetDescription"]["StatusMessage"])
        print("Images: " + str(response["DatasetDescription"]["ImageStats"]
["Total"]))
        print(
            "Labeled: " + str(response["DatasetDescription"]["ImageStats"]
["Labeled"])
        )
        print("Normal: " + str(response["DatasetDescription"]["ImageStats"]
["Normal"]))
        print(
            "Anomaly: " + str(response["DatasetDescription"]["ImageStats"]
["Anomaly"])
        )

        print("Done...")

    except ClientError as err:
        print("Service error: " + format(err))
```

```
        raise


def main():
    project_name = (
        "my-project"  # Change to the project name that contains the dataset.
    )
    dataset_type = "train"  # Change to the dataset type (train or test)
    describe_dataset(project_name, dataset_type)


if __name__ == "__main__":
    main()
```

Deleting a dataset

You can delete a dataset from a project by using the console or the `DeleteDataset` operation. The images referenced by a dataset aren't deleted. If you delete the test dataset from a project that has a training and a test dataset, the project reverts to a single dataset project—the remaining dataset is split during training to create a training and test dataset. If you delete the training dataset, you can't train a model in the project until you create a new training dataset.

Deleting a dataset (console)

Perform the steps in the following procedure to delete a dataset. If you delete all of the datasets in a project, the **Create dataset** page is displayed.

To delete a dataset (console)

1. Open the Amazon Lookout for Vision console at https://console.aws.amazon.com/lookoutvision/.
2. Choose **Get started**.
3. In the left navigation pane, choose **Projects**.
4. On the **Projects** page, select the project that contains the dataset that you want to delete.
5. In the left navigation pane, choose **Dataset**.
6. Choose **Actions** and then select the dataset that you want to delete.
7. In the **Delete** dialog box, enter **delete** to confirm that you want to delete the dataset.
8. Choose **Delete training dataset** or **Delete test dataset** to delete the dataset.

Deleting a dataset (SDK)

Use the `DeleteDataset` operation to delete a dataset.

To delete a dataset (SDK)

1. If you haven't already done so, do the following:
   a. Create or update an IAM user with permissions to access Amazon Lookout for Vision. For more information, see Step 3: Set up permissions (p. 3).
   b. Install and configure the AWS CLI and the AWS SDKs. For more information, see Step 5: Set up the AWS CLI and AWS SDKs (p. 5).
2. Use the following example code to delete a model.

CLI

Change the value of the following

- `project-name` to the name of the project that contains the model that you want to delete.
- `dataset-type` to either `train` or `test`, depending on which dataset you want to delete. If you have a single dataset project, specify `train` to delete the dataset.

```
aws lookoutvision delete-dataset --project-name project name\
  --dataset-type dataset type
```

Python

In the function `main`, change the following values:

- `project_name` to the name of the project that contains the model that you want to delete.
- `dataset_type` to either `train` or `test`, depending on which dataset you want to delete.

```
# Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
# SPDX-License-Identifier: Apache-2.0


import boto3

from botocore.exceptions import ClientError


def delete_dataset(project_name, dataset_type):
    """
    Deletes an Amazon Lookout for Vision dataset
    param: project_name: The name of the project that contains the dataset that you
 want to delete.
    param: dataset_type: the type (train or test) of the dataset that you want to
 delete.
    """
    try:
        client = boto3.client("lookoutvision")

        # Delete the dataset
        print("Deleting dataset:" + dataset_type)
        client.delete_dataset(ProjectName=project_name, DatasetType=dataset_type)

        print("Done...")

    except ClientError as err:
        print("Service error: " + format(err))
        raise


def main():
    project_name = "my-project"  # the desired project.
    dataset_type = "train"  # or 'test' to delete the test dataset.
    delete_dataset(project_name, dataset_type)


if __name__ == "__main__":
    main()
```

Viewing your models

A project can have multiple versions of a model. You can use the console to view the models in a project. You can also use the `ListModels` operation.

Note
The list of models is eventually consistent. If you create a model, you might have to wait a short while before the models list is up to date.

Viewing your models (console)

Perform the steps in the following procedure to view the your project's models in the console.

To view your models (console)

1. Open the Amazon Lookout for Vision console at https://console.aws.amazon.com/lookoutvision/.
2. Choose **Get started**.
3. In the left navigation pane, choose **Projects**.
4. On the **Projects** page, select the project that contains the models that you want to view.
5. In the left navigation pane, choose **Models** and then view the model details.

Viewing your models (SDK)

To get view the versions of a model you use the `ListModels` operation. To get information about a specific model version, use the `DescribeModel` operation. The following example lists all the model versions in a project and then displays performance and output configuration information for individual model versions.

To view your models (SDK)

1. If you haven't already done so, do the following:
   a. Create or update an IAM user with permissions to access Amazon Lookout for Vision. For more information, see Step 3: Set up permissions (p. 3).
   b. Install and configure the AWS CLI and the AWS SDKs. For more information, see Step 5: Set up the AWS CLI and AWS SDKs (p. 5).
2. Use the following example code to list your models and get information about a model.

   CLI

   Use the `list-models` command to list the models in a project.

   Change the following value:

   - `project-name` to the name of the project that contains the model that you want to view.

```
aws lookoutvision list-models --project-name project name
```

   Use the `describe-model` command to get information about a model. Change the following values:

   - `project-name` to the name of the project that contains the model that you want to view.
   - `model-version` to the version of the model that you want to describe.

```
aws lookoutvision describe-model --project-name project name\
    --model-version model version
```

Python

In the function `main`, change the following value:

- `project_name` to the name of the project.

```
# Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
# SPDX-License-Identifier: Apache-2.0

import boto3

from botocore.exceptions import ClientError

def describe_models(project_name):
    """
    Gets information about all models in an Amazon Lookout for Vsion project.
    param: project_name: The name of the project that you want to use.
    """
    try:

        client = boto3.client("lookoutvision")

        # list models
        response = client.list_models(ProjectName=project_name)
        print("Project: " + project_name)
        for model in response["Models"]:
            print("Model version: " + model["ModelVersion"])
            print("\tARN: " + model["ModelArn"])
            if "Description" in model:
                print("\tDescription: " + model["Description"])

            # Get model description
            model_description = client.describe_model(
                ProjectName=project_name, ModelVersion=model["ModelVersion"]
            )
            print("\tStatus: " + model_description["ModelDescription"]["Status"])
            print(
                "\tMessage: " + model_description["ModelDescription"]
["StatusMessage"]
            )
            print(
                "\tCreated: "
                + str(model_description["ModelDescription"]["CreationTimestamp"])
            )

            if model_description["ModelDescription"]["Status"] == "TRAINED":
                print(
                    "\tTraining duration: "
                    + str(
                        model_description["ModelDescription"]
["EvaluationEndTimestamp"]
                        - model_description["ModelDescription"]
["CreationTimestamp"]
                    )
                )
                print(
                    "\tRecall: "
```

```
                    + str(
                        model_description["ModelDescription"]["Performance"]
["Recall"]
                    )
                )
                print(
                    "\tPrecision: "
                    + str(
                        model_description["ModelDescription"]["Performance"][
                            "Precision"
                        ]
                    )
                )
                print(
                    "\tF1: "
                    + str(
                        model_description["ModelDescription"]["Performance"]
["F1Score"]
                    )
                )
                print(
                    "\tTraining output : s3://"
                    + str(
                        model_description["ModelDescription"]["OutputConfig"][
                            "S3Location"
                        ]["Bucket"]
                    )
                    + "/"
                    + str(
                        model_description["ModelDescription"]["OutputConfig"][
                            "S3Location"
                        ]["Prefix"]
                    )
                )

            print()

        print("Done...")

    except ClientError as err:
        print("Service error: " + format(err))
        raise


def main():
    project_name = "my-project"  # Change to the name of your project.
    describe_models(project_name)


if __name__ == "__main__":
    main()
```

Deleting a model

You can delete a version of a model by using the console or by using the `DeleteModel` operation. You can't delete model version that is running or being trained.

If the model is version running, first use the `StopModel` operation to stop the model version. For more information, see Stopping your Amazon Lookout for Vision model (p. 52). If a model is training, wait until it finishes before you delete the model.

It might take a few seconds to delete a model. To determine if a model has been deleted, call ListProjects and check if the version of the model (`ModelVersion`) is in the `Models` array.

Deleting a model (console)

Perform the following steps to delete a model from the console.

To delete a model (console)

1. Open the Amazon Lookout for Vision console at https://console.aws.amazon.com/lookoutvision/.
2. Choose **Get started**.
3. In the left navigation pane, choose **Projects**.
4. On the **Projects** page, select the project that contains the model that you want to delete.
5. In the left navigation pane, choose **Models**.
6. On the **models** view, select the radio button for the model that you want to delete.
7. Choose **Delete** at the top of the page.
8. In the **Delete** dialog box, enter **delete** to confirm that you want to delete the model.
9. Choose **Delete model** to delete the model.

Deleting a model (SDK)

Use the following procedure to delete model with the `DeleteModel` operation.

To delete a model (SDK)

1. If you haven't already done so, do the following:
   a. Create or update an IAM user with permissions to access Amazon Lookout for Vision. For more information, see Step 3: Set up permissions (p. 3).
   b. Install and configure the AWS CLI and the AWS SDKs. For more information, see Step 5: Set up the AWS CLI and AWS SDKs (p. 5).
2. Use the following example code to delete a model.

   CLI

   Change the following values:

   - `project-name` to the name of the project that contains the model that you want to delete.
   - `model-version` to the version of the model that you want to delete.

```
aws lookoutvision delete-model --project-name project name\
  --model-version model version
```

   Python

   In the function `main`, change the following values:

   - `project_name` to the name of the project that contains the model that you want to delete.
   - `model_version` to the version of the model that you want to delete.

