aws

开发人员指南

Amazon Lookout for Vision

Amazon Lookout for Vision: 开发人员指南

Table of Contents

什么是 Amazon Lookout for Vision？ 1
主要优势 1
您是否是 Amazon Lookout for Vision 的新最终用户？ 1
设置 Amazon Lookout for Vision 2
第 1 步：创建 AWS 账户 2
注册获取 AWS 账户 2
创建具有管理访问权限的用户 3
步骤 2：设置权限 4
使用 AWS 托管策略设置控制台访问权限 4
设置 Amazon S3 桶权限 5
分配权限 6
步骤 3：创建控制台桶 6
使用 Amazon Lookout for Vision 控制台创建控制台桶 7
使用 Amazon S3 创建控制台桶 8
控制台桶设置 8
步骤 4：设置 AWS CLI 和 AWS 软件开发工具包 9
安装软件 AWS 开发工具包 9
授予编程式访问权限 9
设置 SDK 权限 12
调用 Amazon Lookout for Vision 操作 16
第 5 步：（可选）使用自己的 AWS KMS 密钥 20
了解 Amazon Lookout for Vision 22
选择模型类型 23
图像分类模型 23
图像分割模型 23
创建您的模型 24
创建项目 24
创建数据集 25
训练您的模型 26
评估您的模型 26
使用您的模型 27
在边缘设备上使用您的模型 27
使用您的控制面板 28
开始使用 29

步骤 1：创建清单文件并上传图像 31
步骤 2：创建模型 32
步骤 3：启动模型 39
步骤 4：分析图像 42
步骤 5：停止模型 46
后续步骤 48
创建您的模型 49
创建您的项目 49
创建项目（控制台） 50
创建项目（SDK） 50
创建您的数据集 52
为数据集准备图像 52
创建数据集 54
本地计算机 55
Amazon S3 存储桶 56
清单文件 59
标注图像 84
选择模型类型 84
图像分类（控制台） 85
分割图像（控制台） 86
训练您的模型 89
训练模型（控制台） 90
训练模型（SDK） 91
模型训练问题排查 97
异常标签颜色与掩码图像中异常的颜色不匹配 97
掩码图像不是 PNG 格式 99
分割或分类标签不准确或缺失 99
改进您的模型 101
步骤 1：评估模型的性能 101
图像分类指标 101
图像分割模型指标 101
精度 102
调用 102
F1 分数 103
平均交并比 (IoU) 103
测试结果 104

步骤 2：改进您的模型 104
查看性能指标 106
查看性能指标（控制台） 106
查看性能指标 (SDK) 107
验证您的模型 111
运行试用检测任务 111
验证试用检测结果 112
使用注释工具更正分割标签 113
运行您的模型 116
推理单位 116
使用推理单位管理吞吐量 117
可用区 118
启动您的模型 118
启动您的模型（控制台） 119
启动您的模型 (SDK) 120
停止您的模型 125
停止您的模型（控制台） 125
停止您的模型 (SDK) 126
检测图像中的异常 131
调用 DetectAnomalies 131
了解 DetectAnomalies 的响应 135
分类模型 135
分割模型 136
确定图像是否异常 137
分类 137
分段 139
显示分类和分割信息 144
使用 AWS Lambda 函数查找异常 159
步骤 1：创建 AWS Lambda 函数（控制台） 159
步骤 2：（可选）创建层（控制台） 161
步骤 3：添加 Python 代码（控制台） 162
步骤 4：试用您的 Lambda 函数 166
在边缘设备上使用您的模型 171
将模型部署到核心设备 172
核心设备要求 173
经过测试的设备、芯片架构和操作系统 173

核心设备内存和存储 175
必需的软件 175
设置您的核心设备 177
设置您的核心设备 177
将您的模型打包 178
包设置 178
将您的模型打包（控制台） 181
将您的模型打包 (SDK) 181
获取有关模型打包作业的信息 185
编写您的客户端应用程序组件 187
设置环境 187
使用模型 189
创建客户端应用程序组件 193
将您的组件部署到设备 198
部署组件所需的 IAM 权限 199
部署您的组件（控制台） 199
部署组件 (SDK) 201
Lookout for Vision Edge Agent API 参考 202
使用模型检测异常 202
获取模型信息 202
运行模型 203
DetectAnomalies 203
DescribeModel 208
ListModels 210
StartModel 211
StopModel 213
ModelStatus 214
使用 控制面板 215
管理您的资源 218
查看您的项目 218
查看您的项目（控制台） 219
查看您的项目 (SDK) 219
删除项目 222
删除项目（控制台） 222
删除项目 (SDK) 222
查看您的数据集 224

查看项目中的数据集（控制台） 225
查看项目中的数据集 (SDK) 225
向您的数据集中添加图像 228
添加更多图像 228
添加更多图像 (SDK) 229
从数据集中移除图像 234
从数据集中移除图像（控制台） 235
从数据集中移除图像 (SDK) 236
删除数据集 236
删除数据集（控制台） 225
删除数据集 (SDK) 237
从项目中导出数据集 (SDK) 239
查看您的模型 247
查看您的模型（控制台） 248
查看您的模型 (SDK) 248
删除模型 250
删除模型（控制台） 251
删除模型 (SDK) 251
标记模型 254
标记模型（控制台） 255
标记模型 (SDK) 256
查看您的试用检测任务 258
查看您的试用检测任务（控制台） 258
示例代码和数据集 259
示例代码 259
示例数据集 259
图像分割数据集 260
图像分类数据集 260
安全性 263
数据保护 263
数据加密 264
互联网络流量隐私保护 265
Identity and Access Management 265
受众 266
使用身份进行身份验证 266
使用策略管理访问 269

Amazon Lookout for Vision 如何与 IAM 协同工作 271
基于身份的策略示例 277
亚马逊云科技托管式策略 280
故障排除 290
合规性验证 291
韧性 292
基础设施安全性 293
监控 294
使用 CloudWatch 进行监控 294
CloudTrail 日志 297
CloudTrail 中的 Lookout for Vision 信息 297
了解 Lookout for Vision 日志文件条目 298
AWS CloudFormation 资源 300
Lookout for Vision 和 AWS CloudFormation 模板 300
了解有关 AWS CloudFormation 的更多信息 300
AWS PrivateLink 301
Lookout for Vision VPC 端点的注意事项 301
为 Lookout for Vision 创建接口 VPC 端点 301
为 Lookout for Vision 创建 VPC 端点策略 302
配额 303
模型配额 303
文档历史记录 305
AWS 术语表 309
..... cccx

什么是 Amazon Lookout for Vision？

您可以使用 Amazon Lookout for Vision 准确地大规模发现工业产品中的外观缺陷。它使用计算机视觉，可以发现工业产品中缺失的组件、车辆或结构的损坏、生产线中的违规行为，甚至硅芯片晶圆或任何注重质量的其他实物中的微小缺陷，如印刷电路板上缺少电容器。

主要优势

Amazon Lookout for Vision 具有以下优势：

- 快速高效地改进流程：您可以使用 Amazon Lookout for Vision，在工业过程中快速高效地大规模实施基于计算机视觉的检验。您只需提供 30 张基线好图像，Lookout for Vision 就可以自动构建用于缺陷检测的自定义 ML 模型。然后，您可以批量或实时处理来自网络摄像机的图像，以便快速准确地识别凹痕、裂缝和划痕等异常。
- 快速提高生产质量：借助 Amazon Lookout for Vision，您可以实时减少生产过程中的缺陷。它可以在控制面板中标识并报告视觉异常，因此您可以快速采取措施以阻止出现更多缺陷，从而提高生产质量并降低成本。
- 降低运营成本：Amazon Lookout for Vision 可以报告视觉检查数据中存在的趋势，如识别缺陷率最高的流程或标记近期缺陷变化情况。利用这些信息，您可以在发生代价高昂的计划外停机之前，确定是否安排进行生产线维护，或将生产转移到其他机器上进行。

您是否是 Amazon Lookout for Vision 的新最终用户？

如果您是 Amazon Lookout for Vision 的新用户，我们建议您按顺序阅读以下内容：

1. 设置 Amazon Lookout for Vision：在本节中，您将设置自己的账户详细信息。
2. 开始使用 Amazon Lookout for Vision：在本节中，您将了解如何创建自己的第一个 Amazon Lookout for Vision 模型。

设置 Amazon Lookout for Vision

在本节中，您将注册亚马逊云科技账户并设置 Amazon Lookout for Vision。

有关支持 Amazon Lookout for Vision 的 AWS 区域的信息，请参阅亚马逊 Lookout for Vision 终端节点和配额。

主题

- 第 1 步：创建 AWS 账户
- 步骤 2：设置权限
- 步骤 3：创建控制台桶
- 步骤 4：设置 AWS CLI 和 AWS 软件开发工具包
- 步骤 5：（可选）使用自己的 Amazon Key Management Service 密钥

第 1 步：创建 AWS 账户

在此步骤中，您将注册一个 AWS 帐户并创建一个管理用户。

主题

- 注册获取 AWS 账户
- 创建具有管理访问权限的用户

注册获取 AWS 账户

如果您没有 AWS 账户，请完成以下步骤来创建一个。

要注册 AWS 账户

1. 打开 https://portal.aws.amazon.com/billing/signup。
2. 按照屏幕上的说明进行操作。

   在注册时，将接到一通电话，要求使用电话键盘输入一个验证码。

   当您注册时 AWS 账户，就会创建AWS 账户根用户一个。根用户有权访问该账户中的所有 AWS 服务 和资源。作为安全最佳实践，请为用户分配管理访问权限，并且只使用根用户来执行需要根用户访问权限的任务。

AWS 注册过程完成后会向您发送一封确认电子邮件。在任何时候，您都可以通过转至 https://aws.amazon.com/ 并选择我的账户来查看当前的账户活动并管理您的账户。

创建具有管理访问权限的用户

注册后，请保护您的安全 AWS 账户 AWS 账户根用户 AWS IAM Identity Center，启用并创建管理用户，这样您就不会使用 root 用户执行日常任务。

保护你的 AWS 账户根用户

1. 选择 Root 用户并输入您的 AWS 账户 电子邮件地址，以账户所有者的身份登录。AWS Management Console在下一页上，输入您的密码。

   要获取使用根用户登录方面的帮助，请参阅《AWS 登录 用户指南》中的以根用户身份登录。
2. 为您的根用户启用多重身份验证 (MFA)。

   有关说明，请参阅 I A M 用户指南中的为 AWS 账户 根用户启用虚拟 MFA 设备（控制台）。

创建具有管理访问权限的用户

1. 启用 IAM Identity Center

   有关说明，请参阅《AWS IAM Identity Center 用户指南》中的启用 AWS IAM Identity Center。
2. 在 IAM Identity Center 中，为用户授予管理访问权限。

   有关使用 IAM Identity Center 目录 作为身份源的教程，请参阅《用户指南》 IAM Identity Center 目录中的使用默认设置配置AWS IAM Identity Center 用户访问权限。

以具有管理访问权限的用户身份登录

- 要使用您的 IAM Identity Center 用户身份登录，请使用您在创建 IAM Identity Center 用户时发送到您的电子邮件地址的登录网址。

  有关使用 IAM Identity Center 用户登录的帮助，请参阅AWS 登录 用户指南中的登录 AWS 访问门户。

将访问权限分配给其他用户

1. 在 IAM Identity Center 中，创建一个权限集，该权限集遵循应用最低权限的最佳做法。

有关说明，请参阅《AWS IAM Identity Center 用户指南》中的创建权限集。

2. 将用户分配到一个组，然后为该组分配单点登录访问权限。

   有关说明，请参阅《AWS IAM Identity Center 用户指南》中的添加组。

步骤 2：设置权限

要使用 Amazon Lookout for Vision，您需要获得 Lookout for Vision 控制台 AWS 、软件开发工具包操作和用于模型训练的 Amazon S3 存储桶的访问权限。

> **Note**
>
> 如果您仅使用 AWS SDK 操作，则可以使用限 AWS 于 SDK 操作的策略。有关更多信息，请参阅 设置 SDK 权限。

主题

- 使用 AWS 托管策略设置控制台访问权限
- 设置 Amazon S3 桶权限
- 分配权限

使用 AWS 托管策略设置控制台访问权限

使用以下 AWS 托管策略为 Amazon Lookout for Vision 控制台和 SDK 操作应用适当的访问权限。

- AmazonLookoutVisionConsoleFullAccess— 允许完全访问亚马逊 Lookout for Vision 控制台和 SDK 操作。您需要 `AmazonLookoutVisionConsoleFullAccess` 权限才能创建控制台桶。有关更多信息，请参阅 步骤 3：创建控制台桶。
- AmazonLookoutVisionConsoleReadOnlyAccess— 允许对 Amazon Lookout for Vision 控制台和 SDK 操作进行只读访问。

若要分配权限，请参阅 分配权限。

有关 AWS 托管策略的信息，请参阅 AWS 托管策略。

设置 Amazon S3 桶权限

Amazon Lookout for Vision 使用 Amazon S3 桶来存储以下文件：

- 数据集图像：用于训练模型的图像。有关更多信息，请参阅 创建您的数据集。
- Amazon SageMaker Ground Truth 格式化清单文件。例如， SageMaker GroundTruth 任务输出的清单文件。有关更多信息，请参阅 使用 Amazon G SageMaker round Truth 清单文件创建数据集。
- 模型训练的输出。

如果您使用控制台，Lookout for Vision 会创建一个 Amazon S3 桶（控制台桶）来管理您的项目。`LookoutVisionConsoleReadOnlyAccess` 和 `LookoutVisionConsoleFullAccess` 托管式策略包括了对控制台桶的 Amazon S3 访问权限。

您可以使用控制台存储桶来存储数据集图像和 G SageMaker round Truth 格式的清单文件。或者，您可以使用其他 Amazon S3 桶。存储桶必须归您的 AWS 账户所有，并且必须位于您使用 Lookout for Vision 的 AWS 区域。

要使用其他桶，请将以下策略添加到所需的用户或组。使用所需的桶名称替换 `my-bucket`。有关添加 IAM 策略的信息，请参阅创建 IAM 策略。

```
{
    "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/*"
            ]
        }
    ]
}
```

若要分配权限，请参阅 分配权限。

分配权限

要提供访问权限，请为您的用户、组或角色添加权限：

- 中的用户和群组 AWS IAM Identity Center：

  创建权限集合。按照《AWS IAM Identity Center 用户指南》中创建权限集的说明进行操作。
- 通过身份提供商在 IAM 中托管的用户：

  创建适用于身份联合验证的角色。按照《IAM 用户指南》中为第三方身份提供商创建角色（联合身份验证）的说明进行操作。
- IAM 用户：
  - 创建您的用户可以担任的角色。按照《IAM 用户指南》中为 IAM 用户创建角色的说明进行操作。
  - （不推荐使用）将策略直接附加到用户或将用户添加到用户组。按照《IAM 用户指南》中向用户添加权限（控制台）中的说明进行操作。

步骤 3：创建控制台桶

要使用 Amazon Lookout for Vision 控制台，您需要一个名为控制台桶的 Amazon S3 桶。控制台桶用于存储以下内容：

- 使用控制台上传到数据集的图像。
- 使用控制台开始的模型训练的训练结果。
- 试用检测结果。
- 当您使用控制台通过自动标注 S3 桶中的图像来创建数据集时，控制台创建的临时清单文件。控制台不会删除清单文件。

当你首次 AWS 在新区域打开亚马逊 Lookout for Vision 控制台时，Lookout for Vision 会代表你创建控制台存储桶。请注意控制台存储桶名称，因为您可能需要在 AWS SDK 操作或控制台任务（例如创建数据集）中使用存储桶名称。

或者，您可以使用 Amazon S3 来创建控制台桶。如果 Amazon S3 桶策略不允许 Amazon Lookout for Vision 控制台成功创建控制台桶，请使用此方法。例如，策略禁止自动创建 Amazon S3 桶。

> **Note**
>
> 如果您只使用 AWS 软件开发工具包而不使用 Lookout for Vision 控制台，则无需创建控制台存储桶。您可以使用其他 S3 桶，名称由自己选择。

控制台桶名称的格式为 `lookoutvision-<region>-<random value>`。随机值可确保桶名称之间不会发生冲突。

主题

- 使用 Amazon Lookout for Vision 控制台创建控制台桶
- 使用 Amazon S3 创建控制台桶
- 控制台桶设置

使用 Amazon Lookout for Vision 控制台创建控制台桶

使用以下步骤使用 Amazon Lookout for Vision 控制台为某个 AWS 地区创建控制台存储桶。有关我们启用的 S3 桶设置的信息，请参阅 控制台桶设置。

使用 Amazon Lookout for Vision 控制台创建控制台桶

1. 确保您正在使用的用户或组具有 `AmazonLookoutVisionConsoleFullAccess` 权限。有关更多信息，请参阅 步骤 2：设置权限。
2. 打开 Amazon Lookout for Vision 控制台，网址为 https://console.aws.amazon.com/lookoutvision/。
3. 在导航栏上，选中选择区域。然后选择要为其创建控制台存储桶的 AWS 区域。
4. 选择开始。
5. 如果这是您首次在当前亚马逊云科技区域打开控制台，请在首次设置对话框中执行以下操作：

   a. 记下所显示的 Amazon S3 存储桶的名称。稍后您将需要此信息。

b. 选择创建 S3 桶，由 Amazon Lookout for Vision 代表您创建控制台桶。

如果当前 AWS 区域的控制台存储桶已经存在，则不会显示 “首次设置” 对话框。

6. 关闭浏览器窗口。

使用 Amazon S3 创建控制台桶

您可以使用 Amazon S3 创建控制台桶。您必须在启用 Amazon S3 版本控制的情况下创建桶。我们建议使用 Amazon S3 生命周期配置，移除对象的非当前（先前）版本并删除未完成的分段上传。我们不建议使用会删除对象当前版本的生命周期配置。对于您使用 Amazon Lookout for Vision 控制台创建的控制台桶，有关我们为之启用的 S3 桶设置的信息，请参阅 控制台桶设置。

1. 确定要在其中创建控制台存储桶的 AWS 区域。有关受支持区域的信息，请参阅 Amazon Lookout for Vision 端点和配额。
2. 使用创建桶部分的 S3 控制台说明，创建一个桶。执行以下操作：

   a. 在步骤 3 中，指定前缀为 `lookoutvision-`*`region`*`-`*`your-identifier`* 的桶名称。将 `region` 更改为上一步中选择的区域代码。将 `your-identifier` 更改为您选择的唯一标识符。例如，`lookoutvision-us-east-1-my-console-bucket-1`

   b. 在第 4 步中，选择要使用的 AWS 区域。
3. 按照在桶上启用版本控制部分的 S3 控制台说明，启用桶的版本控制。
4. （可选）按照在桶上设置生命周期配置部分的 S3 控制台说明，指定桶的生命周期配置。执行以下操作以移除对象的非当前（先前）版本，并删除未完成的分段上传。您不需要执行步骤 6、8、9、10。

   a. 在步骤 5 中，选择应用到存储桶中的所有对象。

   b. 在步骤 7 中，选择永久删除对象的非当前版本和删除过期的对象删除标记或未完成的分段上传。

   c. 在步骤 11 中，输入删除对象的非当前版本之前要等待的天数。

   d. 在步骤 12 中，输入删除未完成的分段上传之前要等待的天数。

控制台桶设置

如果您使用 Amazon Lookout for Vision 控制台创建控制台桶，我们将在控制台桶上启用以下设置。

- 控制台桶中对象的版本控制。
- 控制台桶中对象的服务器端加密。
- 用于删除非当前对象（30 天）和未完成分段上传（3 天）的生命周期配置。
- 阻止公有访问控制台桶。

步骤 4：设置 AWS CLI 和 AWS 软件开发工具包

以下步骤向您展示如何安装 AWS Command Line Interface (AWS CLI) 和 AWS 软件开发工具包。本文档中的示例使用 AWS CLI、Python 和 Java AWS 软件开发工具包。

主题

- 安装软件 AWS 开发工具包
- 授予编程式访问权限
- 设置 SDK 权限
- 调用 Amazon Lookout for Vision 操作

安装软件 AWS 开发工具包

按照步骤下载和配置 AWS 软件开发工具包。

要设置 AWS CLI 和 S AWS DK

- 下载并安装您要使用的AWS CLI和 AWS 软件开发工具包。本指南提供了 AWS CLI、Java 和 Python 的示例。有关安装 AWS 软件开发工具包的信息，请参阅适用于 Amazon Web Services 的工具。

授予编程式访问权限

您可以在本地计算机或其他 AWS 环境（例如 Amazon Elastic Compute Cloud 实例）上运行本指南中的 AWS CLI 和代码示例。要运行这些示例，您需要授予对示例使用的 AWS SDK 操作的访问权限。

主题

- 在本地计算机上运行代码
- 在 AWS 环境中运行代码

在本地计算机上运行代码

要在本地计算机上运行代码，我们建议您使用短期凭证向用户授予对 AWS SDK 操作的访问权限。有关在本地计算机上运行 AWS CLI 和代码示例的具体信息，请参阅在本地计算机上使用配置文件。

如果用户想在 AWS 外部进行交互，则需要编程访问权限 AWS Management Console。授予编程访问权限的方式取决于正在访问的用户类型 AWS。

要向用户授予编程式访问权限，请选择以下选项之一。

哪个用户需要编程式访问权限？	目的	方式
人力身份 （在 IAM Identity Center 中管理的用户）	使用临时证书签署向 AWS CLI、 AWS 软件开发工具包或 AWS API 发出的编程请求。	按照您希望使用的界面的说明进行操作。 • 有关的 AWS CLI，请参阅《AWS Command Line Interface 用户指南》 AWS IAM Identity Center中的配置 AWS CLI 以使用。 • 有关 AWS 软件开发工具包、工具和 AWS API，请参阅《软件开发工具包和AWS 工具参考指南》中的 IAM 身份中心身份验证。
IAM	使用临时证书签署向 AWS CLI、 AWS 软件开发工具包或 AWS API 发出的编程请求。	按照 IAM 用户指南中的将临时证书与 AWS 资源配合使用中的说明进行操作。
IAM	（不推荐使用） 使用长期凭证签署向 AWS CLI、 AWS 软件开发工具包或 AWS API 发出的编程请求。	按照您希望使用的界面的说明进行操作。 • 有关信息 AWS CLI，请参阅用户指南中的使用 IAM 用户证书进行身份验证。AWS Command Line Interface

哪个用户需要编程式访问权限？	目的	方式
		• 有关 AWS SDK 和工具，请参阅 S AWS DK 和工具参考指南中的使用长期凭证进行身份验证。 • 有关 AWS API，请参阅 IAM 用户指南中的管理 IAM 用户的访问密钥。