```
# Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
# SPDX-License-Identifier: Apache-2.0
```

```
import time
import boto3

from botocore.exceptions import ClientError


def delete_model(project_name, model_version):
    """
    Deletes an Amazon Lookout for Vision model. The model must first be
    stopped and can't be in training.
    param: project_name: The name of the project that contains the desired model.
    param: model_version: The version of the model that you want to delete.
    """

    try:
        client = boto3.client("lookoutvision")

        # Delete the model
        print("Deleting model:" + model_version)
        response = client.delete_model(
            ProjectName=project_name, ModelVersion=model_version
        )

        model_exists = True

        while model_exists:
            model_exists = False
            response = client.list_models(ProjectName=project_name)
            for model in response["Models"]:
                if model["ModelVersion"] == model_version:
                    model_exists = True

            if model_exists is False:
                print("Model deleted")
            else:
                print("Model is being deleted...")
                time.sleep(2)

        print("Deleted Model: " + model_version)
        print("Done...")

    except ClientError as err:
        print("Service error: " + format(err))
        raise


def main():
    project_name = "my-project"  # The desired project
    model_version = "1"  # The version of the model that you want to delete.

    delete_model(project_name, model_version)


if __name__ == "__main__":
    main()
```

Tagging models

You can identify, organize, search for, and filter your Amazon Lookout for Vision models by using tags. Each tag is a label consisting of a user-defined key and value. For example, to help determine billing for

your models, you could tag your models with a `Cost center` key and add the appropriate cost center number as a value. For more information, see Tagging AWS resources.

Use tags to:

- Track billing for a model by using cost allocation tags. For more information, see Using Cost Allocation Tags.
- Control access to a model by using Identity and Access Management (IAM). For more information, see Controlling access to AWS resources using resource tags.
- Automate model management. For example, you can run automated start or stop scripts that turn off development models during non-business hours to reduce costs. For more information, see Running your trained Amazon Lookout for Vision model (p. 48).

You can tag models by using the Amazon Lookout for Vision console or by using the AWS SDKs.

Topics

Tagging models (console)

You can use the Amazon Lookout for Vision console to add tags to models, view the tags attached to a model, and remove tags.

Adding or removing tags (console)

This procedure explains how to add tags to, or remove tags from, an existing model. You can also add tags to a new model when it is trained. For more information, see Training your model (p. 35).

To add tags to, or remove tags from, an existing model (console)

1. Open the Amazon Lookout for Vision console at https://console.aws.amazon.com/lookoutvision/.
2. Choose **Get started**.
3. In the navigation pane, choose **Projects**.
4. On the **Projects** resources page, choose the project that contains the model that you want to tag.
5. In the navigation pane, under the project you previously chose, choose **Models**.
6. In the **Models** section, choose the model that you want to add a tag to.
7. On the model's details page, choose the **Tags** tab.
8. In the **Tags** section, choose **Manage tags**.
9. On the **Manage tags** page, choose **Add new tag**.
10. Enter a key and a value.
    a. For **Key**, enter a name for the key.
    b. For **Value**, enter a value.
11. To add more tags, repeat steps 9 and 10.
12. (Optional) To remove a tag, choose **Remove** next to the tag that you want to remove. If you are removing a previously saved tag, it is removed when you save your changes.
13. Choose **Save changes** to save your changes.

Viewing model tags (console)

You can use the Amazon Lookout for Vision console to view the tags attached to a model.

To view the tags attached to *all models within a project*, you must use the AWS SDK. For more information, see Listing model tags (SDK) (p. 81).

To view the tags attached to a model

1. Open the Amazon Lookout for Vision console at https://console.aws.amazon.com/lookoutvision/.
2. Choose **Get started**.
3. In the navigation pane, choose **Projects**.
4. On the **Projects** resources page, choose the project that contains the model whose tag you want to view.
5. In the navigation pane, under the project you previously chose, choose **Models**.
6. In the **Models** section, choose the model whose tag you want to view.
7. On the model's details page, choose the **Tags** tab. The tags are shown in **Tags** section.

Tagging models (SDK)

You can use the AWS SDK to:

- Add tags to a new model
- Add tags to an existing model
- List the tags attached to a model
- Remove tags from a model

This section includes AWS CLI examples. If you haven't installed the AWS CLI, see Step 5: Set up the AWS CLI and AWS SDKs (p. 5).

Adding tags to a new model (SDK)

You can add tags to a model when you create it using the CreateModel operation. Specify one or more tags in the `Tags` array input parameter.

```
aws lookoutvision create-model --project-name "project name"\
  --output-config '{ "S3Location": { "Bucket": "output bucket", "Prefix":  "output
 folder" } }'\
  --tags '[{"Key":"Key","Value":"Value"}]'
```

For information about creating and training a model, see Training a model (SDK) (p. 37).

Adding tags to an existing model (SDK)

To add one or more tags to an existing model, use the TagResource operation. Specify the model's Amazon Resource Name (ARN) (`ResourceArn`) and the tags (`Tags`) that you want to add.

```
aws lookoutvision tag-resource --resource-arn "resource-arn"\
  --tags '[{"Key":"Key","Value":"Value"}]'
```

Listing model tags (SDK)

To list the tags attached to a model, use the ListTagsForResource operation and specify the model's Amazon Resource Name (ARN), the (`ResourceArn`). The response is a map of tag keys and values that are attached to the specified model.

```
aws lookoutvision list-tags-for-resource --resource-arn resource-arn
```

To see which models in a project have a specific tag, call `ListModels` to get a list of models. Then call `ListTagsForResource` for each model in the response from `ListModels`. Inspect the response from `ListTagsForResource` to see if the required tag is present.

For example code that searches for a tag value across all projects, see find_tag.py.

Removing tags from a model (SDK)

To remove one or more tags from a model, use the UntagResource operation. Specify the model's Amazon Resource Name (ARN) (`ResourceArn`) and the tag keys (`Tag-Keys`) that you want to remove.

```
aws lookoutvision untag-resource --resource-arn resource-arn\
  --tag-keys '["Key"]'
```

Viewing your trial detection tasks

You can view your trial detections by using the console. You can't use the AWS SDK to view trial detection tasks.

> **Note**
> The list of trial detections is eventually consistent. If you create a trial detection, you might have to wait a short while before the trial detections list is up to date.

Viewing your trial detection tasks (console)

Use the following procedures to view your trial detections.

To view your trial detection tasks

1. Open the Amazon Lookout for Vision console at https://console.aws.amazon.com/lookoutvision/.
2. Choose **Get started**.
3. In the left navigation pane, choose **Trial detections**.
4. On the trial detections page, choose a trial detection task to view its details.

Code examples for Lookout for Vision

The following code examples show how to use Lookout for Vision with an AWS software development kit (SDK).

For a complete list of AWS SDK developer guides and code examples, including help getting started and information about previous versions, see Using Lookout for Vision with an AWS SDK (p. 13).

Code examples

Scenario examples for Lookout for Vision

The following code examples show how to use Lookout for Vision with AWS SDKs.

Examples

Create a Lookout for Vision manifest file using an AWS SDK

The following code example shows how to create a Lookout for Vision manifest file and upload it to Amazon S3.

For more information, see Creating a manifest file.

Python

SDK for Python (Boto3)

```
class Datasets:

    @staticmethod
    def create_manifest_file_s3(s3_resource, image_s3_path, manifest_s3_path):
        """
        Creates a manifest file and uploads to Amazon S3.

        :param s3_resource: A Boto3 Amazon S3 resource.
        :param image_s3_path: The Amazon S3 path to the images referenced by the
                              manifest file. The images must be in an Amazon S3
 bucket
                              with the following folder structure.
                                s3://doc-example-bucket/<train or test>/
                                    normal/
                                    anomaly/
                              Place normal images in the normal folder and
 anomalous
                              images in the anomaly folder.
        :param manifest_s3_path: The Amazon S3 location in which to store the
 created
                                 manifest file.
        """
        output_manifest_file = "temp.manifest"
        try:

            # Current date and time in manifest file format.
            dttm = datetime.now().strftime("%Y-%m-%dT%H:%M:%S.%f")

            # Get bucket and folder from image and manifest file paths.
            bucket, prefix = image_s3_path.replace("s3://", "").split("/", 1)
            if prefix[-1] != '/':
                prefix += '/'
            manifest_bucket, manifest_prefix = manifest_s3_path.replace(
                "s3://", "").split("/", 1)

            with open(output_manifest_file, "w") as mfile:
                logger.info("Creating manifest file")
                src_bucket = s3_resource.Bucket(bucket)

                # Create JSON lines for anomalous images.
                for obj in src_bucket.objects.filter(
                        Prefix=prefix + "anomaly/", Delimiter="/"):
                    image_path = f"s3://{src_bucket.name}/{obj.key}"
                    manifest = Datasets.create_json_line(image_path, "anomaly",
 dttm)
                    mfile.write(json.dumps(manifest) + "\n")

                # Create json lines for normal images.
                for obj in src_bucket.objects.filter(
```

```
                        Prefix=prefix + "normal/", Delimiter="/"):
                    image_path = f"s3://{src_bucket.name}/{obj.key}"
                    manifest = Datasets.create_json_line(image_path, "normal",
dttm)
                    mfile.write(json.dumps(manifest) + "\n")

            logger.info("Uploading manifest file to %s", manifest_s3_path)
            s3_resource.Bucket(manifest_bucket).upload_file(
                output_manifest_file, manifest_prefix)
        except ClientError:
            logger.exception("Error uploading manifest.")
            raise
        except Exception:
            logger.exception("Error uploading manifest.")
            raise
        else:
            logger.info("Completed manifest file creation and upload.")
        finally:
            try:
                os.remove(output_manifest_file)
            except FileNotFoundError:
                pass

    @staticmethod
    def create_json_line(image, class_name, dttm):
        """
        Creates a single JSON line for an image.

        :param image: The S3 location for the image.
        :param class_name: The class of the image (normal or anomaly)
        :param dttm: The date and time that the JSON is created.
        """

        label = 0
        if class_name == "normal":
            label = 0
        elif class_name == "anomaly":
            label = 1
        else:
            logger.error("Unexpected label value: %s for %s", label, image)
            raise Exception(f"Unexpected label value: {label} for {image}")

        manifest = {
            "source-ref": image,
            "anomaly-label": label,
            "anomaly-label-metadata": {
                "confidence": 1,
                "job-name": "labeling-job/anomaly-label",
                "class-name": class_name,
                "human-annotated": "yes",
                "creation-date": dttm,
                "type": "groundtruth/image-classification",
            },
        }
        return manifest
```

- Find instructions and more code on GitHub.

For a complete list of AWS SDK developer guides and code examples, including help getting started and information about previous versions, see Using Lookout for Vision with an AWS SDK (p. 13).

Create, train, and start a Lookout for Vision model using an AWS SDK

The following code example shows how to create, train, and start a Lookout for Vision model.

Python

SDK for Python (Boto3)

Creates and optionally starts an Amazon Lookout for Vision model using command line arguments. The example code creates a new project, training dataset, optional test dataset, and model. After model training is completed, you can use a provided script to try your model with an image.

This example requires a set of images to train its model. You can find example circuit board images on GitHub that you can use for training and testing. For details on how to copy these images to an Amazon Simple Storage Service (Amazon S3) bucket, see Prepare example images.

For complete source code and instructions on how to set up and run, see the full example on GitHub.

Services used in this example

- Lookout for Vision

For a complete list of AWS SDK developer guides and code examples, including help getting started and information about previous versions, see Using Lookout for Vision with an AWS SDK (p. 13).

Find a Lookout for Vision project with a specific tag using an AWS SDK

The following code example shows how to find a Lookout for Vision project with a specific tag.

For more information, see Tagging models.

Python

SDK for Python (Boto3)

```
import logging
import argparse
import boto3

from botocore.exceptions import ClientError


logger = logging.getLogger(__name__)


def find_tag(tags, key, value):
    """
    Finds a tag in the supplied list of tags.