在本地计算机上使用配置文件

您可以使用您在中创建 AWS CLI 的短期证书运行本指南中的和代码示例在本地计算机上运行代码。为了获取凭证和其他设置信息，这些示例使用名为 `lookoutvision-access` 的配置文件。例如：

```
session = boto3.Session(profile_name='lookoutvision-access')
lookoutvision_client = session.client("lookoutvision")
```

个人资料所代表的用户必须有权调用 Lookout for Vision SDK 操作以及示例所需的 AWS 其他 SDK 操作。有关更多信息，请参阅 设置 SDK 权限。若要分配权限，请参阅 分配权限。

要创建适用于 AWS CLI 和代码示例的配置文件，请选择以下选项之一。确保您创建的配置文件的名称为 `lookoutvision-access`。

- 由 IAM 管理的用户 — 按照切换到 IAM 角色 (AWS CLI) 部分的说明进行操作。
- 员工身份（由管理的用户 AWS IAM Identity Center）— 按照配置 AWS CLI 中的说明进行操作 AWS IAM Identity Center。对于这些代码示例，我们建议使用集成式开发环境 (IDE)，该环境支持 AWS Toolkit，可通过 IAM Identity Center 实现身份验证。有关 Java 示例，请参阅使用 Java 开始构建。有关 Python 示例，请参阅使用 Python 开始构建。有关更多信息，请参阅 IAM Identity Center 凭证。

> **Note**
>
> 您可以使用代码获取短期凭证。有关更多信息，请参阅切换到 IAM 角色 (AWS API)。对于 IAM Identity Center，请按照获取用于 CLI 访问的 IAM 角色凭证部分的说明操作，获取角色的短期凭证。

在 AWS 环境中运行代码

您不应使用用户凭据在 AWS 环境中签署 AWS SDK 调用，例如在 AWS Lambda 函数中运行的生产代码。相反，您应该配置一个角色来定义代码所需的权限。然后，将该角色附加到运行代码的环境。关于如何附加角色和提供可用的临时凭证，取决于运行代码的环境：

- AWS Lambda 函数 — 使用 Lambda 在担任 Lambda 函数的执行角色时自动提供给您的函数的临时证书。这些凭证在 Lambda 环境变量中可用。您不需要指定配置文件。有关更多信息，请参阅 Lambda 执行角色。
- Amazon EC2 — 使用 Amazon EC2 实例元数据端点凭证提供程序。该提供程序会使用您附加到 Amazon EC2 实例的 Amazon EC2 实例配置文件，自动为您生成和刷新凭证。有关更多信息，请参阅使用 IAM 角色为 Amazon EC2 实例上运行的应用程序授予权限。
- Amazon Elastic Container Service — 使用 Container 凭证提供程序。Amazon ECS 会向元数据端点发送和刷新凭证。您指定的任务 IAM 角色会提供一项策略，用于管理您的应用程序所使用的凭证。有关更多信息，请参阅与亚马逊云科技服务交互。
- Greengrass 核心设备：使用 X.509 证书通过 TLS 双向身份验证协议连接到 AWS IoT Core。这些证书可以让设备在没有亚马逊云科技凭证的情况下与 Amazon IoT 进行交互。Amazon IoT 凭证提供程序使用 X.509 证书对设备进行身份验证，并且以有限权限临时安全令牌的形式颁发亚马逊云科技凭证。有关更多信息，请参阅与亚马逊云科技服务交互。

有关凭证提供程序的更多信息，请参阅标准化凭证提供程序。

设置 SDK 权限

要使用 Amazon Lookout for Vision SDK 操作，您需要获得 Lookout for Vision API 和用于模型训练的 Amazon S3 桶的访问权限。

主题

- 授予 SDK 操作权限
- 授予 Amazon S3 桶权限

- 分配权限

授予 SDK 操作权限

建议仅授予执行任务所必需的权限（最低权限）。例如，要拨打电话 DetectAnomalies，您需要获得执行权限`lookoutvision:DetectAnomalies`。要查找操作的权限，请查看 API 参考。

刚开始使用应用程序时，您可能不知道具体需要哪些权限，因此可以从广泛权限入手。 AWS 托管式策略可以提供一些权限来帮助您入门。

- AmazonLookoutVisionFullAccess— 允许完全访问亚马逊 Lookout for Vision 软件开发工具包的操作。
- AmazonLookoutVisionReadOnlyAccess— 允许访问只读 SDK 操作。

控制台的托管式策略还提供 SDK 操作的访问权限。有关更多信息，请参阅 步骤 2：设置权限。

有关 AWS 托管策略的信息，请参阅 AWS 托管策略。

如果知道应用程序所需的权限，则可定义特定于您的使用场景的客户管理型策略，从而进一步减少权限。有关更多信息，请参阅客户管理型策略。

> **Note**
>
> 入门说明需要 `s3:PutObject` 权限。有关更多信息，请参阅 步骤 1：创建清单文件并上传图像。

若要分配权限，请参阅 分配权限。

授予 Amazon S3 桶权限

要训练模型，您需要具有适当权限的 Amazon S3 桶来存储图像、清单文件和训练输出。桶必须归您的亚马逊云科技账户所有，并且必须位于您使用 Amazon Lookout for Vision 所在的亚马逊云科技区域。

仅限 SDK 的托管式策略（`AmazonLookoutVisionFullAccess` 和 `AmazonLookoutVisionReadOnlyAccess`）不包括 Amazon S3 桶权限，您需要应用以下权限策略才能访问所使用的桶，包括现有控制台桶。

控制台托管式策略（`AmazonLookoutVisionConsoleFullAccess` 和 `AmazonLookoutVisionConsoleReadOnlyAccess`）包括对控制台桶的访问权限。如果通过 SDK

操作访问控制台桶，并且拥有控制台托管式策略权限，则无需使用以下策略。有关更多信息，请参阅步骤 2：设置权限。

确定任务权限

使用以下信息确定要执行的任务需要哪些权限。

创建数据集

要使用创建数据集 CreateDataset，您需要以下权限。

- `s3:GetBucketLocation`：允许 Lookout for Vision 验证您的桶是否与您使用 Lookout for Vision 所在的区域相同。
- `s3:GetObject`：允许访问 `DatasetSource` 输入参数中指定的清单文件。如果要指定清单文件的确切 S3 对象版本，则还需要清单文件上的 `s3:GetObjectVersion`。有关更多信息，请参阅在 S3 桶中使用版本控制。

创建模型

要使用创建模型 CreateModel，您需要以下权限。

- `s3:GetBucketLocation`：允许 Lookout for Vision 验证您的桶是否与您使用 Lookout for Vision 所在的区域相同。
- `s3:GetObject`：允许访问项目训练及测试数据集中指定的图像。
- `s3:PutObject`：给予在指定的桶中存储训练输出的权限。您需要在 `OutputConfig` 参数中指定输出桶的位置。或者，您可以缩小权限范围，使之仅限 `S3Location` 的 `Prefix` 字段中指定的对象键。有关更多信息，请参阅OutputConfig。

访问图像、清单文件和训练输出

无需 Amazon S3 桶权限即可查看 Amazon Lookout for Vision 操作响应。如果要访问操作响应中引用的图像、清单文件和训练输出，则需要 `s3:GetObject` 权限。如果要访问受版本控制的 Amazon S3 对象，则需要 `s3:GetObjectVersion` 权限。

设置 Amazon S3 桶策略

您可以使用以下策略来指定所需的 Amazon S3 桶权限，以便创建数据集（`CreateDataset`）、创建模型（`CreateModel`）以及访问图像、清单文件和训练输出。将 *`my-bucket`* 的值更改为您要使用的桶的名称。

您可以根据自身需求调整策略。有关更多信息，请参阅 确定任务权限。将策略添加到所需的用户。有关更多信息，请参阅创建 IAM 策略。

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

若要分配权限，请参阅 分配权限。

分配权限

要提供访问权限，请为您的用户、组或角色添加权限：

- 中的用户和群组 AWS IAM Identity Center：

  创建权限集合。按照《AWS IAM Identity Center 用户指南》中创建权限集的说明进行操作。
- 通过身份提供商在 IAM 中托管的用户：

  创建适用于身份联合验证的角色。按照《IAM 用户指南》中为第三方身份提供商创建角色（联合身份验证）的说明进行操作。
- IAM 用户：
  - 创建您的用户可以担任的角色。按照《IAM 用户指南》中为 IAM 用户创建角色的说明进行操作。
  - （不推荐使用）将策略直接附加到用户或将用户添加到用户组。按照IAM 用户指南 中向用户添加权限（控制台）中的说明进行操作。

调用 Amazon Lookout for Vision 操作

运行以下代码，确认您可以调用 Amazon Lookout for Vision API。该代码列出了您 AWS 账户中当前 AWS 区域中的项目。如果您之前未创建项目，则响应为空，但确实会确认您可以调用 `ListProjects` 操作。

通常，调用示例函数需要 Amazon SDK Lookout for Vision 客户端和任何其他所需的参数。Amazon SDK Lookout for Vision 客户端会在主函数中声明。

如果该代码失败，请检查所用的用户是否具有正确的权限。另请检查您用作 Amazon Lookout for Vision 的 AWS 地区并非在 AWS 所有地区都可用。

调用 Amazon Lookout for Vision 操作

1. 如果您尚未这样做，请安装和配置和 AWS SDK。 AWS CLI 有关更多信息，请参阅 步骤 4：设置 AWS CLI 和 AWS 软件开发工具包。
2. 使用以下示例代码查看您的项目。

   CLI

   使用 `list-projects` 命令列出您账户中的项目。

```
aws lookoutvision list-projects \
```

```
--profile lookoutvision-access
```

Python

此代码取自 AWS 文档 SDK 示例 GitHub 存储库。请在此处查看完整示例。

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


from botocore.exceptions import ClientError
import boto3

class GettingStarted:

    @staticmethod
    def list_projects(lookoutvision_client):
        """
        Lists information about the projects that are in in your AWS account
        and in the current AWS Region.

        :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("ARN: " + project["ProjectArn"])
                print()
            print("Done!")
        except ClientError:
            raise
def main():
    session = boto3.Session(profile_name='lookoutvision-access')
    lookoutvision_client = session.client("lookoutvision")

    GettingStarted.list_projects(lookoutvision_client)

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

Java V2

此代码取自 AWS 文档 SDK 示例 GitHub 存储库。请在此处查看完整示例。

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

package com.example.lookoutvision;

import software.amazon.awssdk.auth.credentials.ProfileCredentialsProvider;
import software.amazon.awssdk.services.lookoutvision.LookoutVisionClient;
import software.amazon.awssdk.services.lookoutvision.model.ProjectMetadata;
import
 software.amazon.awssdk.services.lookoutvision.paginators.ListProjectsIterable;
import software.amazon.awssdk.services.lookoutvision.model.ListProjectsRequest;
import
 software.amazon.awssdk.services.lookoutvision.model.LookoutVisionException;

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

public class GettingStarted {

    public static final Logger logger =
 Logger.getLogger(GettingStarted.class.getName());

    /**
     * Lists the Amazon Lookoutfor Vision projects in the current AWS account
 and
     * AWS Region.
     *
     * @param lfvClient   An Amazon Lookout for Vision client.
     * @return List<ProjectMetadata> Metadata for each project.
     */
    public static List<ProjectMetadata> listProjects(LookoutVisionClient
 lfvClient)
            throws LookoutVisionException {

        logger.log(Level.INFO, "Getting projects:");
```

```
        ListProjectsRequest listProjectsRequest = ListProjectsRequest.builder()
                .maxResults(100)
                .build();

        List<ProjectMetadata> projectMetadata = new ArrayList<>();

        ListProjectsIterable projects =
lfvClient.listProjectsPaginator(listProjectsRequest);

        projects.stream().flatMap(r -> r.projects().stream())
                .forEach(project -> {
                    projectMetadata.add(project);
                    logger.log(Level.INFO, project.projectName());
                });

        logger.log(Level.INFO, "Finished getting projects.");

        return projectMetadata;

    }

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

        try {

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

.credentialsProvider(ProfileCredentialsProvider.create("lookoutvision-access"))
                    .build();

            List<ProjectMetadata> projects = Projects.listProjects(lfvClient);

            System.out.printf("Projects%n--------%n");

            for (ProjectMetadata project : projects) {
                System.out.printf("Name: %s%n", project.projectName());
                System.out.printf("ARN: %s%n", project.projectArn());
                System.out.printf("Date: %s%n%n",
project.creationTimestamp().toString());
            }

        } catch (LookoutVisionException lfvError) {
            logger.log(Level.SEVERE, "Could not list projects: {0}: {1}",
```

```
                    new Object[] { lfvError.awsErrorDetails().errorCode(),
                            lfvError.awsErrorDetails().errorMessage() });
            System.out.println(String.format("Could not list projects: %s",
 lfvError.getMessage()));
            System.exit(1);
        }

    }

}
```

步骤 5：（可选）使用自己的 Amazon Key Management Service 密钥

您可以使用 Amazon Key Management Service（KMS），对存储在 Amazon S3 桶中的输入图像管理它们的加密。

默认情况下，您的图像使用亚马逊云科技拥有和管理的密钥进行加密。您也可以选择使用自己的 Amazon Key Management Service（KMS）密钥。有关更多信息，请参阅 Amazon Key Management Service 概念。

如果要使用自己的 KMS 密钥，请使用以下策略指定 KMS 密钥。将 *kms_key_arn* 更改为要使用的 KMS 密钥的 ARN（或 KMS 别名 ARN）。或者，指定 * 以使用任何 KMS 密钥。有关向用户或角色添加策略的信息，请参阅创建 IAM 策略。

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

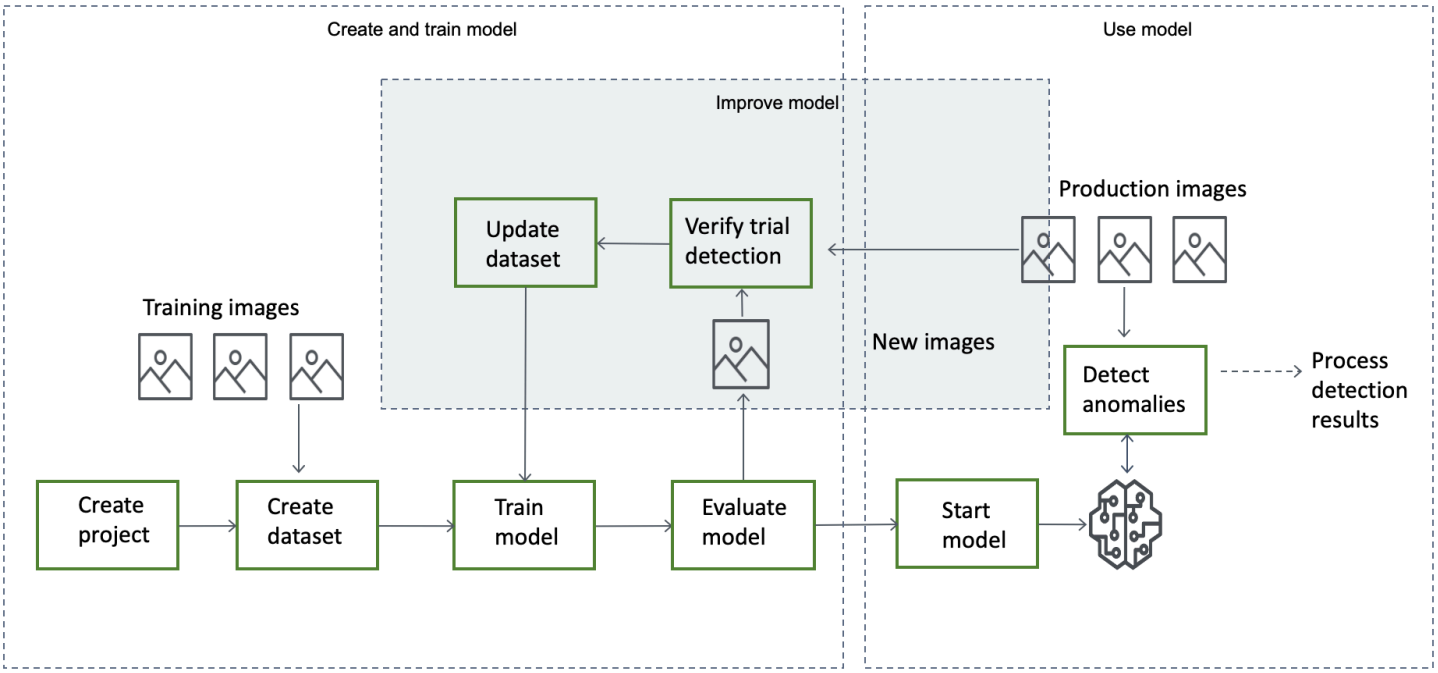
了解 Amazon Lookout for Vision

您可以使用 Amazon Lookout for Vision 准确地大规模发现工业产品中的外观缺陷，适用于以下任务：

- 检测损坏的零件：在制造和组装过程中发现产品表面质量、颜色和形状的损坏。
- 识别缺失的组件：根据对象的缺失、存在或位置来确定缺失的组件。例如，印刷电路板上缺少电容器。
- 发现工艺问题：检测具有重复图案的缺陷，如硅芯片晶圆上同一位置的重复划痕。

使用 Lookout for Vision，您可以创建一个计算机视觉模型，用于预测图像中是否存在异常。您需要提供图像，供 Amazon Lookout for Vision 用来训练和测试您的模型。Amazon Lookout for Vision 提供一些指标，您可用来评估并改进所训练的模型。您可以在 AWS 云中托管所训练的模型，也可以将模型部署到边缘设备。简单的 API 操作即可返回您的模型所做的预测。

创建、评估和使用模型的一般工作流如下：



主题

- 选择模型类型
- 创建您的模型
- 评估您的模型
- 使用您的模型

- 在边缘设备上使用您的模型
- 使用您的控制面板

选择模型类型

在创建模型之前，必须先确定您想要哪种类型的模型。您可以创建两种类型的模型，即图像分类 和图像分割。您需要根据自己的使用场景，确定要创建哪种类型的模型。

图像分类模型

如果只需要知道图像是否包含异常，但不需要知道异常的位置，应创建图像分类模型。图像分类模型可以预测图像是否包含异常。预测包括模型对预测准确性的置信度。模型不提供图像上所发现异常的任何位置信息。

图像分割模型

如果您需要知道异常的位置，如划痕的位置，应创建图像分割模型。Amazon Lookout for Vision 模型使用语义分割，确定各类异常（如划痕或缺失零件）位于图像上的哪些像素。

> **Note**
>
> 语义分割模型可以确定不同类型异常的位置。它不提供各个异常的实例信息。例如，如果图像包含两个凹痕，则 Lookout for Vision 会在代表凹痕异常类型的单个实体中，返回与两个凹痕相关的信息。

Amazon Lookout for Vision 分割模型会做出以下预测：

分类

模型会返回对所分析图像的分类（正常/异常），其中包括模型对预测的置信度。分类信息与分割信息分开进行计算，您不应假设它们之间存在关系。

分段

模型会返回一个图像掩码，用于标记图像上出现异常的像素。根据在数据集中分配给异常标签 的颜色，不同类型的异常会按颜色进行编码。异常标签表示异常的类型。例如，下图中的蓝色掩码用于标记在汽车上发现的划痕 异常类型的位置。

模型会返回掩码中每个异常标签的颜色代码。模型还会返回异常标签的图像覆盖百分比。

借助 Lookout for Vision 分割模型，您可以使用各种标准来分析模型的分析结果。例如：

- 异常位置：如果需要知道异常的位置，请使用分割信息来查看覆盖异常的掩码。
- 异常类型：使用分割信息来确定图像包含的异常类型是否超过可接受的数量。
- 覆盖区域：使用分割信息来确定某类异常覆盖的图像区域是否超过可接受的值。
- 图像分类：如果不需要知道异常的位置，请使用分类信息来确定图像是否包含异常。

有关示例代码，请参阅 检测图像中的异常。

在决定要使用哪种模型之后，您可以创建项目和数据集来管理您的模型。使用标签，您可以将图像归类为正常图像或异常图像。标签还可以标识分割信息，如掩码和异常类型。您对数据集中图像的标注决定了 Lookout for Vision 会创建什么类型的模型。

标注图像分割模型要比标注图像分类模型复杂。要训练分割模型，必须将训练图像归类为正常图像或异常图像。您还必须为每张异常图像定义异常掩码和异常类型。分类模型仅要求将训练图像标识为正常图像或异常图像。

创建您的模型

如下所述，创建模型的步骤为创建项目、创建数据集以及训练模型：

创建项目

创建一个项目来管理您创建的数据集和模型。一个项目应该用于一种使用场景，如检测单一类型机器零件中的异常。

您可以使用控制面板显示项目的概览。有关更多信息，请参阅 使用 Amazon Lookout for Vision 控制面板。

更多信息：创建您的项目。

创建数据集

要训练模型，Amazon Lookout for Vision 需要与您的使用场景对应的正常对象及异常对象的图像。您可以在数据集 中提供这些图像。

数据集是由图像和描述这些图像的标签组成的集合。图像应代表可能出现异常的单一类型的对象。有关更多信息，请参阅 为数据集准备图像。

借助 Amazon Lookout for Vision，一个项目可以使用单个数据集，或者，一个项目也可以具有不同的训练数据集和测试数据集。除非要更好地控制训练、测试和性能调优，否则我们建议使用单一数据集项目。

您可以通过导入图像来创建数据集。根据您如何导入图像，可能还会对图像进行标注。如果未标注，您应使用控制台来标注图像。

导入图像

如果使用 Lookout for Vision 控制台来创建数据集，则可以通过以下方式之一导入图像：

- 从本地计算机导入图像。图像不会被标注。
- 从 S3 桶导入图像。Amazon Lookout for Vision 可以使用包含图像的文件夹名称来对图像进行分类。使用 `normal` 来表示正常图像。使用 `anomaly` 来表示异常图像。您无法自动分配分割标签。
- 导入 Amazon SageMaker Ground Truth 清单文件。清单文件中的图像会被标注。您可以创建并导入自己的清单文件。如果图像数量很多，可以考虑使用 SageMaker Ground Truth 标注服务。然后，您可以导入 Amazon SageMaker Ground Truth 任务的输出清单文件。

标注图像

标签用于描述数据集中的图像。标签可以指明图像为正常还是异常（分类）。标签还可以描述图像上的异常位置（分割）。

如果图像未进行标注，您可以使用控制台来标注它们。

您为数据集中的图像分配的标签决定了 Lookout for Vision 会创建什么类型的模型。

图像分类

要创建图像分类模型，请使用 Lookout for Vision 控制台，将数据集中的图像分类为正常图像或异常图像。

您还可以使用 `CreateDataset` 操作，通过包含分类信息的清单文件创建数据集。

图像分割

要创建图像分割模型，请使用 Lookout for Vision 控制台，将数据集中的图像分类为正常图像或异常图像。您还需要为图像上的异常区域（如果存在）指定像素掩码，并为各个异常掩码指定异常标签。

您还可以使用 `CreateDataset` 操作，通过包含分割和分类信息的清单文件创建数据集。

如果您的项目具有单独的训练数据集和测试数据集，则 Lookout for Vision 会使用训练数据集来学习和确定模型类型。您应以相同的方式标注测试数据集中的图像。

更多信息：创建您的数据集。

训练您的模型

训练会创建模型，并训练它预测图像中是否存在异常。每次训练时都会创建一个新的模型版本。

训练开始时，Amazon Lookout for Vision 会选择最合适的算法来训练您的模型。模型会依次接受训练和测试。在开始使用 Amazon Lookout for Vision 中，您可以训练单数据集项目，该数据集会在内部进行拆分，从而创建训练数据集和测试数据集。您也可以创建具有单独训练数据集和测试数据集的项目。在此配置中，Amazon Lookout for Vision 会使用训练数据集训练您的模型，并使用测试数据集来测试模型。

> **Important**
>
> 您需要按照成功训练模型所花费的时间量付费。

更多信息：训练您的模型。

评估您的模型

使用在测试期间创建的性能指标，评估模型的性能。

使用性能指标，您可以更好地了解所训练模型的性能，并确定是否已准备好将其投入生产。

更多信息：改进您的模型。

如果性能指标表明需要改进，您可以通过用新图像来执行试用检测任务，从而添加更多训练数据。该任务完成后，您可以验证结果并将经过验证的图像添加到训练数据集中。或者，您可以将新的训练图像直接添加到数据集中。接下来，您可以重新训练模型并重新检查性能指标。

更多信息：使用试用检测任务验证您的模型。

使用您的模型

若要在AWS云中使用模型，需要先使用 StartModel 操作启动模型。您可以从控制台获取适用于您的模型的 `StartModel` CLI 命令。

更多信息：启动您的模型。

经过训练的 Amazon Lookout for Vision 模型可以预测输入图像包含正常内容还是异常内容。如果您的模型是分割模型，则预测中会包含异常掩码，用于标记发现异常的像素。

要使用您的模型进行预测，请调用 DetectAnomalies 操作并从本地计算机传递输入图像。您可以从控制台获取用于调用 `DetectAnomalies` 的 CLI 命令。

更多信息：检测图像中的异常。

> **Important**
>
> 您需要为模型的运行时间付费。

如果不再使用您的模型，请使用 StopModel 操作停止模型。您可以从控制台获取 CLI 命令。

更多信息：停止您的模型。

在边缘设备上使用您的模型

您可以在 AWS IoT Greengrass Version 2 核心设备上使用 Lookout for Vision 模型。

更多信息：在边缘设备上使用您的 Amazon Lookout for Vision 模型。

使用您的控制面板

您可以使用控制面板来获取所有项目的概览，以及各个项目的概览信息。

更多信息：使用您的控制面板。

开始使用 Amazon Lookout for Vision

在开始介绍这些开始使用 说明之前，我们建议您先阅读了解 Amazon Lookout for Vision。

这些“开始使用”说明将向您展示如何创建示例图像分割模型。如果要创建示例图像分类模型，请参阅图像分类数据集。

如果要快速试用示例模型，我们提供了示例训练图像和掩码图像。我们还提供了用于创建图像分割清单文件的 Python 脚本。您可以使用清单文件为自己的项目创建数据集，无需标注数据集中的图像。当使用自己的图像创建模型时，您必须标注数据集中的图像。有关更多信息，请参阅 创建您的数据集。

我们提供的图像是正常饼干图像和异常饼干图像。异常饼干的饼干形状上有裂纹。您使用图像训练的模型会预测一种分类（正常或异常），还会找到异常饼干中的裂纹区域（掩码），如以下示例所示。

主题

- 步骤 1：创建清单文件并上传图像
- 步骤 2：创建模型
- 步骤 3：启动模型
- 步骤 4：分析图像
- 步骤 5：停止模型
- 后续步骤

步骤 1：创建清单文件并上传图像

在此过程中，您会将 Amazon Lookout for Vision 文档存储库克隆到您的计算机上。然后，您可以使用 Python（版本 3.7 或更高）脚本创建清单文件，并将训练图像和掩码图像上传到您指定的 Amazon S3 位置。您可以使用清单文件创建自己的模型。随后，您可以使用本地存储库中的测试图像来试用模型。

创建清单文件并上传图像

1. 按照设置 Amazon Lookout for Vision 部分的说明，设置 Amazon Lookout for Vision。请务必安装适用于 Python 的 AWS SDK。
2. 在您要使用 Lookout for Vision 的 AWS 区域中，创建一个 S3 桶。
3. 在 Amazon S3 桶中，创建一个文件夹，名称为 `getting-started`。
4. 记下 Amazon S3 URI 和该文件夹的 Amazon 资源名称 (ARN)。您需要使用它们来设置权限和运行脚本。
5. 确保调用脚本的用户有权调用 `s3:PutObject` 操作。您可以使用以下策略。若要分配权限，请参阅分配权限。

```
{
 "Version": "2012-10-17",
 "Statement": [{
  "Sid": "Statement1",
  "Effect": "Allow",
  "Action": [
   "s3:PutObject"
  ],
  "Resource": [
   "arn:aws:s3::: ARN for S3 folder in step 4/*"
  ]
 }]
}
```

6. 确保您具有名为 `lookoutvision-access` 的本地配置文件，并且该配置文件的用户具有上一步中的权限。有关更多信息，请参阅 在本地计算机上使用配置文件。
7. 下载压缩文件：getting-started.zip。压缩文件包含入门数据集和设置脚本。
8. 解压缩 `getting-started.zip` 文件。
9. 在命令提示符下，执行以下操作：

   a. 导航到 `getting-started` 文件夹。

b. 运行以下命令以创建清单文件，并将训练图像和图像掩码上传到您在步骤 4 中记下的 Amazon S3 路径。

```
python getting_started.py S3-URI-from-step-4
```

c. 当脚本完成后，记下 `train.manifest` 文件的路径，该路径在脚本中显示在 `Create dataset using manifest file:` 之后。该路径应该类似于 `s3://path to getting started folder/manifests/train.manifest`。

步骤 2：创建模型

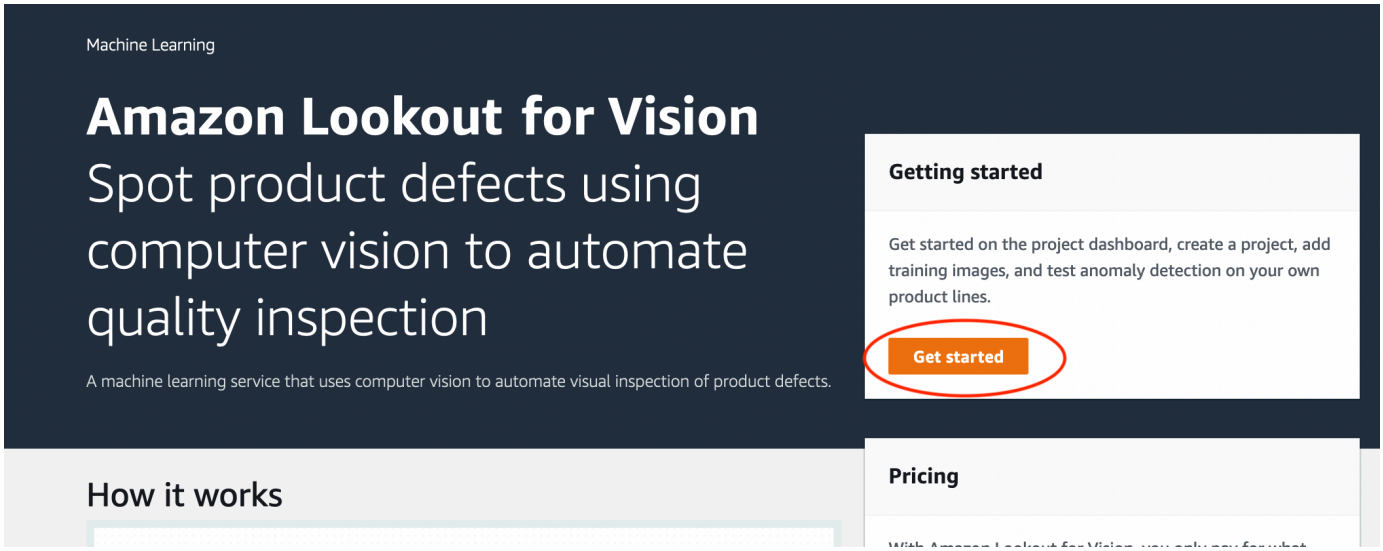
在此过程中，您将使用之前上传到 Amazon S3 桶的图像和清单文件，创建项目和数据集。然后，您将创建模型并查看模型训练的评估结果。

由于您使用入门清单文件创建数据集，因此无需标注数据集的图像。当使用自己的图像创建数据集时，您确实需要标注图像。有关更多信息，请参阅 标注图像。

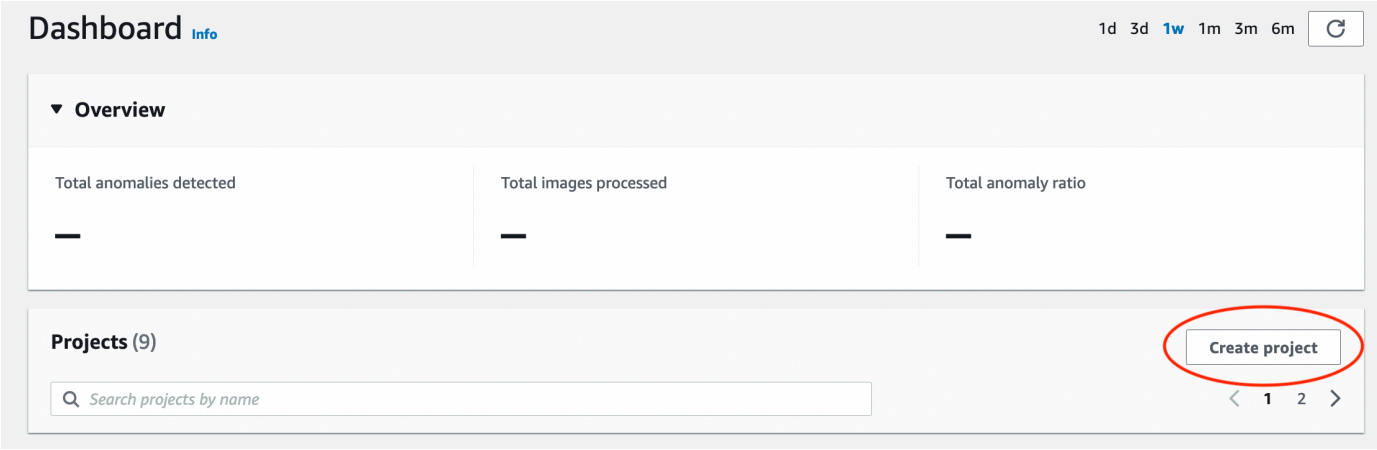
> **Important**
>
> 您需要为成功训练模型付费。

创建模型

1. 打开 Amazon Lookout for Vision 控制台，网址为 https://console.aws.amazon.com/lookoutvision/。
2. 确保您所在的 AWS 区域与您在步骤 1：创建清单文件并上传图像中创建 Amazon S3 桶时相同。要更改区域，请在导航栏上选择当前所显示区域的名称。然后，选择要切换到的区域。
3. 选择开始使用。



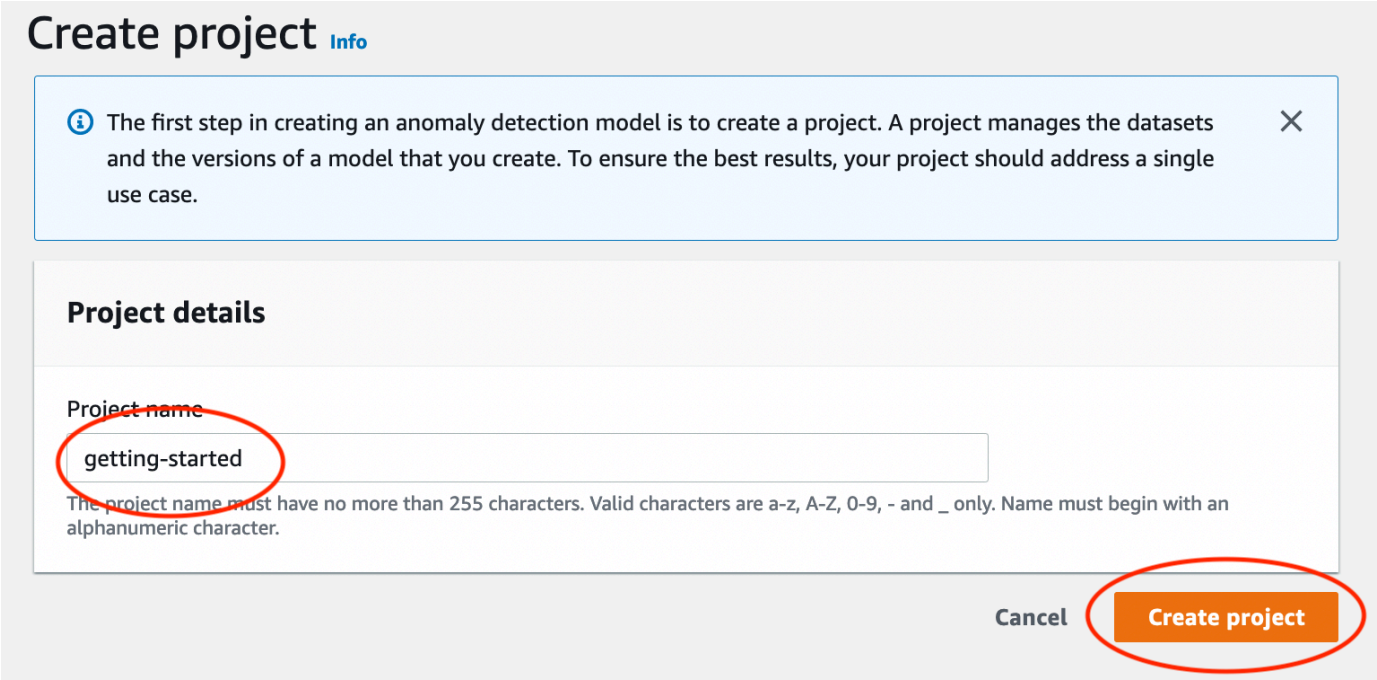
4. 在项目部分，选择创建项目。



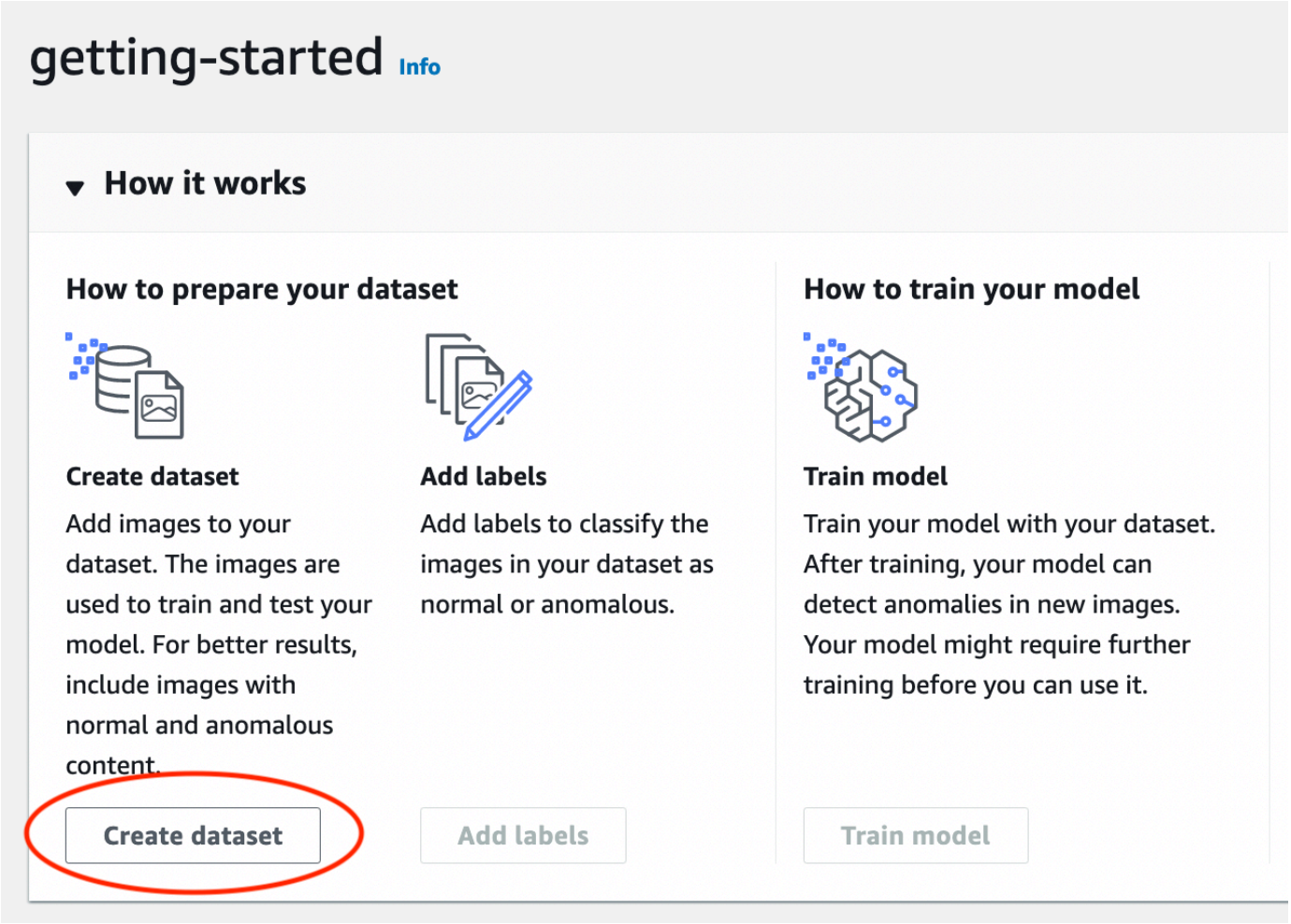
5. 在创建项目页面上，执行以下操作：

   a. 在项目名称中，输入 `getting-started`。

   b. 请选择 Create project（创建项目）。



6. 在项目页面上的工作原理部分，选择创建数据集。



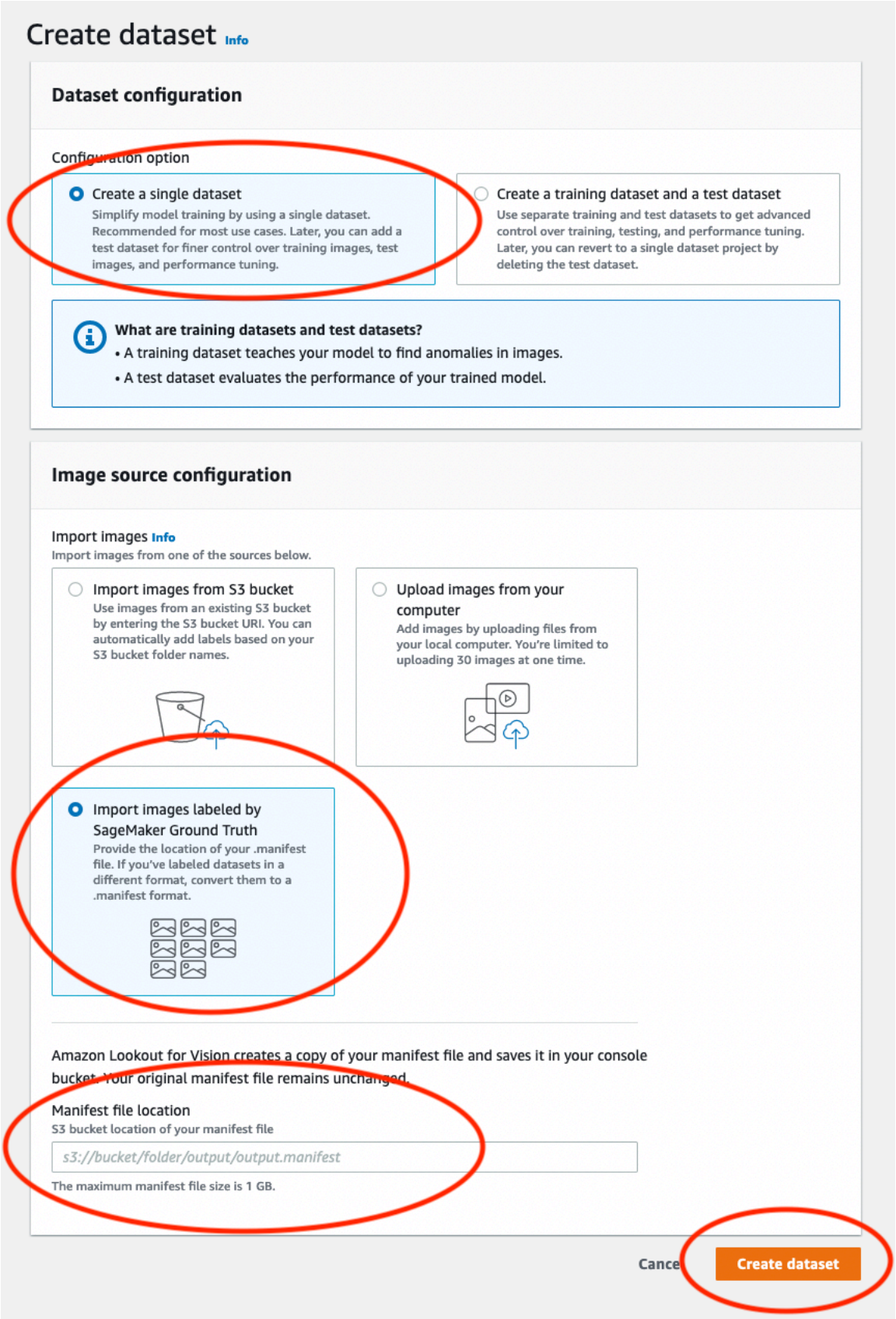
7. 在创建数据集页面上，执行以下操作：

   a. 选择创建单个数据集。

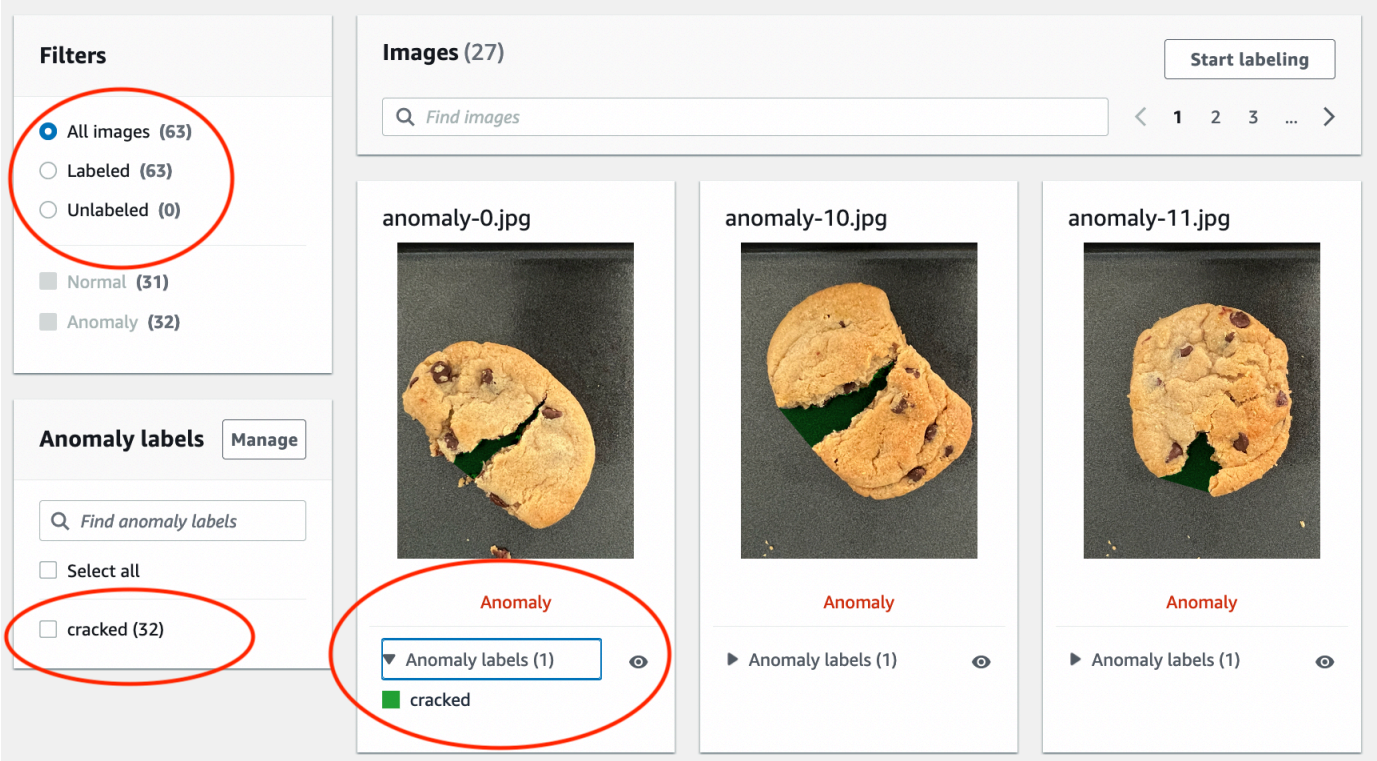
   b. 在图像源配置部分，选择导入由 SageMaker Ground Truth 标注的图像。

   c. 对于 .manifest 文件位置，请输入您在步骤 1：创建清单文件并上传图像步骤 6.c 中记下的清单文件的 Amazon S3 位置。Amazon S3 位置应该类似于 `s3://path to getting started folder/manifests/train.manifest`

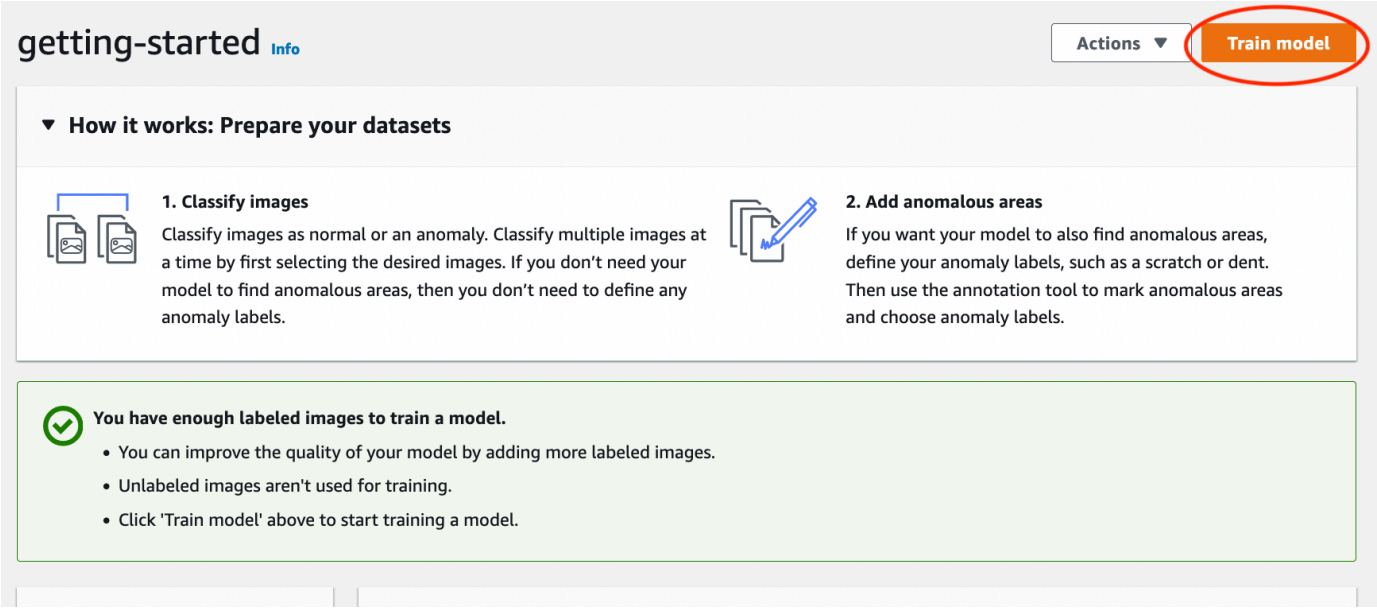
   d. 选择创建数据集。


Create dataset Info
Dataset configuration
Configuration option
Create a single dataset
Simplify model training by using a single dataset. Recommended for most use cases. Later, you can add a test dataset for finer control over training images, test images, and performance tuning.
Create a training dataset and a test dataset
Use separate training and test datasets to get advanced control over training, testing, and performance tuning. Later, you can revert to a single dataset project by deleting the test dataset.
What are training datasets and test datasets?