    :param tags: A list of tags associated with a Lookout for Vision model.
    :param key: The tag to search for.
    :param value: The tag key value to search for.
```

```
    :return: True if the tag value exists, otherwise False.
    """

    found = False
    for tag in tags:
        if key == tag["Key"]:
            logger.info("\t\tMatch found for tag: %s value: %s.", key, value)
            found = True
            break
    return found


def find_tag_in_projects(lookoutvision_client, key, value):
    """
    Finds Lookout for Vision models tagged with the supplied key and value.

    :param lookoutvision_client: A Boto3 Lookout for Vision client.
    :param key: The tag key to find.
    :param value: The value of the tag that you want to find.
    return: A list of matching model versions (and model projects) that were found.
    """
    try:
        found_tags = []
        found = False
        projects = lookoutvision_client.list_projects()
        # Iterate through each project and models within a project.
        for project in projects["Projects"]:
            logger.info("Searching project: %s ...", project["ProjectName"])

            response_models = lookoutvision_client.list_models(
                ProjectName=project["ProjectName"])
            for model in response_models["Models"]:
                model_description = lookoutvision_client.describe_model(
                    ProjectName=project["ProjectName"],
                    ModelVersion=model["ModelVersion"])
                tags = lookoutvision_client.list_tags_for_resource(
                    ResourceArn=model_description["ModelDescription"]["ModelArn"])

                logger.info(
                    "\tSearching model: %s for tag: %s value: %s.",
                    model_description["ModelDescription"]["ModelArn"], key, value)
                if find_tag(tags["Tags"], key, value) is True:
                    found = True
                    logger.info(
                        "\t\tMATCH: Project: %s: model version %s",
                        project["ProjectName"],
                        model_description["ModelDescription"]["ModelVersion"])
                    found_tags.append({
                        "Project": project["ProjectName"],
                        "ModelVersion":
                            model_description["ModelDescription"]["ModelVersion"]})
        if found is False:
            logger.info("No match for tag %s with value %s.", key, value)
    except ClientError:
        logger.exception("Problem finding tags.")
        raise
    else:
        return found_tags


def main():
    logging.basicConfig(level=logging.INFO, format="%(levelname)s: %(message)s")
    parser = argparse.ArgumentParser(usage=argparse.SUPPRESS)
    parser.add_argument("tag", help="The tag that you want to find.")
    parser.add_argument("value", help="The tag value that you want to find.")
    args = parser.parse_args()
```

```
    key = args.tag
    value = args.value
    lookoutvision_client = boto3.client("lookoutvision")

    print(f"Searching your models for tag: {key} with value: {value}.")
    tagged_models = find_tag_in_projects(lookoutvision_client, key, value)

    print("Matched models\n--------------")
    if len(tagged_models) > 0:
        for model in tagged_models:
            print(f"Project: {model['Project']}. model version:
{model['ModelVersion']}")
    else:
        print("No matches found.")


if __name__ == "__main__":
    main()
```

- Find instructions and more code on GitHub.

For a complete list of AWS SDK developer guides and code examples, including help getting started and information about previous versions, see Using Lookout for Vision with an AWS SDK (p. 13).

List Lookout for Vision models that are currently hosted using an AWS SDK

The following code example shows how to list Lookout for Vision models that are currently hosted.

Python

SDK for Python (Boto3)

```
class Hosting:

    @staticmethod
    def list_hosted(lookoutvision_client):
        """
        Displays a list of models in your account that are currently hosted.

        :param lookoutvision_client: A Boto3 Lookout for Vision client.
        """
        try:
            response = lookoutvision_client.list_projects()
            hosted = 0
            print("Hosted models\n-------------")

            for project in response["Projects"]:
                response_models = lookoutvision_client.list_models(
                    ProjectName=project["ProjectName"])

                for model in response_models["Models"]:
                    model_description = lookoutvision_client.describe_model(
                        ProjectName=project["ProjectName"],
                        ModelVersion=model["ModelVersion"])

                    if model_description["ModelDescription"]["Status"] == "HOSTED":
                        print(
                            f"Project: {project['ProjectName']} Model version: "
```

```
                            f"{model['ModelVersion']}")
                    hosted += 1
            print(f"{hosted} model(s) hosted")
        except ClientError:
            logger.exception("Problem listing hosted models.")
            raise
```

- Find instructions and more code on GitHub.

For a complete list of AWS SDK developer guides and code examples, including help getting started and information about previous versions, see Using Lookout for Vision with an AWS SDK (p. 13).

API examples for Lookout for Vision

The following code examples show how to use Lookout for Vision with AWS SDKs.

Examples

- Create a Lookout for Vision dataset using an AWS SDK (p. 88)
- Create a Lookout for Vision model using an AWS SDK (p. 90)
- Create a Lookout for Vision project using an AWS SDK (p. 91)
- Delete a Lookout for Vision dataset using an AWS SDK (p. 92)
- Delete a Lookout for Vision model using an AWS SDK (p. 92)
- Delete a Lookout for Vision project using an AWS SDK (p. 93)
- Describe a Lookout for Vision dataset using an AWS SDK (p. 94)
- Describe a Lookout for Vision model using an AWS SDK (p. 95)
- Detect anomalies in an image with a trained Lookout for Vision model using an AWS SDK (p. 96)
- List Lookout for Vision models using an AWS SDK (p. 98)
- List Lookout for Vision projects using an AWS SDK (p. 99)
- Start a Lookout for Vision model using an AWS SDK (p. 100)
- Stop a Lookout for Vision model using an AWS SDK (p. 101)

Create a Lookout for Vision dataset using an AWS SDK

The following code example shows how to create a Lookout for Vision dataset.

For more information, see Creating your dataset.

Python

SDK for Python (Boto3)

```
class Datasets:

    @staticmethod
    def create_dataset(lookoutvision_client, project_name, manifest_file,
 dataset_type):
        """
```

```
        Creates a new Lookout for Vision dataset

        :param lookoutvision_client: A Lookout for Vision Boto3 client.
        :param project_name: The name of the project in which you want to
                             create a dataset.
        :param bucket: The bucket that contains the manifest file.
        :param manifest_file: The path and name of the manifest file.
        :param dataset_type: The type of the dataset (train or test).
        """
        try:

            bucket, key = manifest_file.replace("s3://", "").split("/", 1)
            logger.info("Creating %s dataset type...", dataset_type)
            dataset = {
                "GroundTruthManifest": {"S3Object": {"Bucket": bucket, "Key": key}}
            }
            response = lookoutvision_client.create_dataset(
                ProjectName=project_name,
                DatasetType=dataset_type,
                DatasetSource=dataset,
            )
            logger.info("Dataset Status: %s", response["DatasetMetadata"]
["Status"])
            logger.info(
                "Dataset Status Message: %s",
                response["DatasetMetadata"]["StatusMessage"],
            )
            logger.info("Dataset Type: %s", response["DatasetMetadata"]
["DatasetType"])

            # Wait until either created or failed.
            finished = False
            status = ""
            dataset_description = {}
            while finished is False:
                dataset_description = lookoutvision_client.describe_dataset(
                    ProjectName=project_name, DatasetType=dataset_type
                )
                status = dataset_description["DatasetDescription"]["Status"]

                if status == "CREATE_IN_PROGRESS":
                    logger.info("Dataset creation in progress...")
                    time.sleep(2)
                elif status == "CREATE_COMPLETE":
                    logger.info("Dataset created.")
                    finished = True
                else:
                    logger.info(
                        "Dataset creation failed: %s",
                        dataset_description["DatasetDescription"]["StatusMessage"])
                    finished = True

            if status != "CREATE_COMPLETE":
                message = dataset_description["DatasetDescription"]
["StatusMessage"]
                logger.exception("Couldn't create dataset: %s", message)
                raise Exception(f"Couldn't create dataset: {message}")

        except ClientError:
            logger.exception("Service error: Couldn't create dataset.")
            raise
```

- Find instructions and more code on GitHub.
- For API details, see CreateDataset in *AWS SDK for Python (Boto3) API Reference*.

For a complete list of AWS SDK developer guides and code examples, including help getting started and information about previous versions, see Using Lookout for Vision with an AWS SDK (p. 13).

Create a Lookout for Vision model using an AWS SDK

The following code example shows how to create a Lookout for Vision model.

For more information, see Training your model.

Python

SDK for Python (Boto3)

```
class Models:

    @staticmethod
    def create_model(
            lookoutvision_client, project_name, training_results, tag_key=None,
            tag_key_value=None):
        """
        Creates a version of a Lookout for Vision model.

        :param lookoutvision_client: A Boto3 Lookout for Vision client.
        :param project_name: The name of the project in which you want to create a
                             model.
        :param training_results: The Amazon S3 location where training results are
 stored.
        :param tag_key: The key for a tag to add to the model.
        :param tag_key_value - A value associated with the tag_key.
        return: The model status and version.
        """
        try:
            logger.info("Training model...")
            output_bucket, output_folder = training_results.replace(
                "s3://", "").split("/", 1)
            output_config = {
                "S3Location": {"Bucket": output_bucket, "Prefix": output_folder}}
            tags = []
            if tag_key is not None:
                tags = [{"Key": tag_key, "Value": tag_key_value}]

            response = lookoutvision_client.create_model(
                ProjectName=project_name, OutputConfig=output_config, Tags=tags)

            logger.info("ARN: %s", response["ModelMetadata"]["ModelArn"])
            logger.info("Version: %s", response["ModelMetadata"]["ModelVersion"])
            logger.info("Started training...")

            print("Training started. Training might take several hours to
 complete.")

            # Wait until training completes.
            finished = False
            status = "UNKNOWN"
            while finished is False:
                model_description = lookoutvision_client.describe_model(
                    ProjectName=project_name,
                    ModelVersion=response["ModelMetadata"]["ModelVersion"])
                status = model_description["ModelDescription"]["Status"]

                if status == "TRAINING":
                    logger.info("Model training in progress...")
```

```
                    time.sleep(600)
                    continue

                if status == "TRAINED":
                    logger.info("Model was successfully trained.")
                else:
                    logger.info(
                        "Model training failed: %s ",
                        model_description["ModelDescription"]["StatusMessage"])
                finished = True
        except ClientError:
            logger.exception("Couldn't train model.")
            raise
        else:
            return status, response["ModelMetadata"]["ModelVersion"]
```

- Find instructions and more code on GitHub.
- For API details, see CreateModel in *AWS SDK for Python (Boto3) API Reference*.

For a complete list of AWS SDK developer guides and code examples, including help getting started and information about previous versions, see Using Lookout for Vision with an AWS SDK (p. 13).

Create a Lookout for Vision project using an AWS SDK

The following code example shows how to create a Lookout for Vision project.

For more information, see Creating your project.

Python

SDK for Python (Boto3)

```
class Projects:

    @staticmethod
    def create_project(lookoutvision_client, project_name):
        """
        Creates a new Lookout for Vision project.

        :param lookoutvision_client: A Boto3 Lookout for Vision client.
        :param  project_name: The name for the new project.
        :return project_arn: The ARN of the new project.
        """
        try:
            logger.info("Creating project: %s", project_name)
            response =
 lookoutvision_client.create_project(ProjectName=project_name)
            project_arn = response["ProjectMetadata"]["ProjectArn"]
            logger.info("project ARN: %s", project_arn)
        except ClientError:
            logger.exception("Couldn't create project %s.", project_name)
            raise
        else:
            return project_arn
```

- Find instructions and more code on GitHub.
- For API details, see CreateProject in *AWS SDK for Python (Boto3) API Reference*.

For a complete list of AWS SDK developer guides and code examples, including help getting started and information about previous versions, see Using Lookout for Vision with an AWS SDK (p. 13).

Delete a Lookout for Vision dataset using an AWS SDK

The following code example shows how to delete a Lookout for Vision dataset.

For more information, see Deleting a dataset.

Python

SDK for Python (Boto3)

```
class Datasets:

    @staticmethod
    def delete_dataset(lookoutvision_client, project_name, dataset_type):
        """
        Deletes a Lookout for Vision dataset

        :param lookoutvision_client: A Boto3 Lookout for Vision client.
        :param project_name: The name of the project that contains the dataset that
                             you want to delete.
        :param dataset_type: The type (train or test) of the dataset that you
                             want to delete.
        """
        try:
            logger.info(
                "Deleting the %s dataset for project %s.", dataset_type,
 project_name)
            lookoutvision_client.delete_dataset(
                ProjectName=project_name, DatasetType=dataset_type)
            logger.info("Dataset deleted.")
        except ClientError:
            logger.exception("Service error: Couldn't delete dataset.")
            raise
```

- Find instructions and more code on GitHub.
- For API details, see DeleteDataset in *AWS SDK for Python (Boto3) API Reference*.

For a complete list of AWS SDK developer guides and code examples, including help getting started and information about previous versions, see Using Lookout for Vision with an AWS SDK (p. 13).

Delete a Lookout for Vision model using an AWS SDK

The following code example shows how to delete a Lookout for Vision model.

For more information, see Deleting a model.

Python

SDK for Python (Boto3)

```
class Models:
```

```
    @staticmethod
    def delete_model(lookoutvision_client, project_name, model_version):
        """
        Deletes a Lookout for Vision model. The model must first be stopped and
can't
        be in training.

        :param lookoutvision_client: A Boto3 Lookout for Vision client.
        :param project_name: The name of the project that contains the desired
model.
        :param model_version: The version of the model that you want to delete.
        """
        try:
            logger.info("Deleting model: %s", model_version)
            lookoutvision_client.delete_model(
                ProjectName=project_name, ModelVersion=model_version)

            model_exists = True
            while model_exists:
                response =
lookoutvision_client.list_models(ProjectName=project_name)

                model_exists = False
                for model in response["Models"]:
                    if model["ModelVersion"] == model_version:
                        model_exists = True

                if model_exists is False:
                    logger.info("Model deleted")
                else:
                    logger.info("Model is being deleted...")
                    time.sleep(2)

            logger.info("Deleted Model: %s", model_version)
        except ClientError:
            logger.exception("Couldn't delete model.")
            raise
```

- Find instructions and more code on GitHub.
- For API details, see DeleteModel in *AWS SDK for Python (Boto3) API Reference*.

For a complete list of AWS SDK developer guides and code examples, including help getting started and information about previous versions, see Using Lookout for Vision with an AWS SDK (p. 13).

Delete a Lookout for Vision project using an AWS SDK

The following code example shows how to delete a Lookout for Vision project.

For more information, see Deleting a project.

Python

SDK for Python (Boto3)

```
class Projects:

    @staticmethod
    def delete_project(lookoutvision_client, project_name):
        """
        Deletes a Lookout for Vision Model
```

```
        :param lookoutvision_client: A Boto3 Lookout for Vision client.
        :param project_name: The name of the project that you want to delete.
        """
        try:
            logger.info("Deleting project: %s", project_name)
            response =
 lookoutvision_client.delete_project(ProjectName=project_name)
            logger.info("Deleted project ARN: %s ", response["ProjectArn"])
        except ClientError as err:
            logger.exception("Couldn't delete project %s.", project_name)
            raise
```

- Find instructions and more code on GitHub.
- For API details, see DeleteProject in *AWS SDK for Python (Boto3) API Reference*.

For a complete list of AWS SDK developer guides and code examples, including help getting started and information about previous versions, see Using Lookout for Vision with an AWS SDK (p. 13).

Describe a Lookout for Vision dataset using an AWS SDK

The following code example shows how to describe a Lookout for Vision dataset.

For more information, see Viewing your dataset.

Python

SDK for Python (Boto3)

```
class Datasets:

    @staticmethod
    def describe_dataset(lookoutvision_client, project_name, dataset_type):
        """
        Gets information about a Lookout for Vision dataset.

        :param lookoutvision_client: A Boto3 Lookout for Vision client.
        :param project_name: The name of the project that contains the dataset that
                             you want to describe.
        :param dataset_type: The type (train or test) of the dataset that you want
                             to describe.
        """
        try:
            response = lookoutvision_client.describe_dataset(
                ProjectName=project_name, DatasetType=dataset_type)
            print(f"Name: {response['DatasetDescription']['ProjectName']}")
            print(f"Type: {response['DatasetDescription']['DatasetType']}")
            print(f"Status: {response['DatasetDescription']['Status']}")
            print(f"Message: {response['DatasetDescription']['StatusMessage']}")
            print(
                f"Images: {response['DatasetDescription']['ImageStats']['Total']}")
            print(
                f"Labeled: {response['DatasetDescription']['ImageStats']
['Labeled']}")
            print(
                f"Normal: {response['DatasetDescription']['ImageStats']
['Normal']}")
```

```
            print(
                f"Anomaly: {response['DatasetDescription']['ImageStats']
['Anomaly']}")
        except ClientError:
            logger.exception("Service error: problem listing datasets.")
            raise
        print("Done.")
```

- Find instructions and more code on GitHub.
- For API details, see DescribeDataset in *AWS SDK for Python (Boto3) API Reference*.

For a complete list of AWS SDK developer guides and code examples, including help getting started and information about previous versions, see Using Lookout for Vision with an AWS SDK (p. 13).

Describe a Lookout for Vision model using an AWS SDK

The following code example shows how to describe a Lookout for Vision model.

For more information, see Viewing your models.

Python

SDK for Python (Boto3)

```
class Models:

    @staticmethod
    def describe_model(lookoutvision_client, project_name, model_version):
        """
        Shows the performance metrics for a trained model.

        :param lookoutvision_client: A Boto3 Amazon Lookout for Vision client.
        :param project_name: The name of the project that contains the desired
 model.
        :param model_version: The version of the model.
        """
        response = lookoutvision_client.describe_model(
            ProjectName=project_name, ModelVersion=model_version)
        model_description = response["ModelDescription"]
        print(f"\tModel version: {model_description['ModelVersion']}")
        print(f"\tARN: {model_description['ModelArn']}")
        if "Description" in model_description:
            print(f"\tDescription: {model_description['Description']}")
        print(f"\tStatus: {model_description['Status']}")
        print(f"\tMessage: {model_description['StatusMessage']}")
        print(f"\tCreated: {str(model_description['CreationTimestamp'])}")

        if model_description['Status'] in ("TRAINED", "HOSTED"):
            training_start = model_description["CreationTimestamp"]
            training_end = model_description["EvaluationEndTimestamp"]
            duration = training_end - training_start
            print(f"\tTraining duration: {duration}")

            print("\n\tPerformance metrics\n\t-------------------")
            print(f"\tRecall: {model_description['Performance']['Recall']}")
            print(f"\tPrecision: {model_description['Performance']['Precision']}")
            print(f"\tF1: {model_description['Performance']['F1Score']}")
```

```
            training_output_bucket = model_description["OutputConfig"]
["S3Location"]["Bucket"]
            prefix = model_description["OutputConfig"]["S3Location"]["Prefix"]
            print(f"\tTraining output: s3://{training_output_bucket}/{prefix}")
```

- Find instructions and more code on GitHub.
- For API details, see DescribeModel in *AWS SDK for Python (Boto3) API Reference*.

For a complete list of AWS SDK developer guides and code examples, including help getting started and information about previous versions, see Using Lookout for Vision with an AWS SDK (p. 13).

Detect anomalies in an image with a trained Lookout for Vision model using an AWS SDK

The following code example shows how to detect anomalies in an image with a trained Lookout for Vision model.

For more information, see Detecting anomalies in an image.

Python

SDK for Python (Boto3)

```
class Inference:
    """
    Shows how to detect anomalies in an image using a trained Lookout for Vision
 model.
    """

    @staticmethod
    def detect_anomalies(lookoutvision_client, project_name, model_version, photo):
        """
        Detects anomalies in an image (jpg/png) by using your Lookout for Vision
 model.