• A training dataset teaches your model to find anomalies in images.
• A test dataset evaluates the performance of your trained model.
Image source configuration
Import images Info
Import images from one of the sources below.
Import images from S3 bucket
Use images from an existing S3 bucket by entering the S3 bucket URI. You can automatically add labels based on your S3 bucket folder names.
Upload images from your computer
Add images by uploading files from your local computer. You're limited to uploading 30 images at one time.
Import images labeled by SageMaker Ground Truth
Provide the location of your .manifest file. If you've labeled datasets in a different format, convert them to a .manifest format.
Amazon Lookout for Vision creates a copy of your manifest file and saves it in your console bucket. Your original manifest file remains unchanged.
Manifest file location
S3 bucket location of your manifest file
s3://bucket/folder/output/output.manifest
The maximum manifest file size is 1 GB.
Cancel
Create dataset

8. 在项目详细信息页面上的图像部分，查看数据集图像。您可以查看每个数据集图像的分类和图像分割信息（掩码和异常标签）。您还可以搜索图像，按标注状态（已标注/未标注）筛选图像，或者按分配给图像的异常标签筛选图像。

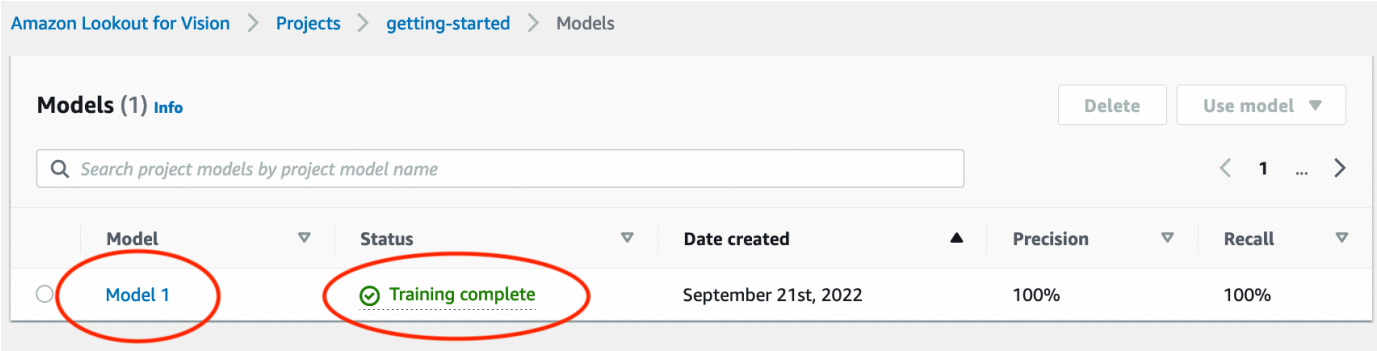


9. 在项目详细信息页面上，选择训练模型。



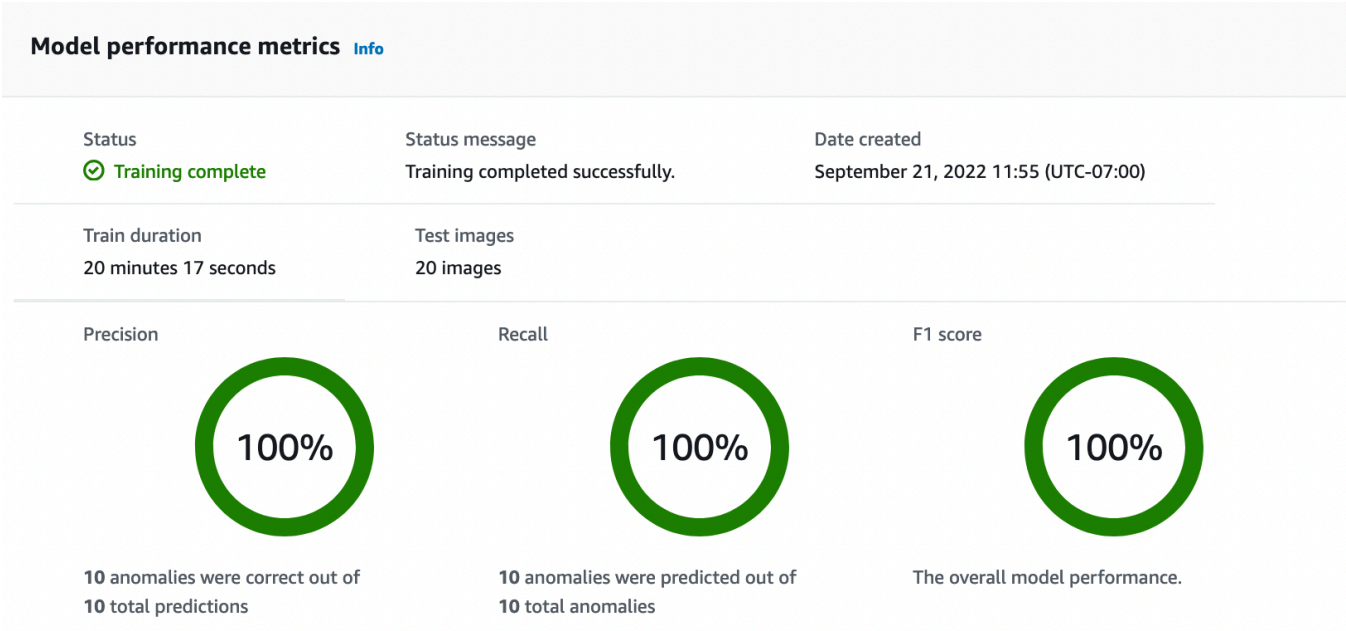
10. 在训练模型详细信息页面上，选择训练模型。
11. 在是否要训练您的模型？对话框中，选择训练模型。

12. 在项目模型页面中，您可以看到训练已经开始。通过查看模型版本的状态列，检查当前状态。完成模型训练至少需要 30 分钟。当状态更改为训练完成时，表示已成功完成训练。
13. 训练结束后，在模型页面中选择模型 1。

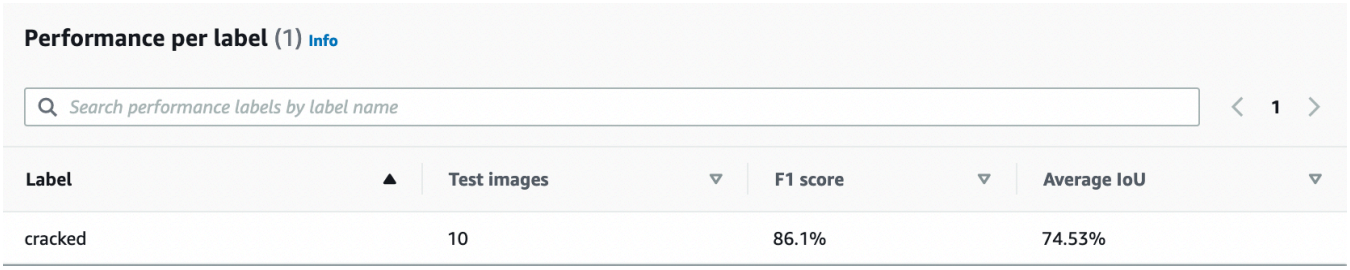


14. 在模型的详细信息页面中，查看性能指标选项卡中的评估结果。有以下方面的指标：

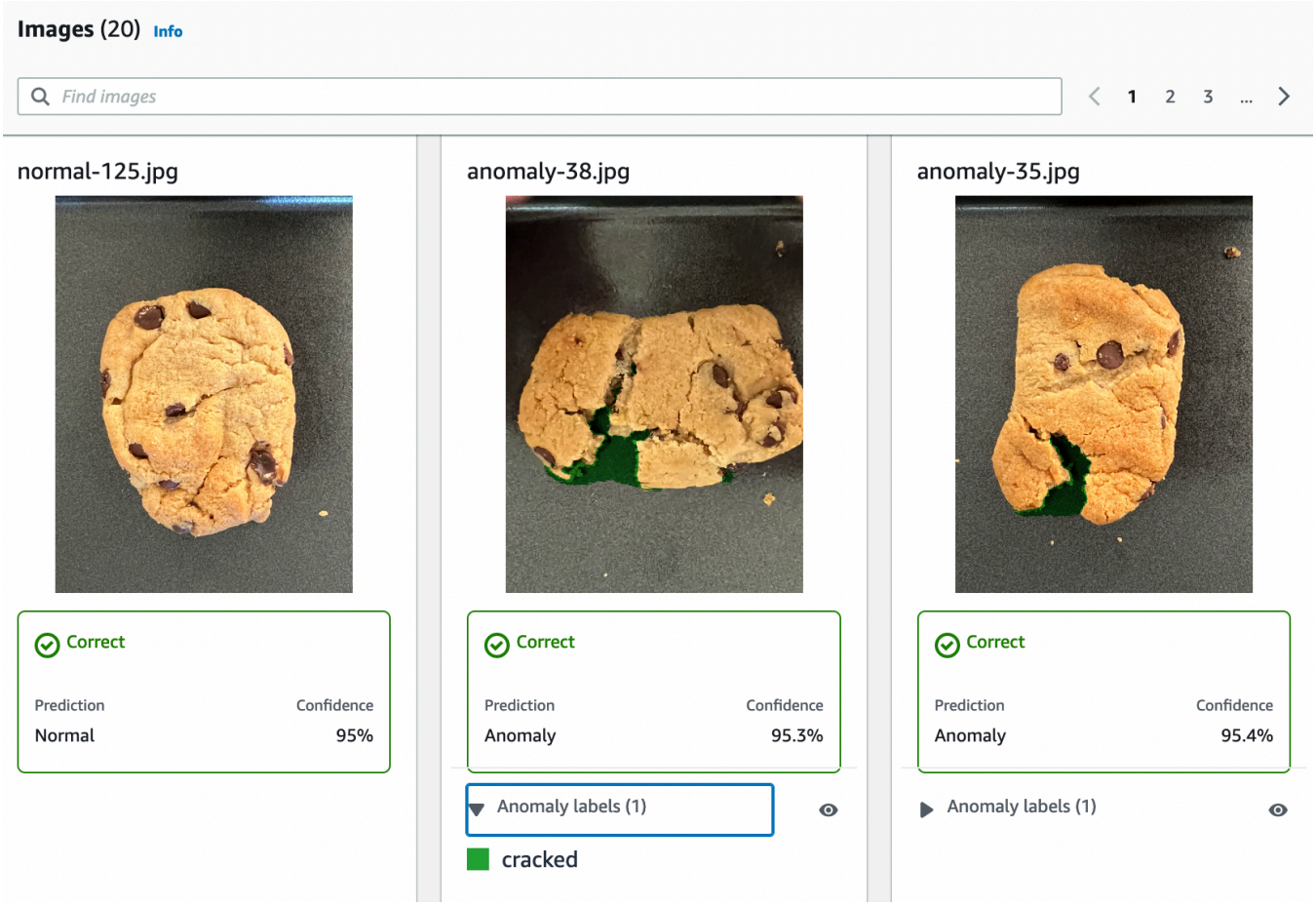
- 模型所做分类预测对应的整体模型性能指标（精度、召回率和 F1 分数）。



- 测试图像中发现的异常标签对应的性能指标（平均 IoU、F1 分数）



- 对测试图像所做的预测（分类、分割掩码和异常标签）



由于模型训练为非确定性，因此您的评估结果可能与此页面上显示的结果有所不同。有关更多信息，请参阅 改进您的 Amazon Lookout for Vision 模型。

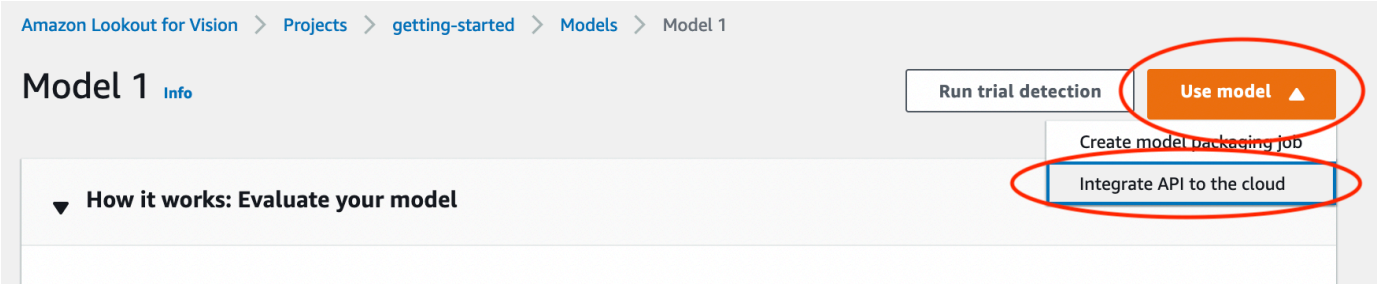
步骤 3：启动模型

在此步骤中，您将开始托管模型，以便它可以立即用来分析图像。有关更多信息，请参阅 运行经过训练的 Amazon Lookout for Vision 模型。

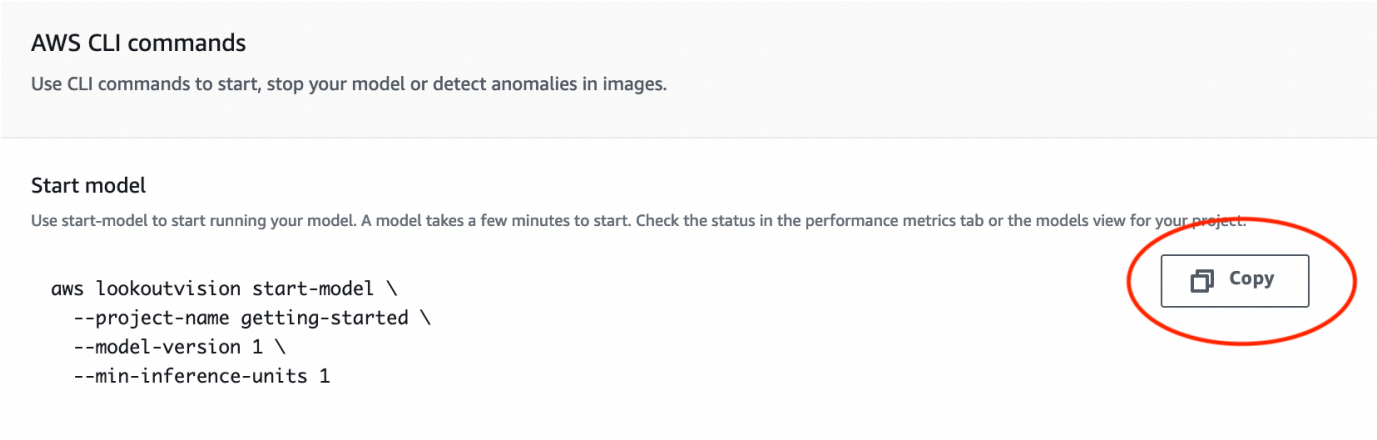
> **Note**
>
> 您需要按照模型的运行时间量付费。您将在步骤 5：停止模型中停止模型。

启动模型

1. 在模型的详细信息页面上，选择使用模型，然后选择将 API 集成到云。



2. 在 AWS CLI 命令部分，复制 `start-model` AWS CLI 命令。



3. 确保将 AWS CLI 配置为在您使用 Amazon Lookout for Vision 控制台的同一 AWS 区域内运行。要更改 AWS CLI 使用的 AWS 区域，请参阅安装软件 AWS 开发工具包。

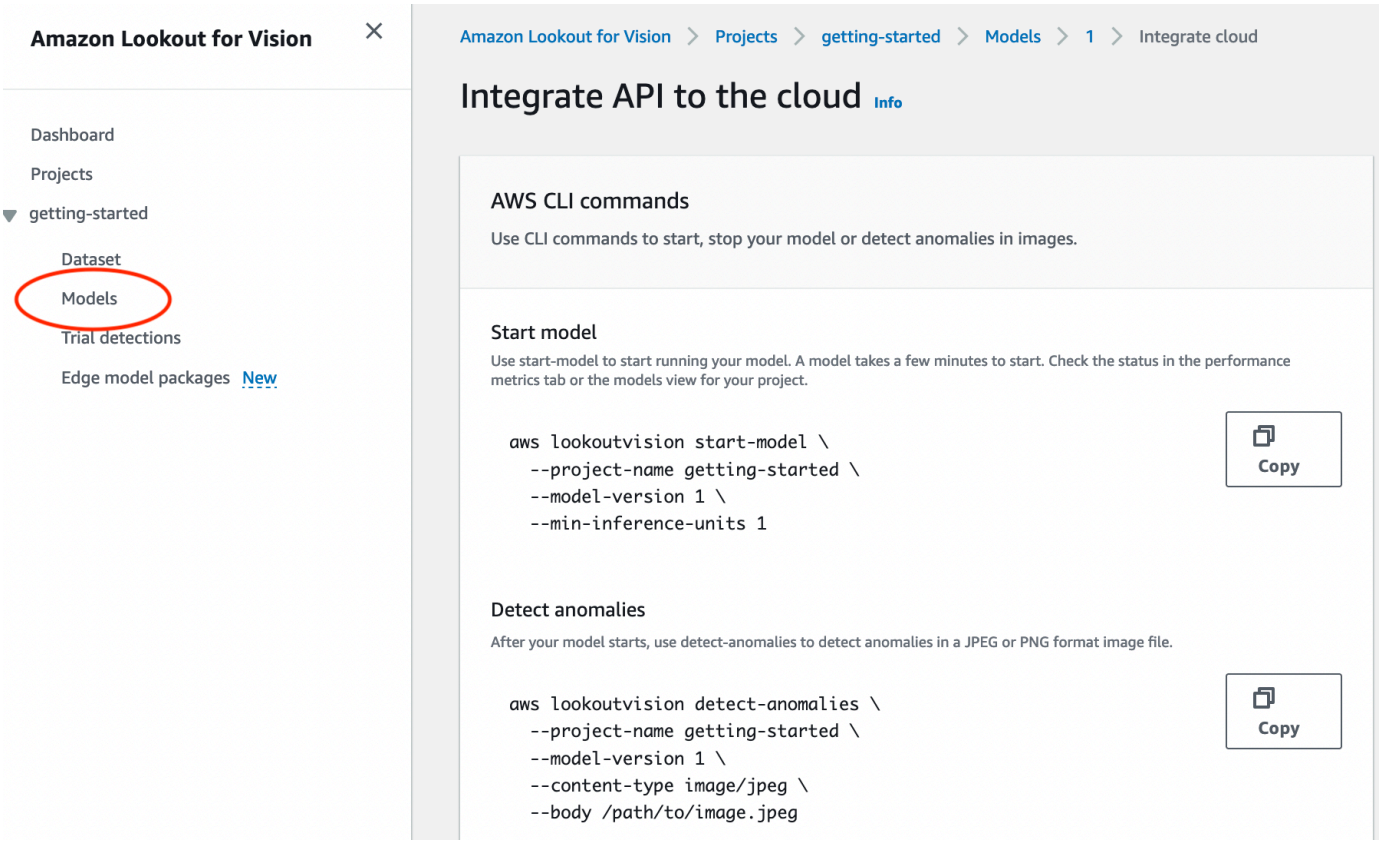
4. 在命令提示符下，输入 `start-model` 命令以启动模型。如果您使用 `lookoutvision` 配置文件来获取凭证，请添加 `--profile lookoutvision-access` 参数。例如：

```
aws lookoutvision start-model \
  --project-name getting-started \
  --model-version 1 \
  --min-inference-units 1 \
  --profile lookoutvision-access
```

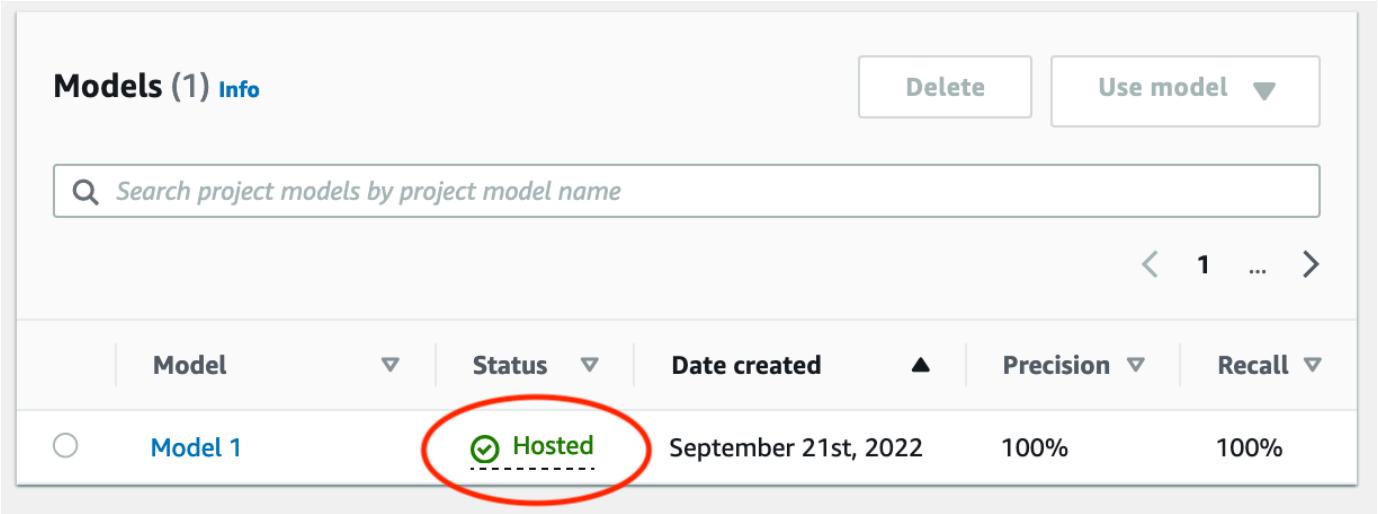
如果调用成功，将显示以下输出：

```
{
    "Status": "STARTING_HOSTING"
}
```

5. 返回控制台，在导航窗格中选择模型。



6. 等待状态列中模型（模型 1）的状态显示为已托管。如果您之前在项目中训练过模型，请等待最新的模型版本完成。

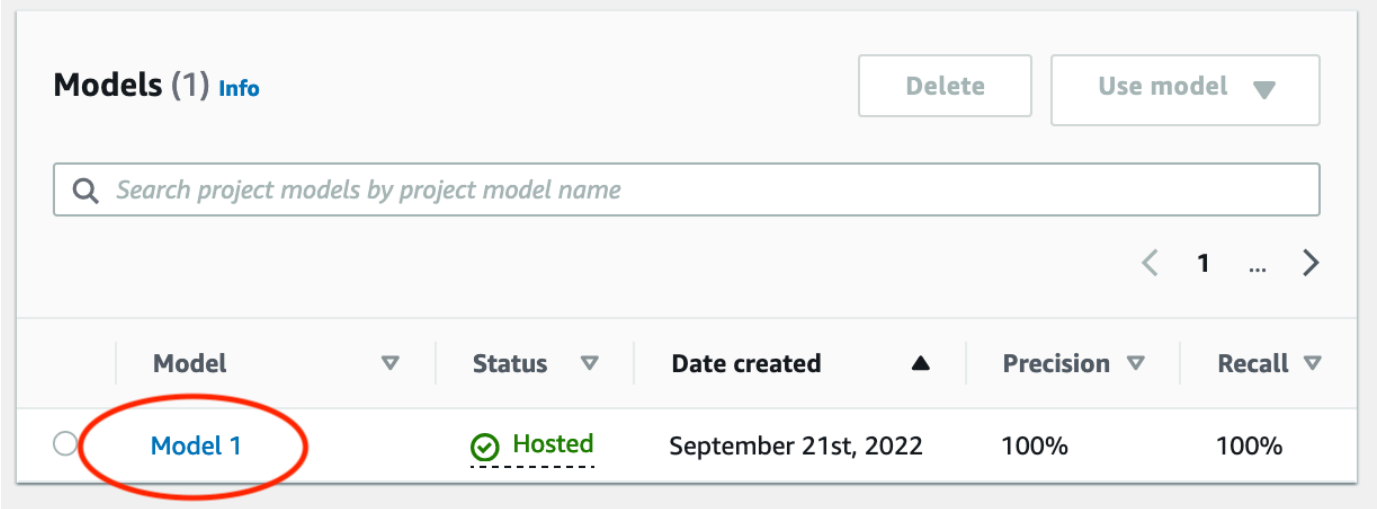


步骤 4：分析图像

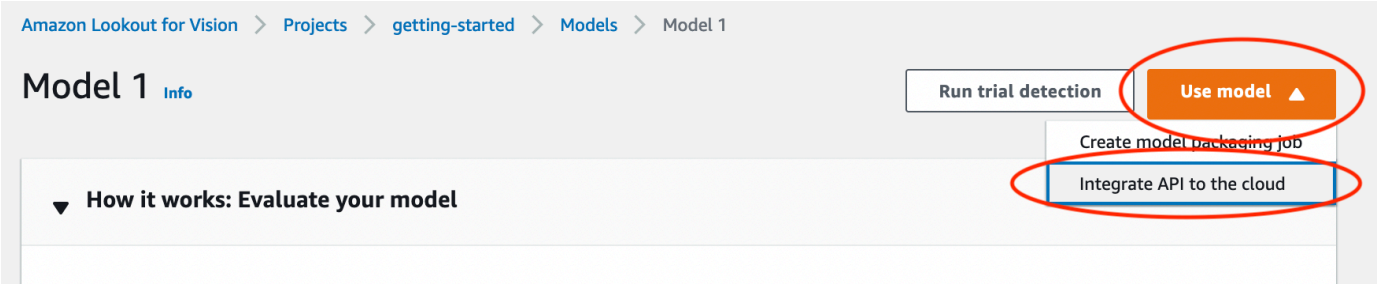
在此步骤中，您将使用自己的模型来分析图像。我们提供了一些示例图像供您使用，它们位于您计算机上 Lookout for Vision 文档存储库中的开始使用 `test-images` 文件夹中。有关更多信息，请参阅 检测图像中的异常。

分析图像

1. 在模型页面上，选择模型 1。



2. 在模型的详细信息页面上，选择使用模型，然后选择将 API 集成到云。



3. 在 AWS CLI 命令部分，复制 `detect-anomalies` AWS CLI 命令。

4. 在命令提示符处，输入上一步中的 detect-anomalies 命令以分析异常图像。对于 --body 参数，请指定您计算机上开始使用 test-images 文件夹中的异常图像。如果您使用 lookoutvision 配置文件来获取凭证，请添加 --profile lookoutvision-access 参数。例如：

```
aws lookoutvision detect-anomalies \
  --project-name getting-started \
  --model-version 1 \
  --content-type image/jpeg \
  --body /path/to/test-images/test-anomaly-1.jpg \
  --profile lookoutvision-access
```

该输出值应该类似于以下内容：

```
{
    "DetectAnomalyResult": {
        "Source": {
            "Type": "direct"
        },
        "IsAnomalous": true,
        "Confidence": 0.983975887298584,
        "Anomalies": [
            {
                "Name": "background",
                "PixelAnomaly": {
                    "TotalPercentageArea": 0.9818974137306213,
                    "Color": "#FFFFFF"
                }
            },
            {
                "Name": "cracked",
                "PixelAnomaly": {
                    "TotalPercentageArea": 0.018102575093507767,
                    "Color": "#23A436"
                }
            }
        ],
        "AnomalyMask": "iVBORw0KGgoAAAANSUhEUgAAAkAAAAMACA......"
    }
}
```

5. 在输出中，请注意以下内容：

- IsAnomalous 是所预测分类的布尔值。如果图像异常，则为 true，否则为 false。
- Confidence 是浮点值，表示 Amazon Lookout for Vision 对预测的置信度。0 表示最低置信度，1 表示最高置信度。
- Anomalies 是在图像中发现的异常列表。Name 是异常标签。PixelAnomaly 包括异常区域总百分比 (TotalPercentageArea) 和异常标签的颜色 (Color)。列表中还包括“背景”异常，覆盖图像上发现的异常之外的区域。
- AnomalyMask 是掩码图像，用于显示所分析图像上的异常位置。

您可以使用响应中的信息，显示所分析图像和异常掩码的融合，如以下示例所示。有关示例代码，请参阅 显示分类和分割信息。


Classification:
Prediction: Anomalous
Confidence: 99.9%
Segmentation:
Anomaly: cracked. Area: 6.2%

6. 在命令提示符下，对开始使用 test-images 文件夹中的正常图像进行分析。如果您使用 lookoutvision 配置文件来获取凭证，请添加 --profile lookoutvision-access 参数。例如：

```
aws lookoutvision detect-anomalies \
  --project-name getting-started \
  --model-version 1 \
  --content-type image/jpeg \
  --body /path/to/test-images/test-normal-1.jpg \
  --profile lookoutvision-access
```

该输出值应该类似于以下内容：


```
{
    "DetectAnomalyResult": {
        "Source": {
            "Type": "direct"
        },
        "IsAnomalous": false,
        "Confidence": 0.9916400909423828,
        "Anomalies": [
            {
                "Name": "background",
                "PixelAnomaly": {
                    "TotalPercentageArea": 1.0,
                    "Color": "#FFFFFF"
                }
            }
        ],
        "AnomalyMask": "iVBORw0KGgoAAAANSUhEUgAAAkAAAA....."
    }
}
```

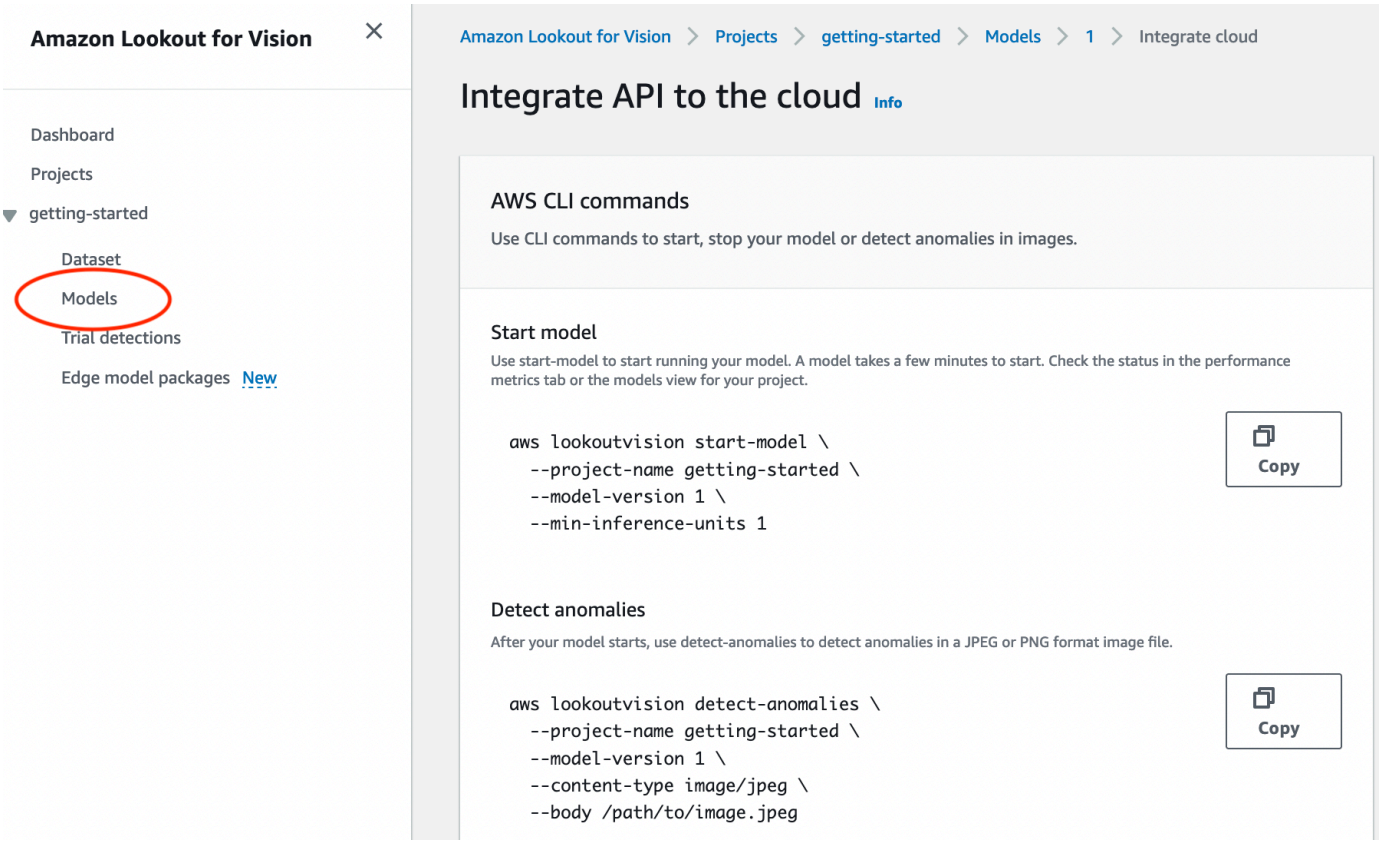

7. 在输出中，注意 IsAnomalous 的值为 false，表示将该图像分类为无异常。使用 Confidence 有助于确定您对分类的信心。此外，Anomalies 数组只有 background 异常标签。

步骤 5：停止模型

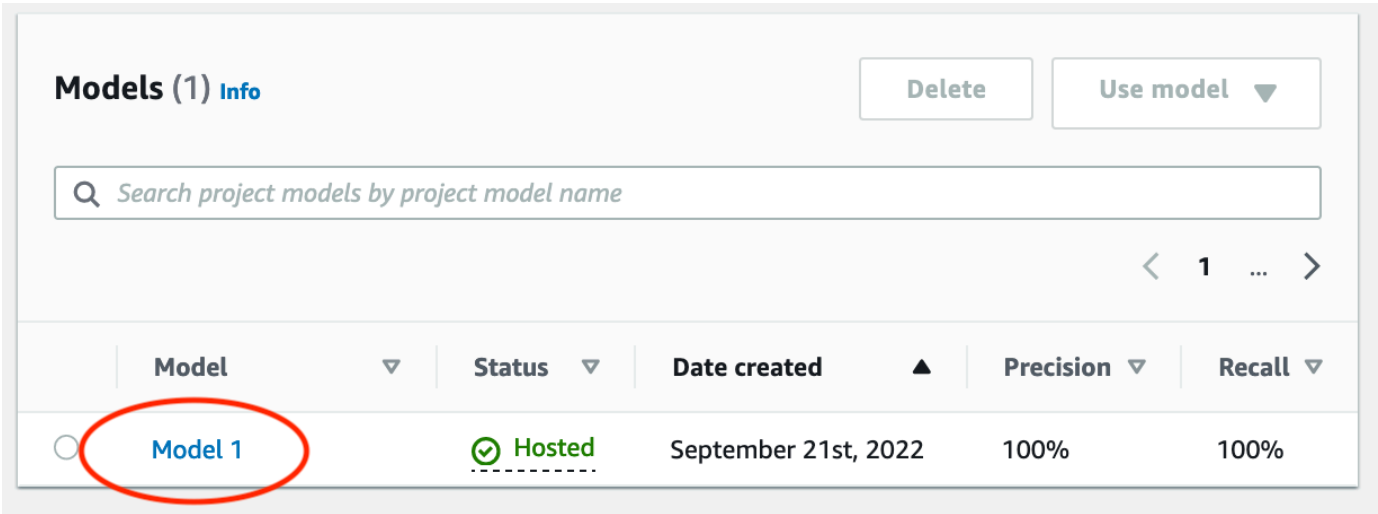
在此步骤中，您将停止模型。您需要按照模型的运行时间量付费。如果不使用模型，您应将其停止。您可以在下次需要时重新启动模型。有关更多信息，请参阅 启动您的 Amazon Lookout for Vision 模型。

停止模型

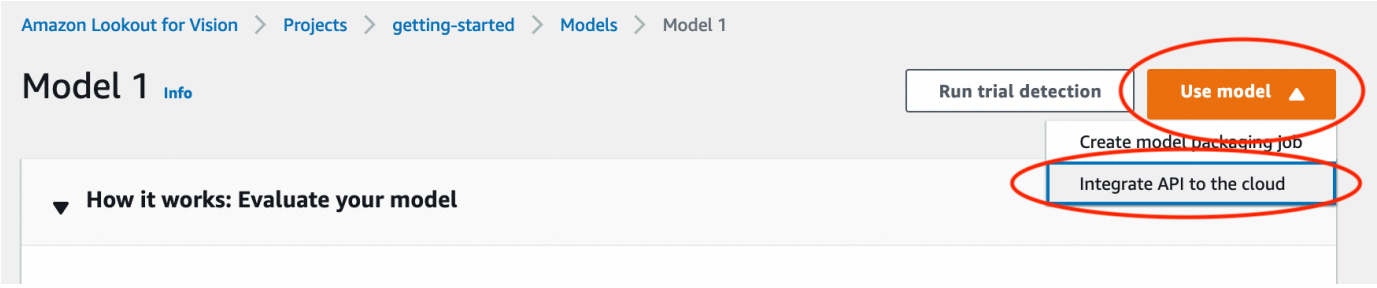
1. 在导航窗格中，选择模型。



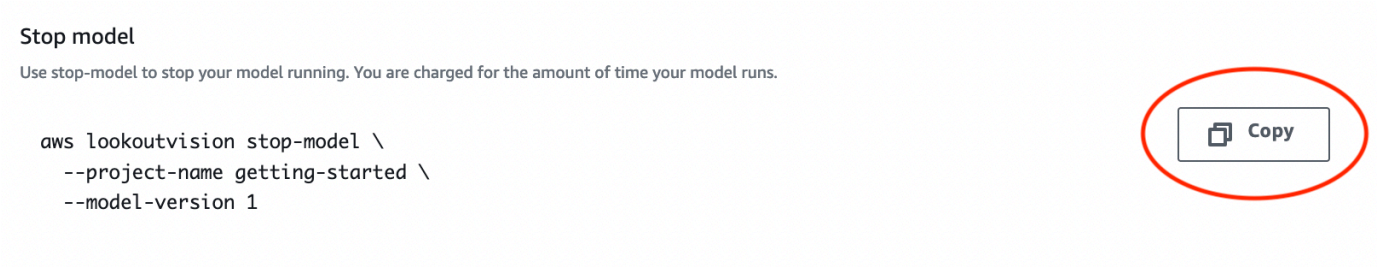
2. 在模型页面中，选择模型 1。



3. 在模型的详细信息页面上，选择使用模型，然后选择将 API 集成到云。



4. 在 AWS CLI 命令部分，复制 `stop-model` AWS CLI 命令。



5. 在命令提示符下，输入上一步中的 `stop-model` AWS CLI 命令以停止模型。如果您使用 `lookoutvision` 配置文件来获取凭证，请添加 `--profile lookoutvision-access` 参数。例如：