        :param lookoutvision_client: A Boto3 Lookout for Vision client.
        :param project_name: The name of the project that contains the model that
                             you want to use.
        :param model_version: The version of the model that you want to use.
        :param photo: The path and name of the image in which you want to detect
                      anomalies.
        :return: Whether anomalies were detected and with what confidence.
        """
        try:
            image_type = imghdr.what(photo)
            if image_type == "jpeg":
                content_type = "image/jpeg"
            elif image_type == "png":
                content_type = "image/png"
            else:
                logger.info("Invalid image type for %s", photo)
                raise ValueError(
                    f"Invalid file format. Supply a jpeg or png format file:
 {photo}")

            logger.info("Detecting anomalies in %s", photo)
            with open(photo, "rb") as image:
```

```
                response = lookoutvision_client.detect_anomalies(
                    ProjectName=project_name,
                    ContentType=content_type,
                    Body=image.read(),
                    ModelVersion=model_version)
            anomalous = response["DetectAnomalyResult"]["IsAnomalous"]
            confidence = response["DetectAnomalyResult"]["Confidence"]

            logger.info("Anomalous?: %s", anomalous)
            logger.info("Confidence: %s", confidence)
        except FileNotFoundError:
            logger.exception("Couldn't find file: %s", photo)
            raise
        except ClientError:
            logger.exception("Couldn't detect anomalies.")
            raise
        else:
            return anomalous, confidence

    @staticmethod
    def download_from_s3(s3_resource, photo):
        """
        Downloads an image from an S3 bucket.

        :param s3_resource: A Boto3 Amazon S3 resource.
        :param photo: The Amazon S3 path of a photo to download.
        return: The local path to the downloaded file.
        """
        try:
            bucket, key = photo.replace("s3://", "").split("/", 1)
            local_file = os.path.basename(photo)
        except ValueError as err:
            logger.exception("Couldn't get S3 info for %s: %s", photo)
            raise

        try:
            logger.info("Downloading %s", photo)
            s3_resource.Bucket(bucket).download_file(key, local_file)
        except ClientError:
            logger.exception("Couldn't download %s from S3.", photo)
            raise

        return local_file


def main():
    """
    Detects anomalies in an image file.
    """
    try:
        logging.basicConfig(level=logging.INFO, format="%(levelname)s:
 %(message)s")
        lookoutvision_client = boto3.client("lookoutvision")
        s3_resource = boto3.resource('s3')

        parser = argparse.ArgumentParser(usage=argparse.SUPPRESS)
        parser.add_argument(
            "project", help="The project containing the model that you want to
 use.")
        parser.add_argument(
            "version", help="The version of the model that you want to use.")
        parser.add_argument(
            "image",
            help="The file that you want to analyze. Supply a local file path or a
 "
                 "path to an S3 object.")
```

```
        args = parser.parse_args()

        if args.image.startswith("s3://"):
            photo = Inference.download_from_s3(s3_resource, args.image)
        else:
            photo = args.image

        print(f"Analyzing {photo}.")
        anomalous, confidence = Inference.detect_anomalies(
            lookoutvision_client, args.project, args.version, photo)

        if args.image.startswith("s3://"):
            os.remove(photo)

        state = "anomalous" if anomalous else "normal"
        print(
            f"Your model is {confidence:.0%} confident that the image is {state}.")
    except ClientError as err:
        print(f"Service error: {err.response['Error']['Message']}")
    except FileNotFoundError as err:
        print(f"The supplied file couldn't be found: {err.filename}.")
    except ValueError as err:
        print(f"A value error occurred: {err}.")
    else:
        print("Successfully completed analysis.")


if __name__ == "__main__":
    main()
```

- Find instructions and more code on GitHub.
- For API details, see DetectAnomalies in *AWS SDK for Python (Boto3) API Reference*.

For a complete list of AWS SDK developer guides and code examples, including help getting started and information about previous versions, see Using Lookout for Vision with an AWS SDK (p. 13).

List Lookout for Vision models using an AWS SDK

The following code example shows how to list Lookout for Vision models.

For more information, see Viewing your models.

Python

SDK for Python (Boto3)

```
class Models:

    @staticmethod
    def describe_models(lookoutvision_client, project_name):
        """
        Gets information about all models in a Lookout for Vision project.

        :param lookoutvision_client: A Boto3 Lookout for Vision client.
        :param project_name: The name of the project that you want to use.
        """
        try:
            response = lookoutvision_client.list_models(ProjectName=project_name)
            print("Project: " + project_name)
            for model in response["Models"]:
```

```
                Models.describe_model(
                    lookoutvision_client, project_name, model["ModelVersion"])
                print()
            print("Done...")
        except ClientError:
            logger.exception("Couldn't list models.")
            raise
```

- Find instructions and more code on GitHub.
- For API details, see ListModels in *AWS SDK for Python (Boto3) API Reference*.

For a complete list of AWS SDK developer guides and code examples, including help getting started and information about previous versions, see Using Lookout for Vision with an AWS SDK (p. 13).

List Lookout for Vision projects using an AWS SDK

The following code example shows how to list Lookout for Vision projects.

For more information, see Viewing your projects.

Python

SDK for Python (Boto3)

```
class Projects:

    @staticmethod
    def list_projects(lookoutvision_client):
        """
        Lists information about the projects in your account.

        :param lookoutvision_client: A Boto3 Lookout for Vision client.
        """
        try:
            response = lookoutvision_client.list_projects()
            for project in response["Projects"]:
                print("Project: " + project["ProjectName"])
                print("\tARN: " + project["ProjectArn"])
                print("\tCreated: " + str(["CreationTimestamp"]))
                print("Datasets")
                project_description = lookoutvision_client.describe_project(
                    ProjectName=project["ProjectName"])
                if not project_description["ProjectDescription"]["Datasets"]:
                    print("\tNo datasets")
                else:
                    for dataset in project_description["ProjectDescription"]
["Datasets"]:
                        print(f"\ttype: {dataset['DatasetType']}")
                        print(f"\tStatus: {dataset['StatusMessage']}")

                print("Models")
                response_models = lookoutvision_client.list_models(
                    ProjectName=project["ProjectName"])
                if not response_models["Models"]:
                    print("\tNo models")
                else:
                    for model in response_models["Models"]:
                        Models.describe_model(
                            lookoutvision_client, project["ProjectName"],
                            model["ModelVersion"])
```

```
                print("-----------------------------------------------------------
\n")
            print("Done!")
        except ClientError:
            logger.exception("Problem listing projects.")
            raise
```

- Find instructions and more code on GitHub.
- For API details, see ListProjects in *AWS SDK for Python (Boto3) API Reference*.

For a complete list of AWS SDK developer guides and code examples, including help getting started and information about previous versions, see Using Lookout for Vision with an AWS SDK (p. 13).

Start a Lookout for Vision model using an AWS SDK

The following code example shows how to start a Lookout for Vision model.

For more information, see Starting your model.

Python

SDK for Python (Boto3)

```
class Hosting:

    @staticmethod
    def start_model(
            lookoutvision_client, project_name, model_version,
 min_inference_units):
        """
        Starts the hosting of a Lookout for Vision model.

        :param lookoutvision_client: A Boto3 Lookout for Vision client.
        :param project_name:  The name of the project that contains the version of
 the
                              model that you want to start hosting.
        :param model_version: The version of the model that you want to start
 hosting.
        :param min_inference_units: The number of inference units to use for
 hosting.
        """
        try:
            logger.info(
                "Starting model version %s for project %s", model_version,
 project_name)
            lookoutvision_client.start_model(
                ProjectName=project_name,
                ModelVersion=model_version,
                MinInferenceUnits=min_inference_units)
            print("Starting hosting...")

            status = ""
            finished = False

            # Wait until hosted or failed.
            while finished is False:
                model_description = lookoutvision_client.describe_model(
                    ProjectName=project_name, ModelVersion=model_version)
                status = model_description["ModelDescription"]["Status"]
```

```
                if status == "STARTING_HOSTING":
                    logger.info("Host starting in progress...")
                    time.sleep(10)
                    continue

                if status == "HOSTED":
                    logger.info("Model is hosted and ready for use.")
                    finished = True
                    continue

                logger.info("Model hosting failed and the model can't be used.")
                finished = True

            if status != "HOSTED":
                logger.error("Error hosting model: %s", status)
                raise Exception(f"Error hosting model: {status}")
        except ClientError:
            logger.exception("Couldn't host model.")
            raise
```

- Find instructions and more code on GitHub.
- For API details, see StartModel in *AWS SDK for Python (Boto3) API Reference*.

For a complete list of AWS SDK developer guides and code examples, including help getting started and information about previous versions, see Using Lookout for Vision with an AWS SDK (p. 13).

Stop a Lookout for Vision model using an AWS SDK

The following code example shows how to stop a Lookout for Vision model.

For more information, see Stopping your model.

Python

SDK for Python (Boto3)

```
class Hosting:

    @staticmethod
    def stop_model(lookoutvision_client, project_name, model_version):
        """
        Stops a running Lookout for Vision Model.