```
aws lookoutvision stop-model \
  --project-name getting-started \
  --model-version 1 \
  --profile lookoutvision-access
```

如果调用成功，将显示以下输出：

```
{
    "Status": "STOPPING_HOSTING"
}
```

6. 返回控制台，在左侧导航页面中选择模型。
7. 当状态列中的模型状态为训练完成时，表示模型已停止。

后续步骤

当您准备好使用自己的图像创建模型后，请先按照创建您的项目中的说明进行操作。这些说明中包括使用 Amazon Lookout for Vision 控制台和 AWS SDK 创建模型的步骤。

如果您想尝试其他示例数据集，请参阅示例代码和数据集。

创建您的 Amazon Lookout for Vision 模型

Amazon Lookout for Vision 模型是一种机器学习模型，它通过发现用于训练模型的图像中的模式，以此预测新图像中是否存在异常。本节将向您介绍如何创建和训练模型。训练模型后，您需要评估它的性能。有关更多信息，请参阅 改进您的 Amazon Lookout for Vision 模型。

在创建第一个模型之前，我们建议您阅读 了解 Amazon Lookout for Vision 和 开始使用 Amazon Lookout for Vision。如果要使用 AWS SDK，请阅读 调用 Amazon Lookout for Vision 操作。

主题

- 创建您的项目
- 创建您的数据集
- 标注图像
- 训练您的模型
- 模型训练问题排查

创建您的项目

Amazon Lookout for Vision 项目是创建和管理 Lookout for Vision 模型所需的一组资源。项目用于管理以下内容：

- 数据集：用于训练模型的图像和图像标签。有关更多信息，请参阅 创建您的数据集。
- 模型：训练用来检测异常的软件。一个模型可以有多个版本。有关更多信息，请参阅 训练您的模型。

我们建议您将一个项目用于一种使用场景，如检测单一类型机器零件中的异常。

> **Note**
>
> 您可以使用 AWS CloudFormation 来配置和配置 Amazon Lookout for Vision 项目。有关更多信息，请参阅 使用 AWS CloudFormation 创建 Amazon Lookout for Vision 资源。

要查看您的项目，请参阅 查看您的项目 或打开 使用 Amazon Lookout for Vision 控制面板。要删除模型，请参阅 删除模型。

主题

- 创建项目（控制台）
- 创建项目（SDK）

创建项目（控制台）

下面的过程展示了如何使用控制台创建项目。

创建项目（控制台）

1. 打开 Amazon Lookout for Vision 控制台，网址为 https://console.aws.amazon.com/lookoutvision/。
2. 在左侧导航窗格中，选择项目。
3. 选择创建项目。
4. 在项目名称中输入项目名称。
5. 选择创建项目。此时将显示项目的详细信息页面。
6. 按照 创建您的数据集 中的步骤，创建您的数据集。

创建项目（SDK）

您可以使用该CreateProject操作来创建 Amazon Lookout for Vision 项目。`CreateProject` 的响应包含项目名称和项目的 Amazon 资源名称（ARN）。之后，致电CreateDataset将训练和测试数据集添加到您的项目中。有关更多信息，请参阅 使用清单文件创建数据集（SDK）。

要查看您已创建的项目，请调用 `ListProjects`。有关更多信息，请参阅 查看您的项目。

创建项目 (SDK)

1. 如果您尚未这样做，请安装和配置和 AWS SDK。 AWS CLI 有关更多信息，请参阅 步骤 4：设置 AWS CLI 和 AWS 软件开发工具包。
2. 使用以下示例代码创建模型。

   CLI

   将 `project-name` 的值更改为要对项目使用的名称。

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

```
  --profile lookoutvision-access
```

Python

此代码取自 AWS 文档 SDK 示例 GitHub 存储库。请在此处查看完整示例。

```
    @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
```

Java V2

此代码取自 AWS 文档 SDK 示例 GitHub 存储库。请在此处查看完整示例。

```
/**
 * Creates an Amazon Lookout for Vision project.
 *
 * @param lfvClient   An Amazon Lookout for Vision client.
 * @param projectName The name of the project that you want to create.
 * @return ProjectMetadata Metadata information about the created project.
 */
public static ProjectMetadata createProject(LookoutVisionClient lfvClient,
 String projectName)
                throws LookoutVisionException {
```

```
        logger.log(Level.INFO, "Creating project: {0}", projectName);
        CreateProjectRequest createProjectRequest =
 CreateProjectRequest.builder().projectName(projectName)
                        .build();

        CreateProjectResponse response =
 lfvClient.createProject(createProjectRequest);

        logger.log(Level.INFO, "Project created. ARN: {0}",
 response.projectMetadata().projectArn());

        return response.projectMetadata();

}
```

3. 按照 使用 Amazon G SageMaker round Truth 清单文件创建数据集 中的步骤，创建您的数据集。

创建您的数据集

数据集包含用于训练和测试模型的图像及分配的标签。您可以使用 Amazon Lookout for Vision 控制台或通过操作为项目创建数据集CreateDataset。必须根据要创建的模型类型（图像分类或图像分割），对数据集图像进行标注。

主题

- 为数据集准备图像
- 创建数据集
- 使用存储在本地计算机上的图像创建数据集
- 使用存储在 Amazon S3 桶中的图像创建数据集
- 使用 Amazon G SageMaker round Truth 清单文件创建数据集

为数据集准备图像

您需要一组图像才能创建数据集。您的图像必须是 PNG 或 JPEG 格式的文件。关于需要的图像数量和类型，取决于您的项目具有单个数据集还是单独的训练数据集和测试数据集。

单数据集项目

要创建图像分类模型，您需要满足以下条件才能开始训练：

- 至少有 20 张正常对象的图像。
- 至少有 10 张异常对象的图像。

要创建图像分割模型，您需要满足以下条件才能开始训练：

- 每种异常类型至少各 20 张图像。
- 每张异常图像（存在异常类型的图像）必须只有一种异常类型。
- 至少有 20 张正常对象的图像。

单独训练数据集和测试数据集项目

要创建图像分类模型，您需要满足以下条件：

- 训练数据集中至少有 10 张正常对象的图像。
- 测试数据集中至少有 10 张正常对象的图像。
- 测试数据集中至少有 10 张异常对象的图像。

要创建图像分割模型，您需要满足以下条件：

- 每个数据集中每种异常类型需要至少各有 10 张图像。
- 每张异常图像（存在异常类型的图像）必须仅包含一种异常类型。
- 每个数据集必须至少有 10 张正常对象的图像。

要创建更高质量的模型，请使用超过最低数量的图像。如果要创建分割模型，我们建议您包含多种异常类型的图像，但这些图像不计入 Lookout for Vision 开始训练所需的最低数量值。

您的图像中应该包含单一类型的对象。此外，还应确保一致的图像捕获条件，如摄像机定位、照明和物体姿态。

训练及测试数据集中的所有图像必须尺寸相同。随后，使用已训练的模型分析的图像必须与训练及测试数据集图像尺寸相同。有关更多信息，请参阅 检测图像中的异常。

所有训练及测试图像都必须是唯一的图像，图像中最好是独特的物体。正常图像应捕获所分析对象的正常变化。异常图像应捕获异常的多样性采样。

Amazon Lookout for Vision 提供了可供您使用的示例图像。有关更多信息，请参阅 图像分类数据集。

有关图像限制，请参阅 配额。

创建数据集

在为项目创建数据集时，您应选择项目的初始数据集配置。您还要选择供 Lookout for Vision 导入图像的位置。

为项目选择数据集配置

在项目中创建第一个数据集时，您可以选择以下一种数据集配置：

- 单数据集：单数据集项目使用单个数据集来训练和测试您的模型。使用单个数据集可以让 Amazon Lookout for Vision 来选择训练图像和测试图像，从而简化训练工作。在训练期间，Amazon Lookout for Vision 会在内部将数据集拆分为训练数据集和测试数据集。您无权访问拆分后的数据集。对于大多数场景，我们建议使用单数据集项目。
- 单独的训练数据集和测试数据集：如果要更好地控制训练、测试和性能调优，您可以将项目配置为采用单独的训练数据集和测试数据集。如果要控制用于测试的图像，或者您已经有想要使用的基准图像组，请使用单独的测试数据集。

您可以向现有单数据集项目中添加测试数据集。然后，单数据集将成为训练数据集。如果从具有单独训练数据集和测试数据集的项目中移除测试数据集，则项目将成为单数据集项目。有关更多信息，请参阅删除数据集。

导入图像

创建数据集时，您应选择从何处导入图像。图像可能已经进行标注，具体取决于如何导入图像。如果在创建数据集后未标注图像，请参阅 标注图像。

您可以创建数据集，并通过以下方式之一导入其图像：

- 从本地计算机导入图像。图像不会被标注。您应使用 Lookout for Vision 控制台添加标签。
- 从 S3 桶导入图像。Amazon Lookout for Vision 可以使用文件夹名称来对图像进行分类，从而标注图像。使用 `normal` 来表示正常图像。使用 `anomaly` 来表示异常图像。您无法自动分配分割标签。
- 导入 Amazon G SageMaker round Truth 清单文件，其中包括带标签的图片。您可以创建并导入自己的清单文件。如果你有很多图片，可以考虑使用 G SageMaker round Truth 标签服务。然后，您可以从 Amazon G SageMaker round Truth 任务中导入输出清单文件。如有必要，您可以使用 Lookout for Vision 控制台来添加或更改标签。

如果您使用的是 AWS 软件开发工具包，则可以使用 Amazon G SageMaker round Truth 清单文件创建数据集。有关更多信息，请参阅 使用 Amazon G SageMaker round Truth 清单文件创建数据集。

在创建数据集后，如果图像已进行标注，您便可以训练模型。如果图像未进行标注，请根据要创建的模型类型来添加标签。有关更多信息，请参阅 标注图像。

您可以向现有数据集中添加更多图像。有关更多信息，请参阅 向您的数据集中添加图像。

使用存储在本地计算机上的图像创建数据集

您可以使用直接从计算机加载的图像来创建数据集。一次最多可以上传 30 张图像。在此过程中，您可以创建单数据集项目，或具有单独训练数据集和测试数据集的项目。

> **Note**
>
> 如果您刚完成 创建您的项目，则控制台应该会显示您的项目控制面板，并且您无需执行步骤 1-3。

使用本地计算机上的图像创建数据集（控制台）

1. 打开 Amazon Lookout for Vision 控制台，网址为 https://console.aws.amazon.com/lookoutvision/。
2. 在左侧导航窗格中，选择项目。
3. 在项目页面上，选择要向其添加数据集的项目。
4. 在项目详细信息页面上，选择创建数据集。
5. 选择单数据集选项卡或单独的训练数据集和测试数据集选项卡，然后按照步骤进行操作。

   Single dataset

   a. 在数据集配置部分，选择创建单个数据集。
   b. 在图像源配置部分，选择从您的计算机上传图像。
   c. 选择创建数据集。
   d. 在数据集页面上，选择添加图像。
   e. 从您的计算机文件中，选择要上传到数据集的图像。可以从本地计算机拖动或选择要上传的图像。
   f. 选择上传图像。

Separate training and test datasets

a. 在数据集配置部分，选择创建训练数据集和测试数据集。

b. 在训练数据集详细信息部分，选择从您的计算机上传图像。

c. 在测试数据集详细信息部分中，选择从您的计算机上传图像。

> **Note**
>
> 您的训练数据集和测试数据集可以有不同的图像源。

d. 选择创建数据集。此时将出现一个数据集页面，其中具有训练选项卡和测试选项卡，分别代表各自的数据集。

e. 选择操作，然后选择将图像添加到训练数据集。

f. 选择要上传到数据集的图像。可以从本地计算机拖动或选择要上传的图像。

g. 选择上传图像。

h. 重复步骤 5e-5g。在步骤 5e 中，选择操作，然后选择将图像添加到测试数据集。

6. 按照 标注图像 中的步骤，标注您的图像。

7. 按照 训练您的模型 中的步骤，训练您的模型。

使用存储在 Amazon S3 桶中的图像创建数据集

您可以使用存储在 Amazon S3 桶中的图像创建数据集。使用此选项，您可以利用 Amazon S3 桶中的文件夹结构自动对图像进行分类。您可以将图像存储在控制台桶中，或自己账户内的其他 Amazon S3 桶中。

设置用于自动标注的文件夹

在创建数据集期间，您可以选择根据包含图像的文件夹的名称，为图像分配标签名称。这些文件夹必须是您在创建数据集时，在 S3 URI 中指定的 Amazon S3 文件夹路径的子文件夹。

以下是“开始使用”示例图像的 `train` 文件夹。如果您将 Amazon S3 文件夹位置指定为 `S3-bucket/circuitboard/train/`，则 `normal` 文件夹中的图像会分配到 `Normal` 标签。`anomaly` 文件夹中的图像会分配到 `Anomaly` 标签。不会使用更深层子文件夹的名称来标注图像。

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

使用 Amazon S3 桶中的图像创建数据集

以下过程将使用存储在 Amazon S3 桶中的分类示例图像，创建一个数据集。要使用自己的图像，请创建设置用于自动标注的文件夹中描述的文件夹结构。

该过程还展示了如何创建单数据集项目，或采用单独训练数据集和测试数据集的项目。

如果您不选择自动标注图像，则需要在创建数据集后对图像进行标注。有关更多信息，请参阅 图像分类（控制台）。

> **Note**
>
> 如果您刚完成 创建您的项目，则控制台应该会显示您的项目控制面板，并且您无需执行步骤 1-4。

使用存储在 Amazon S3 桶中的图像创建数据集

1. 将“开始使用”图像上传到您的 Amazon S3 桶（如果尚未如此）。有关更多信息，请参阅 图像分类数据集。
2. 打开 Amazon Lookout for Vision 控制台，网址为 https://console.aws.amazon.com/lookoutvision/。
3. 在左侧导航窗格中，选择项目。
4. 在项目页面上，选择要向其添加数据集的项目。此时将显示项目的详细信息页面。

5. 选择创建数据集。此时将显示创建数据集页面。

> **Tip**
>
> 如果要按照“开始使用”说明进行操作，请选择创建训练数据集和测试数据集。

6. 选择单数据集选项卡或单独的训练数据集和测试数据集选项卡，然后按照步骤进行操作。

   Single dataset

   a. 在数据集配置部分，选择创建单个数据集。

   b. 在图像源配置部分，输入步骤 7-9 的信息。

   Separate training and test datasets

   a. 在数据集配置部分，选择创建训练数据集和测试数据集。

   b. 对于您的训练数据集，请在训练数据集详细信息部分输入步骤 7-9 的信息。

   c. 对于您的测试数据集，请在测试数据集详细信息部分输入步骤 7-9 的信息。

> **Note**
>
> 您的训练数据集和测试数据集可以有不同的图像源。

7. 选择从 Amazon S3 桶导入图像。
8. 在 S3 URI 中，输入 Amazon S3 桶的位置和文件夹路径。将 `bucket` 更改为您的 Amazon S3 桶名称。

   a. 如果要创建单数据集项目或训练数据集，请输入以下内容：

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

   b. 如果要创建测试数据集，请输入以下内容：

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

9. 选择根据文件夹自动为图像附加标签。
10. 选择创建数据集。此时将打开一个数据集页面，其中包含您的已标注图像。
11. 按照 训练您的模型 中的步骤，训练您的模型。

使用 Amazon G SageMaker round Truth 清单文件创建数据集

清单文件中包含与可用于训练和测试模型的图像和图像标签相关的信息。您可以在 Amazon S3 桶中存储清单文件，并使用它来创建数据集。您可以创建自己的清单文件，也可以使用现有的清单文件，例如 Amazon G SageMaker round Truth 任务的输出。

主题

- 使用 Amazon SageMaker Ground Truth 任务
- 创建清单文件

使用 Amazon SageMaker Ground Truth 任务

标注图像可能会花费大量时间。例如，在异常周围精确绘制掩码可能会花费数十秒。如果有数百张图像，则可能需要几个小时才完成标注。除了自己给图片贴标签之外，还可以考虑使用 Amazon G SageMaker round Truth。

借助 Amazon G SageMaker round Truth，你可以使用 Amazon Mechanical Turk（你选择的供应商公司）的工作人员或内部的私人员工来创建一组带标签的图像。有关更多信息，请参阅使用 Amazon G SageMaker round Truth 为数据添加标签。

使用 Amazon Mechanical Turk 需要付费。此外，完成 Amazon Ground Truth 标注作业可能需要几天时间。如果存在成本问题，或者您需要快速训练模型，我们建议使用 Amazon Lookout for Vision 控制台来标注图像。

您可以使用 Amazon G SageMaker round Truth 标签作业来标记适用于图像分类模型和图像分割模型的图片。任务完成后，您应使用输出清单文件创建 Amazon Lookout for Vision 数据集。

图像分类

要为图像分类模型标注图像，请为图像分类（单标签）任务创建标注作业。

图像分割

要为图像分割模型标注图像，请为图像分类（单标签）任务创建标注作业。然后，将该作业链接起来，从而为图像语义分割任务创建标注作业。

您也可以使用标注作业为图像分割模型创建部分清单文件。例如，您可以使用图像分类（单标签）任务对图像进行分类。在使用该作业的输出创建 Lookout for Vision 数据集后，应使用 Amazon Lookout for Vision 控制台向数据集图像添加分割掩码和异常标签。

使用 Amazon G SageMaker round Truth 为图片贴标

以下过程展示了如何使用 Amazon G SageMaker round Truth 图片标签任务为图片添加标签。该过程会创建一个图像分类清单文件，然后可选择将图像标注任务链接起来，从而创建图像分割清单文件。如果希望您的项目采用单独的测试数据集，请重复此过程，以便为测试数据集创建清单文件。

使用 Amazon G SageMaker round Truth（控制台）为图片添加标签

1. 按照创建标注作业（控制台）中的说明，为图像分类（单标签）任务创建 Ground Truth 作业。
   a. 在步骤 10 中，从任务类别下拉菜单中选择图像，然后选择图像分类（单标签）作为任务类型。
   b. 在步骤 16 中，在图像分类（单标签）标注工具部分，添加两个标签：正常和异常。
2. 等待工作人员完成图像分类。
3. 如果要为图像分割模型创建数据集，请执行以下操作。否则，转至步骤 4。
   a. 在 Amazon G SageMaker round Truth 控制台中，打开标注任务页面。
   b. 选择您之前创建的作业。这会启用操作菜单。
   c. 从操作菜单中，选择链接。此时将打开作业详细信息页面。
   d. 在任务类型中，选择语义分割。
   e. 选择下一步。
   f. 在语义分割标注工具部分，为您希望模型发现的每种异常类型添加异常标签。
   g. 选择创建。
   h. 等待工作人员标注您的图像。
4. 打开 Ground Truth 控制台并打开标注作业页面。
5. 如果要创建图像分类模型，请选择您在步骤 1 中创建的作业。如果要创建图像分割模型，请选择您在步骤 3 中创建的作业。
6. 在标注作业摘要中，打开输出数据集位置中的 S3 位置。记下清单文件的位置，应该是 `s3://output-dataset-location/manifests/output/output.manifest`。
7. 如果要为测试数据集创建清单文件，请重复此过程。否则，请按照创建数据集中的说明操作，使用清单文件创建数据集。

创建数据集

通过此过程，可以使用您在 使用 Amazon G SageMaker round Truth 为图片贴标 步骤 6 中记下的清单文件，在 Lookout for Vision 项目中创建一个数据集。清单文件会为单数据集项目创建训练数据集。

如果您希望您的项目拥有单独的测试数据集，则可以运行另一个 Amazon G SageMaker round Truth 任务来为测试数据集创建清单文件。或者，您可以自己创建清单文件。您也可以从 Amazon S3 桶或自己的本地计算机，向测试数据集中导入图像。（可能需要标注这些图像，然后才能训练模型）。

此过程将假设您的项目没有任何数据集。

使用 Lookout for Vision 创建数据集（控制台）

1. 打开 Amazon Lookout for Vision 控制台，网址为 https://console.aws.amazon.com/lookoutvision/。
2. 选择开始。
3. 在左侧导航窗格中，选择项目。
4. 选择要添加到哪一个项目，以便与清单文件一起使用。
5. 在工作原理部分，选择创建数据集。
6. 选择单数据集选项卡或单独的训练数据集和测试数据集选项卡，然后按照步骤进行操作。

   Single dataset

   1. 选择创建单个数据集。
   2. 在图像源配置部分，选择导入由 G SageMaker round Truth 标记的图像。
   3. 对于 .manifest 文件位置，请输入您在 使用 Amazon G SageMaker round Truth 为图片贴标 步骤 6 中记下的清单文件位置。

   Separate training and test datasets

   1. 选择创建训练数据集和测试数据集。
   2. 在训练数据集详细信息部分，选择导入 SageMaker由 Ground Truth 标注的图像。
   3. 在 .manifest 文件位置中，输入您在 使用 Amazon G SageMaker round Truth 为图片贴标 步骤 6 中记下的清单文件位置。
   4. 在测试数据集详细信息部分，选择导入 SageMaker 由 Ground Truth 标注的图像。
   5. 在 .manifest 文件位置中，输入您在 使用 Amazon G SageMaker round Truth 为图片贴标 步骤 6 中记下的清单文件位置。请记住，测试数据集需要单独的清单文件。
7. 选择提交。
8. 按照 训练您的模型 中的步骤，训练您的模型。

创建清单文件

您可以通过导入 G SageMaker round Truth 格式的清单文件来创建数据集。如果您的图像的标签格式不是 G SageMaker round Truth 清单文件，请使用以下信息创建 G SageMaker round Truth 格式的清单文件。

清单文件采用 JSON 行格式，其中每行都是一个完整的 JSON 对象，用于表示图像的标注信息。图像分类和图像分割有不同的格式。必须使用 UTF-8 编码对清单文件进行编码。

> **Note**
>
> 本部分的 JSON 行示例采用了便于阅读的格式。

清单文件引用的图像必须位于同一 Amazon S3 桶中。清单文件可以位于其他桶中。您应在 JSON 行的 `source-ref` 字段中指定图像的位置。

您可以使用代码创建清单文件。Amazon Lookout for Vision Lab Python 笔记本展示了如何为电路板示例图像创建图像分类清单文件。或者，您可以使用代码示例存储库中的数据集示例 AWS 代码。通过使用逗号分隔值（CSV）文件，您可以轻松创建清单文件。有关更多信息，请参阅 通过 CSV 文件创建分类清单文件。

主题

- 为图像分类定义 JSON 行
- 为图像分割定义 JSON 线
- 通过 CSV 文件创建分类清单文件
- 使用清单文件创建数据集（控制台）
- 使用清单文件创建数据集（SDK）

为图像分类定义 JSON 行

在 Amazon Lookout for Vision 清单文件中，您应为要使用的每张图像各定义一个 JSON 行。如果要创建分类模型，则 JSON 行必须包含正常或异常的图像分类。JSON 行采用 G SageMaker round Trut h 分类任务输出格式。清单文件由一个或多个 JSON 行组成，每个行对应一张您要导入的图像。

为已分类的图像创建清单文件

1. 创建空文本文件。

2. 为要导入的每张图像各添加一个 JSON 行。每个 JSON 行应该与下面类似：

```
{"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. 保存该文件。

> **Note**
>
> 您可以使用扩展名 .manifest，但不要求必须如此。

4. 使用您创建的清单文件，创建一个数据集。有关更多信息，请参阅 创建清单文件。

分类 JSON 行

在本部分，您应了解如何创建用于将图像归类为正常或异常的 JSON 行。

异常 JSON 行

以下 JSON 行显示了一张标注为异常的图像。请注意，class-name 的值为 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
}
```

正常 JSON 行

以下 JSON 行显示了一张标注为正常的图像。请注意，class-name 的值为 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 行键和值

以下信息描述了 Amazon Lookout for Vision JSON 行中的键和值。

source-ref

（必需）图像的 Amazon S3 位置。格式为 "s3://*BUCKET*/*OBJECT_PATH*"。所导入数据集中的图像必须存储在同一 Amazon S3 桶中。

anomaly-label

（必需）标签属性。应使用键 anomaly-label，或您选择的其他键名称。Amazon Lookout for Vision 需要该键值（前一示例中 0），但未使用该值。Amazon Lookout for Vision 创建的输出清单将该值转换为 1，表示异常图像，0 值表示正常图像。class-name 的值决定了图像正常还是异常。

必须有相应的元数据，这些元数据通过字段名称进行标识，并附有 -metadata。例如，"anomaly-label-metadata"。

anomaly-label-metadata

（必需）与标签属性相关的元数据。该字段名称必须与标签属性附加 -metadata 之后相同。

confidence

（可选）Amazon Lookout for Vision 当前未使用。如果您确实要指定一个值，请使用值 1。

job-name

（可选）您为用于处理图像的作业选择的名称。

class-name

（必需）如果图像包含正常内容，请指定 normal，否则请指定 anomaly。如果 class-name 的值是任何其他值，则图像将作为未标注图像添加到数据集中。要标注图像，请参阅 向您的数据集中添加图像。

human-annotated

（必需）如果注释由人工完成，请指定 "yes"。否则，请指定 "no"。

creation-date

（可选）创建标签的协调世界时（UTC）日期和时间。

类型

（必需）应该应用于图像的处理类型。对于图像级异常标签，该值为 "groundtruth/image-classification"。

为图像分割定义 JSON 线

在 Amazon Lookout for Vision 清单文件中，您应为要使用的每张图像各定义一个 JSON 行。如果要创建分割模型，则 JSON 行必须包含图像的分割和分类信息。清单文件由一个或多个 JSON 行组成，您要导入的每张图像各有一行。

为已分割的图像创建清单文件

1. 创建空文本文件。
2. 为要导入的每张图像各添加一个 JSON 行。每个 JSON 行应该与下面类似：

```
{"source-ref":"s3://path-to-image","anomaly-label":1,"anomaly-label-metadata":
{"class-name":"anomaly","creation-date":"2021-10-12T14:16:45.668","human-
annotated":"yes","job-name":"labeling-job/classification-job","type":"groundtruth/
image-classification","confidence":1},"anomaly-mask-ref":"s3://path-to-
image","anomaly-mask-ref-metadata":{"internal-color-map":{"0":{"class-
name":"BACKGROUND","hex-color":"#ffffff","confidence":0.0},"1":{"class-
name":"scratch","hex-color":"#2ca02c","confidence":0.0},"2":{"class-
name":"dent","hex-color":"#1f77b4","confidence":0.0}},"type":"groundtruth/
semantic-segmentation","human-annotated":"yes","creation-
date":"2021-11-23T20:31:57.758889","job-name":"labeling-job/segmentation-job"}}
```

3. 保存该文件。

> **Note**
>
> 您可以使用扩展名 .manifest，但不要求必须如此。

4. 使用您创建的清单文件，创建一个数据集。有关更多信息，请参阅 创建清单文件。

分割 JSON 行

在本部分中，您将了解如何创建包含图像分割和分类信息的 JSON 行。

下面的 JSON 行显示了一张包含分割和分类信息的图像。anomaly-label-metadata 包含分类信息。anomaly-mask-ref 和 anomaly-mask-ref-metadata 包含分割信息。

```
{
    "source-ref": "s3://path-to-image",
    "anomaly-label": 1,
    "anomaly-label-metadata": {
        "class-name": "anomaly",
        "creation-date": "2021-10-12T14:16:45.668",
        "human-annotated": "yes",
        "job-name": "labeling-job/classification-job",
        "type": "groundtruth/image-classification",
        "confidence": 1
    },
    "anomaly-mask-ref": "s3://path-to-image",
    "anomaly-mask-ref-metadata": {
        "internal-color-map": {
            "0": {
                "class-name": "BACKGROUND",
                "hex-color": "#ffffff",
                "confidence": 0.0
            },
            "1": {
                "class-name": "scratch",
                "hex-color": "#2ca02c",
                "confidence": 0.0
            },
            "2": {
                "class-name": "dent",
                "hex-color": "#1f77b4",
```

```
                "confidence": 0.0
            }
        },
        "type": "groundtruth/semantic-segmentation",
        "human-annotated": "yes",
        "creation-date": "2021-11-23T20:31:57.758889",
        "job-name": "labeling-job/segmentation-job"
    }
}
```

JSON 行键和值

以下信息描述了 Amazon Lookout for Vision JSON 行中的键和值。

source-ref

（必需）图像的 Amazon S3 位置。格式为 "s3://*BUCKET*/*OBJECT_PATH*"。所导入数据集中的图像必须存储在同一 Amazon S3 桶中。

anomaly-label

（必需）标签属性。应使用键 anomaly-label，或您选择的其他键名称。Amazon Lookout for Vision 需要该键值（前一示例中 1），但未使用该值。Amazon Lookout for Vision 创建的输出清单将该值转换为 1，表示异常图像，0 值表示正常图像。class-name 的值决定了图像正常还是异常。

必须有相应的元数据，这些元数据通过字段名称进行标识，并附有 -metadata。例如，"anomaly-label-metadata"。

anomaly-label-metadata

（必需）与标签属性相关的元数据。包含分类信息。字段名称必须与标签属性附加 -metadata 之后相同。

confidence

（可选）Amazon Lookout for Vision 当前未使用。如果您确实要指定一个值，请使用值 1。

job-name

（可选）您为用于处理图像的作业选择的名称。

class-name

（必需）如果图像包含正常内容，请指定 normal，否则请指定 anomaly。如果 class-name 的值是任何其他值，则图像将作为未标注图像添加到数据集中。要标注图像，请参阅 向您的数据集中添加图像。

human-annotated

（必需）如果注释由人工完成，请指定 "yes"。否则，请指定 "no"。

creation-date

（可选）创建标签的协调世界时（UTC）日期和时间。

类型

（必需）应该应用于图像的处理类型。应使用值 "groundtruth/image-classification"。

anomaly-mask-ref

（必需）掩码图像的 Amazon S3 位置。使用 anomaly-mask-ref 作为键名称，或者使用您选择的键名称。键必须以 -ref 结尾。掩码图像必须包含每种异常类型的有色掩码 internal-color-map。格式为 "s3://*BUCKET*/*OBJECT_PATH*"。所导入数据集中的图像必须存储在同一 Amazon S3 桶中。掩码图像必须是便携式网络图形（PNG）格式的图像。

anomaly-mask-ref-metadata

（必需）图像的分割元数据。使用 anomaly-mask-ref-metadata 作为键名称，或者使用您选择的键名称。键名称必须以 -ref-metadata 结尾。

internal-color-map

（必需）与各种异常类型对应的颜色图。颜色必须与掩码图像中的颜色匹配（anomaly-mask-ref）。

密钥

（必需）颜色图中的键。条目 0 必须包含类名 BACKGROUND，表示图像上异常之外的区域。

class-name

（必需）异常类型的名称，如划痕或凹痕。

hex-color

（必需）异常类型对应的十六进制颜色，如 #2ca02c。颜色必须与 anomaly-mask-ref 中的颜色匹配。BACKGROUND 异常类型的值始终为 #ffffff。

confidence

（必需）Amazon Lookout for Vision 当前未使用，但需要一个浮点值。

human-annotated

（必需）如果注释由人工完成，请指定 "yes"。否则，请指定 "no"。

creation-date

（可选）创建分割信息的协调世界时（UTC）日期和时间。

类型

（必需）应该应用于图像的处理类型。应使用值 "groundtruth/semantic-segmentation"。

通过 CSV 文件创建分类清单文件

此示例 Python 脚本使用逗号分隔值（CSV）文件来标注图像，从而简化了分类清单文件的创建工作。您需要创建 CSV 文件。

清单文件描述了用于训练模型的图像。清单文件由一个或多个 JSON 行组成。每个 JSON 行都描述了一张图像。有关更多信息，请参阅 为图像分类定义 JSON 行。

CSV 文件代表文本文件中多行的表格数据。一行中的各个字段用逗号分隔。有关更多信息，请参阅逗号分隔的值。对于此脚本，CSV 文件中的每一行都包括图像的 S3 位置和图像的异常分类（normal 或 anomaly）。每一行分别对应清单文件中的一个 JSON 行。

例如，以下 CSV 文件描述了示例图像中的一些图像。

```
s3://s3bucket/circuitboard/train/anomaly/train-anomaly_1.jpg,anomaly
s3://s3bucket/circuitboard/train/anomaly/train-anomaly_10.jpg,anomaly
s3://s3bucket/circuitboard/train/anomaly/train-anomaly_11.jpg,anomaly
s3://s3bucket/circuitboard/train/normal/train-normal_1.jpg,normal
s3://s3bucket/circuitboard/train/normal/train-normal_10.jpg,normal
s3://s3bucket/circuitboard/train/normal/train-normal_11.jpg,normal
```

该脚本会为每一行生成 JSON 行。例如，以下是第一行（s3://s3bucket/circuitboard/train/anomaly/train-anomaly_1.jpg,anomaly）的 JSON 行。

```
{"source-ref": "s3://s3bucket/csv_test/train_anomaly_1.jpg","anomaly-label":
 1,"anomaly-label-metadata": {"confidence": 1,"job-name": "labeling-job/anomaly-
classification","class-name": "anomaly","human-annotated": "yes","creation-date":
 "2022-02-04T22:47:07","type": "groundtruth/image-classification"}}
```

如果您的 CSV 文件不包含图像的 Amazon S3 路径，请使用 `--s3-path` 命令行参数指定图像的 Amazon S3 路径。

在创建清单文件之前，该脚本会检查 CSV 文件中是否有重复图像以及任何不是 `normal` 或 `anomaly` 的图像分类。如果发现重复图像或图像分类错误，则该脚本会执行以下操作：

- 在去重 CSV 文件中，记录所有图像的第一个有效图像条目。
- 在错误文件中，记录图像的重复版本。
- 在错误文件中，记录不是 `normal` 或 `anomaly` 的图像分类。
- 不创建清单文件。

错误文件中包含在输入 CSV 文件中发现重复图像或分类错误的行号。请使用错误 CSV 文件更新输入 CSV 文件，然后再次运行该脚本。或者，使用错误 CSV 文件更新去重 CSV 文件，后者中仅包含唯一图像条目和没有图像分类错误的图像。使用更新后的去重 CSV 文件，重新运行该脚本。

如果在输入 CSV 文件中未发现重复项或错误，则该脚本会删除去重图像 CSV 文件和错误文件，因为它们为空。

在此过程中，您将创建 CSV 文件并运行 Python 脚本以创建清单文件。此脚本已使用 Python 版本 3.7 进行测试。

通过 CSV 文件创建清单文件

1. 创建一个 CSV 文件，并且在每一行中包含以下字段（每张图像占一行）。请勿在 CSV 文件中添加标题行。

字段 1	字段 2
图像名称或图像的 Amazon S3 路径。例如，`s3://s3bucket/circuitboard/train/anomaly/train-anomaly_10.jpg` 。您不能混合使用带有 Amazon	图像的异常分类（`normal` 或 `anomaly`）。

字段 1	字段 2
S3 路径的图像和不带 Amazon S3 路径的图像。	

例如，s3://s3bucket/circuitboard/train/anomaly/image_10.jpg,anomaly 或 image_11.jpg,normal

2. 保存 CSV 文件。
3. 运行以下 Python 脚本。提供以下参数：

   - csv_file：您在步骤 1 中创建的 CSV 文件。
   - （可选）--s3-path *s3://path_to_folder/*：要添加到图像文件名的 Amazon S3 路径（字段 1）。如果字段 1 中的图像未包含 S3 路径，则使用 --s3-path。

```
# Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
# SPDX-License-Identifier:  Apache-2.0
"""
Purpose
Shows how to create an Amazon Lookout for Vision manifest file from a CSV file.
The CSV file format is image location,anomaly classification (normal or anomaly)
For example:
s3://s3bucket/circuitboard/train/anomaly/train_11.jpg,anomaly
s3://s3bucket/circuitboard/train/normal/train_1.jpg,normal

If necessary, use the bucket argument to specify the Amazon S3 bucket folder for
 the images.
"""

from datetime import datetime, timezone
import argparse
import logging
import csv
import os
import json

logger = logging.getLogger(__name__)


def check_errors(csv_file):
```

```
    """
    Checks for duplicate images and incorrect classifications in a CSV file.
    If duplicate images or invalid anomaly assignments are found, an errors CSV
file
    and deduplicated CSV file are created. Only the first
    occurrence of a duplicate is recorded. Other duplicates are recorded in the
errors file.
    :param csv_file: The source CSV file
    :return: True if errors or duplicates are found, otherwise false.
    """

    logger.info("Checking %s.", csv_file)

    errors_found = False
    errors_file = f"{os.path.splitext(csv_file)[0]}_errors.csv"
    deduplicated_file = f"{os.path.splitext(csv_file)[0]}_deduplicated.csv"

    with open(csv_file, 'r', encoding="UTF-8") as input_file,\
            open(deduplicated_file, 'w', encoding="UTF-8") as dedup,\
            open(errors_file, 'w', encoding="UTF-8") as errors:

        reader = csv.reader(input_file, delimiter=',')
        dedup_writer = csv.writer(dedup)
        error_writer = csv.writer(errors)
        line = 1
        entries = set()
        for row in reader:

            # Skip empty lines.
            if not ''.join(row).strip():
                continue

            # Record any incorrect classifications.
            if not row[1].lower() == "normal" and not row[1].lower() == "anomaly":
                error_writer.writerow(
                    [line, row[0], row[1], "INVALID_CLASSIFICATION"])
                errors_found = True

            # Write first image entry to dedup file and record duplicates.
            key = row[0]
            if key not in entries:
                dedup_writer.writerow(row)
                entries.add(key)
            else:
```

```
                error_writer.writerow([line, row[0], row[1], "DUPLICATE"])
                errors_found = True
            line += 1

    if errors_found:
        logger.info("Errors found check %s.", errors_file)
    else:
        os.remove(errors_file)
        os.remove(deduplicated_file)

    return errors_found


def create_manifest_file(csv_file, manifest_file, s3_path):
    """
    Read a CSV file and create an Amazon Lookout for Vision classification manifest
 file.
    :param csv_file: The source CSV file.
    :param manifest_file: The name of the manifest file to create.
    :param s3_path: The Amazon S3 path to the folder that contains the images.
    """
    logger.info("Processing CSV file %s.", csv_file)

    image_count = 0
    anomalous_count = 0

    with open(csv_file, newline='', encoding="UTF-8") as csvfile,\
        open(manifest_file, "w", encoding="UTF-8") as output_file:

        image_classifications = csv.reader(
            csvfile, delimiter=',', quotechar='|')

        # Process each row (image) in the CSV file.
        for row in image_classifications:
            # Skip empty lines.
            if not ''.join(row).strip():
                continue

            source_ref = str(s3_path) + row[0]
            classification = 0

            if row[1].lower() == 'anomaly':
                classification = 1
                anomalous_count += 1
```

```
            # Create the JSON line.
             json_line = {}
             json_line['source-ref'] = source_ref
             json_line['anomaly-label'] = str(classification)

             metadata = {}
             metadata['confidence'] = 1
             metadata['job-name'] = "labeling-job/anomaly-classification"
             metadata['class-name'] = row[1]
             metadata['human-annotated'] = "yes"
             metadata['creation-date'] = datetime.now(timezone.utc).strftime('%Y-%m-
%dT%H:%M:%S.%f')
             metadata['type'] = "groundtruth/image-classification"

             json_line['anomaly-label-metadata'] = metadata

             output_file.write(json.dumps(json_line))
             output_file.write('\n')
             image_count += 1

    logger.info("Finished creating manifest file %s.\n"
                "Images: %s\nAnomalous: %s",
                manifest_file,
                image_count,
                anomalous_count)
    return image_count, anomalous_count


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

    parser.add_argument(
        "csv_file", help="The CSV file that you want to process."
    )

    parser.add_argument(
        "--s3_path", help="The Amazon S3 bucket and folder path for the images."
        " If not supplied, column 1 is assumed to include the Amazon S3 path.",
 required=False
    )
```

```
def main():

    logging.basicConfig(level=logging.INFO,
                        format="%(levelname)s: %(message)s")

    try:

        # Get command line arguments.
        parser = argparse.ArgumentParser(usage=argparse.SUPPRESS)
        add_arguments(parser)
        args = parser.parse_args()
        s3_path = args.s3_path
        if s3_path is None:
            s3_path = ""

        csv_file = args.csv_file
        csv_file_no_extension = os.path.splitext(csv_file)[0]
        manifest_file = csv_file_no_extension + '.manifest'

        # Create manifest file if there are no duplicate images.
        if check_errors(csv_file):
            print(f"Issues found. Use {csv_file_no_extension}_errors.csv "\
                "to view duplicates and errors.")
            print(f"{csv_file}_deduplicated.csv contains the first"\
                "occurrence of a duplicate.\n"
                  "Update as necessary with the correct information.")
            print(f"Re-run the script with
 {csv_file_no_extension}_deduplicated.csv")
        else:
            print('No duplicates found. Creating manifest file.')

            image_count, anomalous_count = create_manifest_file(csv_file,
 manifest_file, s3_path)

            print(f"Finished creating manifest file: {manifest_file} \n")

            normal_count = image_count-anomalous_count
            print(f"Images processed: {image_count}")
            print(f"Normal: {normal_count}")
            print(f"Anomalous: {anomalous_count}")

    except FileNotFoundError as err:
```

```
        logger.exception("File not found.:%s", err)
        print(f"File not found: {err}. Check your input CSV file.")

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

4. 如果出现重复的图像或出现分类错误：

   a. 使用错误文件更新去重 CSV 文件或输入 CSV 文件。

   b. 使用更新后的去重 CSV 文件或更新后的输入 CSV 文件再次运行该脚本。

5. 如果您计划使用测试数据集，请重复步骤 1-4，以便为测试数据集创建清单文件。

6. 如有必要，请从您的计算机将图像复制到您在 CSV 文件第 1 列中指定的（或在 `--s3-path` 命令行中指定的）Amazon S3 桶路径。要复制图像，请在命令提示符处输入以下命令。

```
aws s3 cp --recursive your-local-folder/ s3://your-target-S3-location/
```

7. 按照 使用清单文件创建数据集（控制台） 部分的说明操作，创建一个数据集。如果您使用的是 AWS SDK，请参阅使用清单文件创建数据集（SDK）。

使用清单文件创建数据集（控制台）

以下过程向您展示如何通过导入存储在 Amazon S3 存储桶中的 SageMaker 格式清单文件来创建训练或测试数据集。

创建数据集后，您可以向数据集中添加更多图像，或者为图像添加标签。有关更多信息，请参阅 向您的数据集中添加图像。

使用 G SageMaker round Truth 格式的清单文件创建数据集（控制台）

1. 创建或使用现有的、 SageMaker 兼容 Amazon Lookout for Vision 的 Ground Truth 格式清单文件。有关更多信息，请参阅 创建清单文件。
2. 登录 AWS Management Console 并打开亚马逊 S3 控制台，网址为 https://console.aws.amazon.com/s3/。
3. 在 Amazon S3 桶中，创建一个文件夹以用于存放您的清单文件。
4. 上传您的清单文件至您刚才创建的文件夹。
5. 在 Amazon S3 桶中，创建一个文件夹来存储您的图像。
6. 上传您的图像至您刚才创建的文件夹。

> **Important**
>
> 每个 JSON 行中的 `source-ref` 字段值必须对应文件夹中的图像。

7. 打开 Amazon Lookout for Vision 控制台，网址为 https://console.aws.amazon.com/lookoutvision/。
8. 选择开始。
9. 在左侧导航窗格中，选择项目。
10. 选择要添加到哪一个项目，以便与清单文件一起使用。
11. 在工作原理部分，选择创建数据集。
12. 选择单数据集选项卡或单独的训练数据集和测试数据集选项卡，然后按照步骤进行操作。

    Single dataset

    1. 选择创建单个数据集。
    2. 在图像源配置部分，选择导入由 G SageMaker round Truth 标记的图像。
    3. 对于 .manifest 文件位置，请输入您的清单文件位置。

    Separate training and test datasets

    1. 选择创建训练数据集和测试数据集。
    2. 在训练数据集详细信息部分，选择导入 SageMaker由 Ground Truth 标注的图像。
    3. 在 .manifest 文件位置中，输入您的训练清单文件位置。
    4. 在测试数据集详细信息部分，选择导入 SageMaker 由 Ground Truth 标注的图像。
    5. 在 .manifest 文件位置中，输入您的测试清单文件位置。

    > **Note**
    >
    > 您的训练数据集和测试数据集可以有不同的图像源。

13. 选择提交。
14. 按照 训练您的模型 中的步骤，训练您的模型。

Amazon Lookout for Vision 会在 Amazon S3 桶 `datasets` 文件夹中创建一个数据集。您的原始 `.manifest` 文件保持不变。