        :param lookoutvision_client: A Boto3 Lookout for Vision client.
        :param project_name: The name of the project that contains the version of
                             the model that you want to stop hosting.
        :param model_version:  The version of the model that you want to stop
 hosting.
        """
        try:
            logger.info("Stopping model version %s for %s", model_version,
 project_name)
            response = lookoutvision_client.stop_model(
                ProjectName=project_name, ModelVersion=model_version)
            logger.info("Stopping hosting...")

            status = response["Status"]
            finished = False
```

```
            # Wait until stopped or failed.
            while finished is False:
                model_description = lookoutvision_client.describe_model(
                    ProjectName=project_name, ModelVersion=model_version)
                status = model_description["ModelDescription"]["Status"]

                if status == "STOPPING_HOSTING":
                    logger.info("Host stopping in progress...")
                    time.sleep(10)
                    continue

                if status == "TRAINED":
                    logger.info("Model is no longer hosted.")
                    finished = True
                    continue

                logger.info("Failed to stop model: %s ", status)
                finished = True

            if status != "TRAINED":
                logger.error("Error stopping model: %s", status)
                raise Exception(f"Error stopping model: {status}")
        except ClientError:
            logger.exception("Couldn't stop hosting model.")
            raise
```

- Find instructions and more code on GitHub.
- For API details, see StopModel in *AWS SDK for Python (Boto3) API Reference*.

For a complete list of AWS SDK developer guides and code examples, including help getting started and information about previous versions, see Using Lookout for Vision with an AWS SDK (p. 13).

Security in Amazon Lookout for Vision

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

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

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

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

Topics

Data protection in Amazon Lookout for Vision

The AWS shared responsibility model applies to data protection in Amazon Lookout for Vision. 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 sensitive identifying information, such as your customers' account numbers, into free-form fields such as a **Name** field. This includes when you work with Lookout for Vision or other AWS services using the console, API, AWS CLI, or AWS SDKs. Any data that you enter into Lookout for Vision or other services might get picked up for inclusion in diagnostic logs. When you provide a URL to an external server, don't include credentials information in the URL to validate your request to that server.

Data encryption

The following information explains where Amazon Lookout for Vision uses data encryption to protect your data.

Encryption at rest

Images

To train your model, Amazon Lookout for Vision makes a copy of your source training and test images. The copied images are encrypted at rest in Amazon Simple Storage Service (S3) using server-side encryption with an AWS owned key or a key that you provide. The keys are stored using AWS Key Management Service (SSE-KMS). Your source images are unaffected. For more information, see Training your model (p. 35).

Amazon Lookout for Vision models

By default, trained models and manifest files are encrypted in Amazon S3 using server-side encryption with KMS keys stored in AWS Key Management Service (SSE-KMS). Lookout for Vision uses an AWS owned key. For more information, see Protecting Data Using Server-Side Encryption. Training results are written to the bucket specified in the `output_bucket` input parameter to `CreateModel`. The training results are encrypted using the configured encryption settings for the bucket (`output_bucket`).

Amazon Lookout for Vision console bucket

The Amazon Lookout for Vision console creates an Amazon S3 bucket (console bucket) that you can use to manage your projects. The console bucket is encrypted using the default Amazon S3 encryption. For more information, see Amazon Simple Storage Service default encryption for S3 buckets. If you are using your own KMS key, configure the console bucket after it is created. For more information, see Protecting Data Using Server-Side Encryption. Amazon Lookout for Vision blocks public access to the console bucket.

Encryption in transit

Amazon Lookout for Vision API endpoints only support secure connections over HTTPS. All communication is encrypted with Transport Layer Security (TLS).

Key management

You can use AWS Key Management Service (KMS) to manage encryption for the input images that you store in Amazon S3 buckets. For more information, see Step 7: (Optional) Using your own AWS Key Management Service key (p. 8).

By default your images are encrypted with a key that AWS owns and manages. You can also choose to use your own AWS Key Management Service (KMS) key. For more information, see AWS Key Management Service concepts.

Internetwork traffic privacy

An Amazon Virtual Private Cloud (Amazon VPC) endpoint for Amazon Lookout for Vision is a logical entity within a VPC that allows connectivity only to Amazon Lookout for Vision. Amazon VPC routes requests to Amazon Lookout for Vision and routes responses back to the VPC. For more information, see VPC Endpoints in the *Amazon VPC User Guide*. For information about using Amazon VPC endpoints with Amazon Lookout for Vision see Amazon Lookout for Vision and interface VPC endpoints (AWS PrivateLink) (p. 125).

Identity and access management for Amazon Lookout for Vision

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 Lookout for Vision resources. IAM is an AWS service that you can use with no additional charge.

Topics

Audience

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

Service user – If you use the Lookout for Vision service to do your job, then your administrator provides you with the credentials and permissions that you need. As you use more Lookout for Vision 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 Lookout for Vision, see Troubleshooting Amazon Lookout for Vision identity and access (p. 119).

Service administrator – If you're in charge of Lookout for Vision resources at your company, you probably have full access to Lookout for Vision. It's your job to determine which Lookout for Vision 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 Lookout for Vision, see How Amazon Lookout for Vision works with IAM (p. 109).

IAM administrator – If you're an IAM administrator, you might want to learn details about how you can write policies to manage access to Lookout for Vision. To view example Lookout for Vision identity-based policies that you can use in IAM, see Amazon Lookout for Vision identity-based policy examples (p. 111).

Authenticating with identities

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

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

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

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

AWS account root user

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

IAM users and groups

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

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

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

IAM roles

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

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

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

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 Lookout for Vision works with IAM

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

Topics

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

Actions

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

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

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

Policy actions in Lookout for Vision use the following prefix before the action: `lookoutvision:`. For example, to grant someone permission to create an Amazon Lookout for Vision project `CreateProject` API operation, you include the `lookoutvision:CreateProject` action in their policy. Policy statements must include either an `Action` or `NotAction` element. Lookout for Vision 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": [
      "lookoutvision:action1",
      "lookoutvision: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": "lookoutvision:Describe*"
```

To see a list of Lookout for Vision actions, see Actions Defined by Amazon Lookout for Vision 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": "*"
```

For example, the Amazon Lookout for Vision project resource has the following ARN:

```
arn:${Partition}:lookoutvision:${Region}:${Account}:project/${ProjectName}
```

For more information about the format of ARNs, see Amazon Resource Names (ARNs) and AWS Service Namespaces.

Condition keys

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

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

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

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

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

Lookout for Vision 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*.

All Lookout for Vision actions support the `aws:RequestedRegion` condition key.

Examples

To view examples of Lookout for Vision identity-based policies, see Amazon Lookout for Vision identity-based policy examples (p. 111).

Lookout for Vision IAM roles

An IAM role is an entity within your AWS account that has specific permissions.

Using temporary credentials with Lookout for Vision

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.

Lookout for Vision supports using temporary credentials.

Amazon Lookout for Vision identity-based policy examples

By default, IAM users and roles don't have permission to create or modify Lookout for Vision 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

Identity-based policies are very powerful. They determine whether someone can create, access, or delete Lookout for Vision 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 Lookout for Vision 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*.

Accessing a single Amazon Lookout for Vision project

In this example, you want to grant an IAM user in your AWS account access to one of your Amazon Lookout for Vision projects.

```
{
    "Sid": "SpecificProjectOnly",
    "Effect": "Allow",
    "Action": [
        "lookoutvision:DetectAnomalies"
    ],
    "Resource": "arn:aws:lookoutvision:us-east-1:123456789012:model/myproject/*"
}
```

Tag-based policy examples

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

Use a tag to access a resource

This example policy grants an IAM user or role in your AWS account permission to use the `DetectAnomalies` operation with any model tagged with the key `stage` and the value `production`.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "LookoutVision:DetectAnomalies"
            ],
            "Resource": "*",
            "Condition": {
                "StringEquals": {
                    "aws:ResourceTag/stage": "production"
                }
            }
        }
   ]
}
```

Use a tag to deny access to specific Amazon Lookout for Vision operations

This example policy denies permission for an IAM user or role in your AWS account to call the `DeleteModel` or `StopModel` operations with any model tagged with the key `stage` and the value `production`.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Deny",
            "Action": [
                "LookoutVision:DeleteModel",
                "LookoutVision:StopModel"
            ],
            "Resource": "*",
            "Condition": {
                "StringEquals": {
                    "aws:ResourceTag/stage": "production"
                }
            }
        }
   ]
```

```
}
```

AWS managed policies for Amazon Lookout for Vision

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

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

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

AWS managed policy: AmazonLookoutVisionReadOnlyAccess

Use the `AmazonLookoutVisionReadOnlyAccess` policy to allow users read-only access to Amazon Lookout for Vision (and its dependencies) with the following Amazon Lookout for Vision actions (SDK operations). For example, you can use `DescribeModel` to get information about an existing model.

- DescribeDataset
- DescribeModel
- DescribeProject
- ListDatasetEntries
- ListModels
- ListProjects
- ListTagsForResource

To call read-only actions, users don't need Amazon S3 bucket permissions. However, operation responses might include references to Amazon S3 buckets. For example, the `source-ref` entry in the response from `ListDatasetEntries` is a reference to an image in an Amazon S3 bucket. Add Amazon S3 bucket permissions if your users need to access referenced buckets. For example, a user might want to download an image referenced by a `source-ref` field. For more information, see Setting Amazon S3 Bucket permissions (p. 6).

You can attach the `AmazonLookoutVisionReadOnlyAccess` policy to your IAM identities.

Permissions details

This policy includes the following permissions.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "LookoutVisionReadOnlyAccess",
            "Effect": "Allow",
            "Action": [
                "lookoutvision:DescribeDataset",
                "lookoutvision:DescribeModel",
                "lookoutvision:DescribeProject",
                "lookoutvision:ListDatasetEntries",
                "lookoutvision:ListModels",
                "lookoutvision:ListProjects",
                "lookoutvision:ListTagsForResource"
            ],
            "Resource": "*"
        }
    ]
}
```

AWS managed policy: AmazonLookoutVisionFullAccess

Use the `AmazonLookoutVisionFullAccess` policy to allow users full access to Amazon Lookout for Vision (and its dependencies) with Amazon Lookout for Vision actions (SDK operations). For example, you can train a model without having to use the Amazon Lookout for Vision console. For more information, see Actions.

To create a dataset (`CreateDataset`) or create a model (`CreateModel`), your users must have full access permissions to the Amazon S3 bucket that stores dataset images, Amazon SageMaker Ground Truth manifest files, and training output. For more information, see Step 3: Set up permissions (p. 3).

You can also give permission to Amazon Lookout for Vision SDK actions by using the `AmazonLookoutVisionConsoleFullAccess` policy.

You can attach the `AmazonLookoutVisionFullAccess` policy to your IAM identities.

Permissions details

This policy includes the following permissions.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "LookoutVisionFullAccess",
            "Effect": "Allow",
            "Action": [
                "lookoutvision:*"
            ],
```

```
            "Resource": "*"
        }
    ]
}
```

AWS managed policy: AmazonLookoutVisionConsoleFullAccess

Use the `AmazonLookoutVisionFullAccess` policy to allow users full access to the Amazon Lookout for Vision console, actions (SDK operations), and any dependencies that the service has. For more information, see Getting Started with the Amazon Lookout for Vision console (p. 10).

The `LookoutVisionConsoleFullAccess` policy includes permissions to your Amazon Lookout for Vision console bucket. For information about the console bucket, see Step 4: Create the console bucket (p. 4). To store datasets, images, and Amazon SageMaker Ground Truth manifest files in a different Amazon S3 bucket, your users need additional permissions. For more information, see the section called “Setting Amazon S3 bucket permissions” (p. 3).

You can attach the `AmazonLookoutVisionConsoleFullAccess` policy to your IAM identities.

Permissions groupings

This policy is grouped into statements based on the set of permissions provided:

- `LookoutVisionFullAccess` – Allows access to perform all Lookout for Vision actions.
- `LookoutVisionConsoleS3BucketSearchAccess` – Allows listing of all Amazon S3 buckets owned by the caller. Lookout for Vision uses this action to identify the AWS Region-specific Lookout for Vision console bucket, if one exists in the caller's account.
- `LookoutVisionConsoleS3BucketFirstUseSetupAccessPermissions` – Allows creating and configuring Amazon S3 buckets that match the Lookout for Vision console bucket name pattern. Lookout for Vision uses these actions to create and configure a Region-specific Lookout for Vision console bucket when it can't find one.
- `LookoutVisionConsoleS3BucketAccess` – Allows dependent Amazon S3 actions on buckets that match the Lookout for Vision console bucket name pattern. Lookout for Vision uses `s3:ListBucket` to search for image objects when creating a dataset from an Amazon S3 bucket and when starting a trial detection task. Lookout for Vision uses `s3:GetBucketLocation` and `s3:GetBucketVersioning` to validate the bucket's AWS Region, owner, and configuration as part of the following:
  - Creating a dataset
  - Training a model
  - Starting a trial detection task
  - Performing trial detection feedback

  `LookoutVisionConsoleS3ObjectAccess` – Allows reading and writing of Amazon S3 objects inside buckets that match the Lookout for Vision Console bucket name pattern. Lookout for Vision uses these actions to display images in console gallery views and to upload new images for use in datasets. Additionally, these permissions allow Lookout for Vision to write out metadata while creating a dataset, training a model, starting a trial detection task, and performing trial detection feedback.
- `LookoutVisionConsoleDatasetLabelingToolsAccess` – Allows dependent Amazon SageMaker GroundTruth labeling actions. Lookout for Vision uses these actions to scan S3 buckets for images, create GroundTruth manifest files, and to annotate trial detection task results with validation labels.
- `LookoutVisionConsoleDashboardAccess` - Allows reading of Amazon CloudWatch metrics. Lookout for Vision uses these actions to populate the dashboard graphs and anomalies-detected statistics.

- `LookoutVisionConsoleTagSelectorAccess` – Allows reading account-specific tag key and tag value suggestions. Lookout for Vision uses these permissions to provide recommendations for tag keys and tag values within the **Manage tags** console pages.
- `LookoutVisionConsoleKmsKeySelectorAccess` – Allows listing AWS Key Management Service (KMS) keys and aliases. Amazon Lookout for Vision uses this permission to populate the KMS keys in the suggested **Tags** selection on certain Lookout for Vision actions that support customer managed KMS keys for encryption.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "LookoutVisionFullAccess",
            "Effect": "Allow",
            "Action": [
                "lookoutvision:*"
            ],
            "Resource": "*"
        },
        {
            "Sid": "LookoutVisionConsoleS3BucketSearchAccess",
            "Effect": "Allow",
            "Action": [
                "s3:ListAllMyBuckets"
            ],
            "Resource": "*"
        },
        {
            "Sid": "LookoutVisionConsoleS3BucketFirstUseSetupAccess",
            "Effect": "Allow",
            "Action": [
                "s3:CreateBucket",
                "s3:PutBucketVersioning",
                "s3:PutLifecycleConfiguration",
                "s3:PutEncryptionConfiguration",
                "s3:PutBucketPublicAccessBlock"
            ],
            "Resource": "arn:aws:s3:::lookoutvision-*"
        },
        {
            "Sid": "LookoutVisionConsoleS3BucketAccess",
            "Effect": "Allow",
            "Action": [
                "s3:ListBucket",
                "s3:GetBucketLocation",
                "s3:GetBucketAcl",
                "s3:GetBucketVersioning"
            ],
            "Resource": "arn:aws:s3:::lookoutvision-*"
        },
        {
            "Sid": "LookoutVisionConsoleS3ObjectAccess",
            "Effect": "Allow",
            "Action": [
                "s3:GetObject",
                "s3:GetObjectVersion",
                "s3:PutObject",
                "s3:AbortMultipartUpload",
                "s3:ListMultipartUploadParts"
            ],
            "Resource": "arn:aws:s3:::lookoutvision-*/*"
        },
        {
```

```
            "Sid": "LookoutVisionConsoleDatasetLabelingToolsAccess",
            "Effect": "Allow",
            "Action": [
                "groundtruthlabeling:RunGenerateManifestByCrawlingJob",
                "groundtruthlabeling:AssociatePatchToManifestJob",
                "groundtruthlabeling:DescribeConsoleJob"
            ],
            "Resource": "*"
        },
        {
            "Sid": "LookoutVisionConsoleDashboardAccess",
            "Effect": "Allow",
            "Action": [
                "cloudwatch:GetMetricData",
                "cloudwatch:GetMetricStatistics"
            ],
            "Resource": "*"
        },
        {
            "Sid": "LookoutVisionConsoleTagSelectorAccess",
            "Effect": "Allow",
            "Action": [
                "tag:GetTagKeys",
                "tag:GetTagValues"
            ],
            "Resource": "*"
        },
        {
            "Sid": "LookoutVisionConsoleKmsKeySelectorAccess",
            "Effect": "Allow",
            "Action": [
                "kms:ListAliases"
            ],
            "Resource": "*"
        }
    ]
}
```

AWS managed policy: AmazonLookoutVisionConsoleReadOnlyAccess

Use the `AmazonLookoutVisionConsoleReadOnlyAccess` policy to allow users read-only access to the Amazon Lookout for Vision console, actions (SDK operations), and any dependencies that the service has.

The `AmazonLookoutVisionConsoleReadOnlyAccess` policy includes Amazon S3 permissions for the Amazon Lookout for Vision console bucket. If your dataset images or Amazon SageMaker Ground Truth manifest files are in a different Amazon S3 bucket, your users need additional permissions. For more information, see the section called “Setting Amazon S3 bucket permissions” (p. 3).

You can attach the `AmazonLookoutVisionConsoleReadOnlyAccess` policy to your IAM identities.

Permissions groupings

This policy is grouped into statements based on the set of permissions provided:

- `LookoutVisionReadOnlyAccess` – Allows access to perform read-only Lookout for Vision actions.
- `LookoutVisionConsoleS3BucketSearchAccess` – Allows listing of all S3 buckets owned by the caller. Lookout for Vision uses this action to identify the AWS Region-specific Lookout for Vision console bucket, if there is one in the caller’s account.

- `LookoutVisionConsoleS3ObjectReadAccess` – Allows reading Amazon S3 objects and Amazon S3 object versions in Lookout for Vision console buckets. Lookout for Vision uses these actions to display the images in datasets, models, and trial detections.
- `LookoutVisionConsoleDashboardAccess` – Allows reading Amazon CloudWatch metrics. Lookout for Vision uses these actions to populate statistics for dashboard graphs and anomalies detected.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "LookoutVisionReadOnlyAccess",
            "Effect": "Allow",
            "Action": [
                "lookoutvision:DescribeDataset",
                "lookoutvision:DescribeModel",
                "lookoutvision:DescribeProject",
                "lookoutvision:DescribeTrialDetection",
                "lookoutvision:ListDatasetEntries",
                "lookoutvision:ListModels",
                "lookoutvision:ListProjects",
                "lookoutvision:ListTagsForResource",
                "lookoutvision:ListTrialDetections"
            ],
            "Resource": "*"
        },
        {
            "Sid": "LookoutVisionConsoleS3BucketSearchAccess",
            "Effect": "Allow",
            "Action": [
                "s3:ListAllMyBuckets"
            ],
            "Resource": "*"
        },
        {
            "Sid": "LookoutVisionConsoleS3ObjectReadAccess",
            "Effect": "Allow",
            "Action": [
                "s3:GetObject",
                "s3:GetObjectVersion"
            ],
            "Resource": "arn:aws:s3:::lookoutvision-*/*"
        },
        {
            "Sid": "LookoutVisionConsoleDashboardAccess",
            "Effect": "Allow",
            "Action": [
                "cloudwatch:GetMetricData",
                "cloudwatch:GetMetricStatistics"
            ],
            "Resource": "*"
        }
    ]
}
```

Lookout for Vision updates to AWS managed policies

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

Change	Description	Date
Lookout for Vision started tracking changes	Amazon Lookout for Vision started tracking changes for its AWS managed policies.	March 1st, 2021

Troubleshooting Amazon Lookout for Vision identity and access

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

Topics

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

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 project but does not have `lookoutvision:DescribeProject` permissions.

```
User: arn:aws:iam::123456789012:user/mateojackson is not authorized to perform:
 lookoutvision:DescribeProject on resource: my-project
```

In this case, Mateo asks his administrator to update his policies to allow him to access the `my-project` resource using the `lookoutvision:DescribeProject` action.

The following example error occurs when the `marymajor` IAM user tries to detect anomalies in an image but does not have `lookoutvision:DetectAnomalies` permissions.

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

In this case, Mary asks her administrator to update her policies to allow her to call the `lookoutvision:DetectAnomalies` action.

I want to view my access keys

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

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

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

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

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

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

To allow others to access Lookout for Vision, 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 Lookout for Vision.

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

I want to allow people outside of my AWS account to access my Lookout for Vision 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 Lookout for Vision supports these features, see How Amazon Lookout for Vision works with IAM (p. 109).
- To learn how to provide access to your resources across AWS accounts that you own, see Providing access to an IAM user in another AWS account that you own in the *IAM User Guide*.
- To learn how to provide access to your resources to third-party AWS accounts, see Providing access to AWS accounts owned by third parties in the *IAM User Guide*.
- To learn how to provide access through identity federation, see Providing access to externally authenticated users (identity federation) in the *IAM User Guide*.
- To learn the difference between using roles and resource-based policies for cross-account access, see How IAM roles differ from resource-based policies in the *IAM User Guide*.

Logging and monitoring in Amazon Lookout for Vision

Monitoring is an important part of maintaining the reliability, availability, and performance of Amazon Lookout for Vision and your AWS solutions. You should collect monitoring data from all of the parts of your AWS solution so that you can more easily debug a multi-point failure, if one occurs. AWS provides the following tools for monitoring your Lookout for Vision resources and builds and for responding to potential incidents.

Topics

Logging Lookout for Vision API calls with AWS CloudTrail

Amazon Lookout for Vision is integrated with AWS CloudTrail, a service that provides a record of actions taken by a user, role, or an AWS service in Lookout for Vision. CloudTrail captures all API calls for Lookout for Vision as events. The calls captured include calls from the Lookout for Vision console and code calls to the Lookout for Vision API operations. If you create a trail, you can enable continuous delivery of CloudTrail events to an Amazon S3 bucket, including events for Lookout for Vision. 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 Lookout for Vision, the IP address from which the request was made, who made the request, when it was made, and additional details.

To learn more about CloudTrail, see the AWS CloudTrail User Guide.

Lookout for Vision information in CloudTrail

CloudTrail is enabled on your AWS account when you create the account. When activity occurs in Lookout for Vision, 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 Lookout for Vision, create a trail. A *trail* enables CloudTrail to deliver log files to an Amazon S3 bucket. By default, when you create a trail in the console, the trail applies to all AWS Regions. The trail logs events from all Regions in the AWS partition and delivers the log files to the Amazon S3 bucket that you specify. Additionally, you can configure other AWS services to further analyze and act upon the event data collected in CloudTrail logs. For more information, see the following:

- 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

All Lookout for Vision actions are logged by CloudTrail and are documented in the Lookout for Vision API reference documentation. For example, calls to the `CreateProject`, `DetectAnomalies` and `StartModel` actions generate entries in the CloudTrail log files.

If you monitor Amazon Lookout for Vision API calls, you might see calls to the following APIs.

- lookoutvision:StartTrailDetection
- lookoutvision:ListTrailDetection
- lookoutvision:DescribeTrialDetection

These special calls are used by Amazon Lookout for Vision to support various operations related to trial detection. For more information, see Verifying your model with a trial detection task (p. 45).

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 Lookout for Vision log file entries

A trail is a configuration that enables delivery of events as log files to an Amazon S3 bucket that you specify. CloudTrail log files contain one or more log entries. An event represents a single request from any source and includes information about the requested action, the date and time of the action, request parameters, and so on. CloudTrail log files 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 `CreateDataset` action.


```
{
  "eventVersion": "1.08",
  "userIdentity": {
    "type": "AssumedRole",
    "principalId": "AROAYN4CJAYDEXAMPLE:user",
    "arn": "arn:aws:sts::123456789012:assumed-role/Admin/MyUser",
    "accountId": "123456789012",
    "accessKeyId": "ASIAYN4CJAYEXAMPLE",
    "sessionContext": {
      "sessionIssuer": {
        "type": "Role",
        "principalId": "AROAYN4CJAYDCGEXAMPLE",
        "arn": "arn:aws:iam::123456789012:role/Admin",
        "accountId": "123456789012",
        "userName": "Admin"
      },
      "webIdFederationData": {},
      "attributes": {
        "mfaAuthenticated": "false",
        "creationDate": "2020-11-20T13:15:09Z"
      }
    }
  },
  "eventTime": "2020-11-20T13:15:43Z",
  "eventSource": "lookoutvision.amazonaws.com",
  "eventName": "CreateDataset",
  "awsRegion": "us-east-1",
  "sourceIPAddress": "128.0.0.1",
  "userAgent": "aws-cli/3",
  "requestParameters": {
    "projectName": "P1",
    "datasetType": "train",
    "datasetSource": {
      "groundTruthManifest": {
        "s3Object": {
          "bucket": "myuser-bucketname",
          "key": "training.manifest"
        }
      }
    },
```

```
    "clientToken": "EXAMPLE-0526-47dd-a5d3-2ca975820a34"
  },
  "responseElements": {
    "status": "CREATE_IN_PROGRESS"
  },
  "requestID": "EXAMPLE-15e1-4bc9-be38-cda2537c75bf",
  "eventID": "EXAMPLE-c5e7-43e0-8449-8d9b87e15acb",
  "readOnly": false,
  "eventType": "AwsApiCall",
  "managementEvent": true,
  "eventCategory": "Management",
  "recipientAccountId": "123456789012"
}
```


Monitoring Lookout for Vision with Amazon CloudWatch

You can use CloudWatch to monitor image processing and anomaly detecting metrics in Lookout for Vision.

Lookout for Vision metrics

Amazon Lookout for Vision sends the following metrics to CloudWatch for the `DetectAnomalies` action:

Metric	Description
`DetectedAnomalyCount`	The number of anomalies detected in a project Valid Statistics: `Sum,Average` Unit: Count
`ProcessedImageCount`	The total number of images run through anomaly detection Valid Statistics: `Sum,Average` Unit: Count
`InvalidImageCount`	The number of images that were invalid and could not return a result Valid Statistics: `Sum,Average` Unit: Count
`SuccessfulRequestCount`	The number of successful API calls Valid Statistics: `Sum,Average` Unit: Count
`ErrorCount`	The number of API errors Valid Statistics: `Sum,Average` Unit: Count

Metric	Description
`ThrottledCount`	The number of API errors that were due to throttling Valid Statistics: `Sum,Average` Unit: Count
`Time`	The time in milliseconds for Lookout for Vision to compute the anomaly detection Valid Statistics: `Data Samples,Average` Units: Milliseconds for `Average` statistics and Count for `Data Samples` statistics

Dimensions of Lookout for Vision metrics

Lookout for Vision metrics use the Lookout for Vision namespace and provide metrics for the following dimensions:

Dimension	Description
`ProjectName`	You can split metrics by project to see which projects are having problems or need to be updated.
`ModelVersion`	You can split metrics by model version to see which models are having problems or need to be updated.

Compliance validation for Amazon Lookout for Vision

Third-party auditors assess the security and compliance of Amazon Lookout for Vision as part of multiple AWS compliance programs. Amazon Lookout for Vision is compliant with General Data Protection Regulation (GDPR).

To learn whether Lookout for Vision 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-compliant applications.

Note
Not all services are compliant with HIPAA.

- 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.

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

You can establish a private connection between your VPC and Amazon Lookout for Vision by creating an *interface VPC endpoint*. Interface endpoints are powered by AWS PrivateLink, a technology that enables you to privately access Lookout for Vision 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 Lookout for Vision APIs. Traffic between your VPC and Lookout for Vision does not leave the Amazon network.

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

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

Considerations for Lookout for Vision VPC endpoints

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

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

VPC endpoint policies are supported for Lookout for Vision. By default, full access to Lookout for Vision 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 Lookout for Vision

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

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

- com.amazonaws.*region*.lookoutvision

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

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

Creating a VPC endpoint policy for Lookout for Vision

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

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

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

Example: VPC endpoint policy for Lookout for Vision actions

The following is an example of an endpoint policy for Lookout for Vision. When attached to an endpoint, this policy specifies that all users who have access to the VPC interface endpoint are allowed to call the `DetectAnomalies` API operation for the Lookout for Vision model `myModel` associated with project `myProject`.

```
{
   "Statement":[
      {
         "Principal":"*",
         "Effect":"Allow",
         "Action":[
            "lookoutvision:DetectAnomalies"
         ],
         "Resource": "arn:aws:lookoutvision:us-west-2:123456789012:model/myProject/myModel"
      }
   ]
}
```

Resilience in Amazon Lookout for Vision

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

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

Infrastructure security in Amazon Lookout for Vision

As a managed service, Amazon Lookout for Vision 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 Lookout for Vision 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.

Quotas in Amazon Lookout for Vision

The following tables describe the current quotas within Amazon Lookout for Vision. For information about quotas that can be changed, see AWS service quotas.

Model quotas

The following quotas apply to the testing, training, and functionality of a model.

Resource	Quota
Supported file format	PNG and JPEG image formats
Minimum image dimension of image file in an Amazon S3 bucket	64 pixels x 64 pixels
Maximum image dimension of image file in an Amazon S3 bucket	4096 pixels X 4096 pixels is the maximum. Smaller dimensions are able to upload faster.
Differing image dimensions of image files used in a project	All images in the dataset must have the same dimensions
Maximum file size for an image in an Amazon S3 bucket	8 MB
Lack of labels	Images must be labeled as normal or anomaly before training. Images without labels are ignored during training.
Minimum number of images labeled normal in training dataset	10 for a project with separate training and test datasets. 20 for project with a single dataset.
Minimum number of images labeled anomaly in training dataset	0 for a project with separate training and test datasets. 10 for a project with a single dataset.
Maximum number of images in training dataset	16,000
Minimum number of images labeled normal in test dataset	10
Minimum number of images labeled anomaly in test dataset	10
Maximum number of images in test dataset	4,000
Maximum number of images in trial detection dataset	2,000
Maximum dataset manifest file size	1 GB
Maximum number of training datasets in a model	1

Resource	Quota
Maximum training time	24 hours
Maximum testing time	24 hours

Document History for Amazon Lookout for Vision

The following table describes important changes in each release of the *Amazon Lookout for Vision Developer Guide*. For notification about updates to this documentation, you can subscribe to an RSS feed.

- **Latest documentation update:** September 7th, 2021

update-history-change	update-history-description	update-history-date
New Python and Java 2 examples added (p. 130)	Added Python and Java 2 examples for analyzing images with `DetectAnomalies`. For more information, see Detecting anomalies in an image.	September 7, 2021
New AWS managed policies added. (p. 130)	Amazon Lookout for Vision adds support for AWS managed policies. For more information, see AWS managed policies for Amazon Lookout for Vision.	May 11, 2021
Updated inference unit information. (p. 130)	Added information describing inference units and how they are charged. For more information, see Running your trained Amazon Lookout for Vision model.	March 15, 2021
General availability for Amazon Lookout for Vision. (p. 130)	Amazon Lookout for Vision is now generally available. Python code examples updated to handle asynchronous tasks such as training a model.	February 17, 2021
Tagging and AWS CloudFormation support added. (p. 130)	You can now tag Amazon Lookout for Vision models and create projects with AWS CloudFormation. For more information, see Tagging models and Creating Amazon Lookout for Vision projects with AWS CloudFormation.	January 31, 2021
New feature and guide (p. 130)	This is the initial release of the Amazon Lookout for Vision service *Amazon Lookout for Vision Developer Guide*.	December 1, 2020

AWS glossary

For the latest AWS terminology, see the AWS glossary in the *AWS General Reference*.