使用清单文件创建数据集（SDK）

您可以使用该CreateDataset操作来创建与 Amazon Lookout for Vision 项目相关的数据集。

如果要使用单个数据集进行训练和测试，请将 `DatasetType` 值设置为 `train` 以创建单个数据集。在训练过程中，数据集会在内部进行拆分，形成训练数据集和测试数据集。您无权访问拆分后的训练及测试数据集。如果要使用单独的测试数据集，请再次调用 `CreateDataset`，并将 `DatasetType` 值设置为 `test`。在训练期间，将使用训练数据集和测试数据集来训练和测试模型。

您可以选择使用`DatasetSource`参数来指定用于填充数据集的 G SageMaker round Truth 格式清单文件的位置。在这种情况下，对 `CreateDataset` 的调用是异步的。要检查当前状态，请调用 `DescribeDataset`。有关更多信息，请参阅 查看您的数据集。如果在导入过程中出现验证错误，则 `Status` 的值会被设置为 CREATE_FAILED，并且设置状态消息（`StatusMessage`）。

> **Tip**
>
> 如果要创建包含开始使用示例数据集的数据集，请使用脚本在步骤 1：创建清单文件并上传图像中创建的清单文件（`getting-started/dataset-files/manifests/train.manifest`）。
> 如果要创建包含电路板示例图像的数据集，您有两种选择：
>
> 1. 使用代码创建清单文件。Amazon Lookout for Vision Lab Python 笔记本展示了如何为电路板示例图像创建清单文件。或者，使用代码示例存储库中的数据集示例 AWS 代码。
> 2. 如果您已经使用 Amazon Lookout for Vision 控制台创建了包含电路板示例图像的数据集，请重复使用 Amazon Lookout for Vision 为您创建的清单文件。训练和测试清单文件的位置为 `s3://bucket/datasets/project name/train or test/manifests/output/output.manifest`。

如果未指定 `DatasetSource`，则会创建一个空数据集。在这种情况下，对 `CreateDataset` 的调用是同步的。稍后，您可以通过调用 UpdateDataset Entries 将图像标记到数据集中。有关代码示例，请参阅 添加更多图像 (SDK)。

如果要替换数据集，请先使用删除现有数据集，DeleteDataset然后通过调用创建相同数据集类型的新数据集`CreateDataset`。有关更多信息，请参阅 删除数据集。

创建数据集后，即可创建模型。有关更多信息，请参阅 训练模型（SDK）。

您可以通过调用 Entries 来查看数据集中标注的图像（JSON 行 ListDataset ）。您可以通过调用 UpdateDatasetEntries 来添加已标注的图像。

要查看与测试数据集和训练数据集相关的信息，请参阅 查看您的数据集。

创建数据集（SDK）

1. 如果您尚未这样做，请安装和配置和 AWS SDK。 AWS CLI 有关更多信息，请参阅 步骤 4：设置 AWS CLI 和 AWS 软件开发工具包。
2. 使用以下示例代码创建一个数据集。

   CLI

   更改以下值：

   - project-name 更改为要与数据集相关联的项目的名称。
   - dataset-type 更改为要创建的数据集类型（train 或 test ）。
   - dataset-source 更改为清单文件的 Amazon S3 位置。
   - Bucket 更改为包含清单文件的 Amazon S3 桶的名称。
   - Key 更改为包含 Amazon S3 桶中清单文件的路径和文件名。

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

   Python

   此代码取自 AWS 文档 SDK 示例 GitHub 存储库。请在此处查看完整示例。

   ```
       @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
```

Java V2

此代码取自 AWS 文档 SDK 示例 GitHub 存储库。请在此处查看完整示例。

```
/**
 * Creates an Amazon Lookout for Vision dataset from a manifest file.
 * Returns after Lookout for Vision creates the dataset.
 *
 * @param lfvClient    An Amazon Lookout for Vision client.
 * @param projectName  The name of the project in which you want to create a
 *                     dataset.
 * @param datasetType  The type of dataset that you want to create (train or
 *                     test).
 * @param bucket       The S3 bucket that contains the manifest file.
 * @param manifestFile The name and location of the manifest file within the S3
 *                     bucket.
 * @return DatasetDescription The description of the created dataset.
 */
public static DatasetDescription createDataset(LookoutVisionClient lfvClient,
                String projectName,
                String datasetType,
                String bucket,
                String manifestFile)
                throws LookoutVisionException, InterruptedException {

        logger.log(Level.INFO, "Creating {0} dataset for project {1}",
```

```
                        new Object[] { projectName, datasetType });

        // Build the request. If no bucket supplied, setup for empty dataset
creation.
        CreateDatasetRequest createDatasetRequest = null;

        if (bucket != null && manifestFile != null) {

                InputS3Object s3Object = InputS3Object.builder()
                                .bucket(bucket)
                                .key(manifestFile)
                                .build();

                DatasetGroundTruthManifest groundTruthManifest =
DatasetGroundTruthManifest.builder()
                                .s3Object(s3Object)
                                .build();

                DatasetSource datasetSource = DatasetSource.builder()
                                .groundTruthManifest(groundTruthManifest)
                                .build();

                createDatasetRequest = CreateDatasetRequest.builder()
                                .projectName(projectName)
                                .datasetType(datasetType)
                                .datasetSource(datasetSource)
                                .build();
        } else {
                createDatasetRequest = CreateDatasetRequest.builder()
                                .projectName(projectName)
                                .datasetType(datasetType)
                                .build();
        }

        lfvClient.createDataset(createDatasetRequest);

        DatasetDescription datasetDescription = null;

        boolean finished = false;

        // Wait until dataset is created, or failure occurs.
        while (!finished) {
```

```
                datasetDescription = describeDataset(lfvClient, projectName,
datasetType);

                switch (datasetDescription.status()) {
                        case CREATE_COMPLETE:
                                logger.log(Level.INFO, "{0}dataset created for
project {1}",
                                                new Object[] { datasetType,
projectName });
                                finished = true;
                                break;
                        case CREATE_IN_PROGRESS:
                                logger.log(Level.INFO, "{0} dataset creating for
project {1}",
                                                new Object[] { datasetType,
projectName });

                                TimeUnit.SECONDS.sleep(5);

                                break;

                        case CREATE_FAILED:
                                logger.log(Level.SEVERE,
                                                "{0} dataset creation failed for
project {1}. Error {2}",
                                                new Object[] { datasetType,
projectName,

datasetDescription.statusAsString() });
                                finished = true;
                                break;
                        default:
                                logger.log(Level.SEVERE, "{0} error when
creating {1} dataset for project {2}",
                                                new Object[] { datasetType,
projectName,

datasetDescription.statusAsString() });
                                finished = true;
                                break;

                }
        }
```

```
        logger.log(Level.INFO, "Dataset info. Status: {0}\n Message: {1} }",
                        new Object[] { datasetDescription.statusAsString(),
                                       datasetDescription.statusMessage() });

        return datasetDescription;

}
```

3. 按照 训练模型（SDK） 部分的步骤训练您的模型。

标注图像

您可以使用 Amazon Lookout for Vision 控制台，添加或修改为数据集中图像分配的标签。如果您使用的是 SDK，则标签是您提供的清单文件的一部分CreateDataset。您可以通过调用 Entries 来更新图像的UpdateDataset标签。有关代码示例，请参阅 添加更多图像 (SDK)。

选择模型类型

您为图像分配的标签决定了 Lookout for Vision 会创建什么类型的模型。如果您的项目有单独的测试数据集，请用相同的方式标注图像。

图像分类模型

要创建图像分类模型，您需要将每张图像归类为正常或异常。对于每张图像，请执行以下操作：图像分类（控制台）。

图像分割模型

要创建图像分割模型，您需要将每张图像归类为正常或异常。对于每张异常图像，您还应为图像上的每个异常区域指定一个像素掩码，并为像素掩码中的异常类型指定一个异常标签。例如，下图中的蓝色掩码用于标记汽车上的划痕异常类型的位置。您可以在一张图像中指定多种异常标签类型。对于每张图像，请执行以下操作：分割图像（控制台）。

图像分类（控制台）

您应使用 Lookout for Vision 控制台，将数据集中的图像归类为正常或异常。未分类的图像不会用于训练您的模型。

如果您要创建图像分割模型，请跳过此过程并执行以下操作：分割图像（控制台），其中包括图像分类步骤。

> **Note**
>
> 如果您刚完成 创建您的数据集，则控制台此时应该会显示您的模型控制面板，并且您无需执行步骤 1-4。

对您的图像进行分类（控制台）

1. 打开 Amazon Lookout for Vision 控制台，网址为 https://console.aws.amazon.com/lookoutvision/。
2. 在左侧导航窗格中，选择项目。
3. 在项目页面上，选择要使用的项目。
4. 在项目的左侧导航窗格中，选择数据集。
5. 如果您有单独的训练数据集和测试数据集，请选择要使用的数据集对应的选项卡。
6. 选择开始标记。
7. 选中选择此页面上的所有图像。
8. 如果图像正常，请选择分类为正常，否则选择分类为异常。每张照片下方都会出现一个标签。
9. 如果需要更改图像的标签，请执行以下操作：

    a. 在图像下方选择异常或正常。

b. 如果无法确定图像的正确标签，请选择图库中的图像以放大图像。

> **Note**
>
> 在筛选条件部分，您可以选择所需的标签或标签状态以筛选图像标签。

10. 必要时在每页上重复步骤 7-9，直到数据集中的所有图像都已正确进行标注。
11. 选择保存更改。
12. 完成图像标注后，您便可以训练自己的模型。

分割图像（控制台）

如果要创建图像分割模型，您必须将图像归类为正常图像或异常图像。您还必须向异常图像添加分割信息。要指定分割信息，首先要为您希望模型发现的每种异常类型（如凹痕或划痕）指定异常标签。然后，您应为数据集中异常图像上的每个异常，分别指定异常掩码和异常标签。

> **Note**
>
> 如果要创建图像分类模型，则无需分割图像，也不需要指定异常标签。

主题

- 指定异常标签
- 标注图像
- 使用注释工具分割图像

指定异常标签

您应为数据集图像中的每种异常类型，分别定义一个异常标签。

指定异常标签

1. 打开 Amazon Lookout for Vision 控制台，网址为 https://console.aws.amazon.com/lookoutvision/。
2. 在左侧导航窗格中，选择项目。
3. 在项目页面上，选择要使用的项目。

4. 在项目的左侧导航窗格中，选择数据集。
5. 在异常标签中，选择添加异常标签。如果您之前添加过异常标签，请选择管理。
6. 在对话框中，执行以下操作：
   a. 输入要添加的异常标签，然后选择添加异常标签。
   b. 重复上一个步骤，直到已输入您希望模型发现的每个异常标签。
   c. （可选）选择编辑图标以更改标签名称。
   d. （可选）选择删除图标以删除新的异常标签。您无法删除数据集当前正在使用的异常类型。
7. 选择确认，将新的异常标签添加到数据集中。

指定异常标签后，通过执行 标注图像 来标注图像。

标注图像

要标注图像以进行图像分割，请将该图像归类为正常或异常。然后，使用注释工具，通过绘制掩码，紧密覆盖图像中存在的每种异常类型的区域，从而对图像进行分割。

标注图像

1. 如果您有单独的训练数据集和测试数据集，请选择要使用的数据集对应的选项卡。
2. 通过执行以下操作：指定异常标签，为数据集指定异常类型（如果尚未如此）。
3. 选择开始标记。
4. 选中选择此页面上的所有图像。
5. 如果图像正常，请选择分类为正常，否则选择分类为异常。
6. 要更改单张图像的标签，请在图像下方选择正常或异常。

> **Note**
>
> 在筛选条件部分，您可以选择所需的标签或标签状态以筛选图像标签。在排序选项部分，您可以按置信度分数进行排序。

7. 对于每张异常图像，选择该图像以打开注释工具。通过执行以下操作：使用注释工具分割图像，添加分割信息。
8. 选择保存更改。
9. 完成图像标注后，您便可以训练自己的模型。

使用注释工具分割图像

您可以使用注释工具，通过用掩码来标记异常区域以分割图像。

使用注释工具分割图像

1. 通过选择数据集图库中的图像，打开注释工具。如有必要，请选择开始标记以进入标注模式。
2. 在异常标签部分，选择要标记的异常标签。如有必要，请选择添加异常标签以添加新的异常标签。
3. 在页面底部选择绘图工具，然后为异常标签绘制紧密覆盖其异常区域的掩码。下面的图像是掩码紧密覆盖异常区域的示例。

下面的图像是掩码不佳，未紧密覆盖异常区域的示例。

4. 如果还有更多图像需要分割，请选择下一步，然后重复步骤 2 和步骤 3。
5. 选择提交并关闭，完成图像分割。

训练您的模型

在创建数据集后并标注图像后，您便可以训练模型。作为训练过程的一部分，将会使用测试数据集。如果您具有单数据集项目，则作为训练过程的一部分，数据集中的图像会自动拆分为测试数据集和训练数据集。如果您的项目有训练数据集和测试数据集，则它们会分别用来训练和测试数据集。

训练完成后，您可以评估模型的性能并做出必要的改进。有关更多信息，请参阅 改进您的 Amazon Lookout for Vision 模型。

为了训练您的模型，Amazon Lookout for Vision 会复制您的源训练图像和测试图像。默认情况下，所复制的图像会使用亚马逊云科技拥有和管理的密钥进行静态加密。您也可以选择使用自己的 Amazon Key Management Service（KMS）密钥。有关更多信息，请参阅 Amazon Key Management Service 概念。源图像不受影响。

您可以将元数据以标签的形式分配给自己的模型。有关更多信息，请参阅 标记模型。

每次训练模型时，都会创建一个新的模型版本。如果您不再需要某个模型版本，可以将其删除。有关更多信息，请参阅 删除模型。

您需要按照成功训练模型所花费的时间量付费。有关更多信息，请参阅训练时长。

要查看项目中的现有模型，请 查看您的模型。

> **Note**
>
> 如果您刚完成 创建您的数据集 或 向您的数据集中添加图像，则控制台此时应该会显示您的模型控制面板，并且您无需执行步骤 1-4。

主题

- 训练模型（控制台）
- 训练模型（SDK）

训练模型（控制台）

下面的过程演示如何使用控制台训练您的模型。

训练您的模型（控制台）

1. 打开 Amazon Lookout for Vision 控制台，网址为 https://console.aws.amazon.com/lookoutvision/。
2. 在左侧导航窗格中，选择项目。
3. 在项目页面上，选择包含要训练的模型的项目。
4. 在项目详细信息页面上，选择训练模型。如果您有足够多的已标注图像来训练模型，则训练模型按钮可供使用。如果该按钮不可用，请添加更多图像，直到有足够多的已标注图像为止。
5. （可选）如果要使用自己的 Amazon KMS 加密密钥，请执行以下操作：
   a. 在图像数据加密中，选择自定义加密设置(高级)。
   b. 在 encryption.aws_kms_key 中，输入您的密钥的 Amazon 资源名称 (ARN)，或者选择现有的 AWS KMS 密钥。要创建新密钥，请选择创建 AWS IMS 密钥。
6. （可选）如果要向模型添加标签，请执行以下操作：
   a. 在标签部分，选择添加新标签。
   b. 输入以下信息：
      i. 在键中输入键名称。
      ii. 在值中输入键值。

c. 要添加更多标签，请重复步骤 6a 和 6b。

d. （可选）如果要移除标签，请选择要移除的标签旁的移除。如果要移除先前保存的标签，则会在保存所做的更改时将其移除。

7. 选择训练模型。

8. 在是否要训练您的模型？对话框中，选择训练模型。

9. 在模型视图中，您可以看到训练已开始，通过查看模型版本的 Status 列可以检查当前状态。训练模型需要一些时间才能完成。

10. 训练结束后，您可以评估它的性能。有关更多信息，请参阅 改进您的 Amazon Lookout for Vision 模型。

训练模型（SDK）

您可以使用该CreateModel操作开始模型的训练、测试和评估。Amazon Lookout for Vision 使用与项目关联的训练和测试数据集来训练模型。有关更多信息，请参阅 创建项目（SDK）。

每次调用 CreateModel 时，都会创建一个新的模型版本。CreateModel 的响应中包含模型的版本。

您需要为每次成功训练模型付费。使用 ClientToken 输入参数，有助于防止因用户不必要或意外重复进行模型训练而产生费用。ClientToken 是一个幂等输入参数，可确保一组特定的参数仅会完成一次 CreateModel。使用相同的 ClientToken 值重复调用 CreateModel 时，可确保不会重复训练。如果不为 ClientToken 提供值，则您使用的 Amazon SDK 会为您插入一个值。这样在网络错误后，可以防止重试操作启动多个训练作业，但您需要为自己的使用场景提供自己的价值。有关更多信息，请参阅CreateModel。

训练需要一段时间才能完成。要检查当前状态，请调用 DescribeModel 并传递项目名称（在调用 CreateProject 时指定）和模型版本。status 字段用于指示模型训练的当前状态。有关代码示例，请参阅 查看您的模型 (SDK)。

如果训练成功，您便可以评估模型。有关更多信息，请参阅 改进您的 Amazon Lookout for Vision 模型。

要查看您在项目中已创建的模型，请调用 ListModels。有关代码示例，请参阅 查看您的模型。

训练模型（SDK）

1. 如果您尚未这样做，请安装和配置和 AWS SDK。 AWS CLI 有关更多信息，请参阅 步骤 4：设置 AWS CLI 和 AWS 软件开发工具包。

2. 使用以下示例代码训练模型。

CLI

更改以下值：

- project-name 更改为包含要创建的模型的项目的名称。
- output-config 更改为要在其中保存训练结果的位置。替换以下值：
  - output bucket 是 Amazon S3 桶的名称，Amazon Lookout for Vision 将在该桶中保存训练结果。
  - output folder 是要用来保存训练结果的文件夹的名称。
  - Key 是标签键的名称。
  - Value 是与 tag_key 关联的值。

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

Python

此代码取自 AWS 文档 SDK 示例 GitHub 存储库。请在此处查看完整示例。

```
    @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"]
```

Java V2

此代码取自 AWS 文档 SDK 示例 GitHub 存储库。请在此处查看完整示例。

```
/**
 * Creates an Amazon Lookout for Vision model. The function returns after model
 * training completes. Model training can take multiple hours to complete.
 * You are charged for the amount of time it takes to successfully train a
 model.
 * Returns after Lookout for Vision creates the dataset.
 *
 * @param lfvClient   An Amazon Lookout for Vision client.
 * @param projectName The name of the project in which you want to create a
 *                    model.
 * @param description A description for the model.
 * @param bucket      The S3 bucket in which Lookout for Vision stores the
 *                    training results.
 * @param folder      The location of the training results within the S3
 *                    bucket.
 * @return ModelDescription The description of the created model.
 */
public static ModelDescription createModel(LookoutVisionClient lfvClient, String
 projectName,
                String description, String bucket, String folder)
                throws LookoutVisionException, InterruptedException {
```

```
        logger.log(Level.INFO, "Creating model for project: {0}.", new Object[]
{ projectName });

        // Setup input parameters.
        S3Location s3Location = S3Location.builder()
                        .bucket(bucket)
                        .prefix(folder)
                        .build();

        OutputConfig config = OutputConfig.builder()
                        .s3Location(s3Location)
                        .build();

        CreateModelRequest createModelRequest = CreateModelRequest.builder()
                        .projectName(projectName)
                        .description(description)
                        .outputConfig(config)
                        .build();

        // Create and train the model.
        CreateModelResponse response =
lfvClient.createModel(createModelRequest);

        String modelVersion = response.modelMetadata().modelVersion();
        boolean finished = false;
        DescribeModelResponse descriptionResponse = null;

        // Wait until training finishes or fails.

        do {
                DescribeModelRequest describeModelRequest =
DescribeModelRequest.builder()
                                .projectName(projectName)
                                .modelVersion(modelVersion)
                                .build();

                descriptionResponse =
lfvClient.describeModel(describeModelRequest);

                switch (descriptionResponse.modelDescription().status()) {
                        case TRAINED:
                                logger.log(Level.INFO, "Model training completed
for project {0} version {1}.",
```

```
                                                    new Object[] { projectName,
modelVersion });
                                finished = true;
                                break;

                        case TRAINING:
                                logger.log(Level.INFO,
                                                "Model training in progress for
project {0} version {1}.",
                                                new Object[] { projectName,
modelVersion });
                                TimeUnit.SECONDS.sleep(60);

                                break;

                        case TRAINING_FAILED:
                                logger.log(Level.SEVERE,
                                                "Model training failed for for
project {0} version {1}.",
                                                new Object[] { projectName,
modelVersion });
                                finished = true;
                                break;

                        default:
                                logger.log(Level.SEVERE,
                                                "Unexpected error when training
model project {0} version {1}: {2}.",
                                                new Object[] { projectName,
modelVersion,

descriptionResponse.modelDescription()

.status() });
                                finished = true;
                                break;

                }
        } while (!finished);

        return descriptionResponse.modelDescription();

}
```

3. 训练结束后，您可以评估它的性能。有关更多信息，请参阅 改进您的 Amazon Lookout for Vision 模型。

模型训练问题排查

清单文件或训练图像存在的问题可能会导致模型训练失败。在重新训练模型之前，请检查以下潜在问题。

异常标签颜色与掩码图像中异常的颜色不匹配

如果要训练图像分割模型，则清单文件中异常标签的颜色必须与掩码图像中的颜色相匹配。清单文件中图像的 JSON 行包含元数据（`internal-color-map`），用于告知 Amazon Lookout for Vision 哪种颜色与异常标签对应。例如，以下 JSON 行中 `scratch` 异常标签的颜色为 `#2ca02c`。


```
{
    "source-ref": "s3://path-to-image",
    "anomaly-label": 1,
    "anomaly-label-metadata": {
        "class-name": "anomaly",
        "creation-date": "2021-10-12T14:16:45.668",
        "human-annotated": "yes",
        "job-name": "labeling-job/classification-job",
        "type": "groundtruth/image-classification",
        "confidence": 1
    },
    "anomaly-mask-ref": "s3://path-to-image",
    "anomaly-mask-ref-metadata": {
        "internal-color-map": {
            "0": {
                "class-name": "BACKGROUND",
                "hex-color": "#ffffff",
                "confidence": 0.0
            },
            "1": {
                "class-name": "scratch",
                "hex-color": "#2ca02c",
                "confidence": 0.0
            },
            "2": {
                "class-name": "dent",
```

```
                "hex-color": "#1f77b4",
                "confidence": 0.0
            }
        },
        "type": "groundtruth/semantic-segmentation",
        "human-annotated": "yes",
        "creation-date": "2021-11-23T20:31:57.758889",
        "job-name": "labeling-job/segmentation-job"
    }
}
```

如果掩码图像中的颜色与 hex-color 中的值不匹配，则训练会失败，您需要更新清单文件。

更新清单文件中的颜色值

1. 使用文本编辑器，打开用于创建数据集的清单文件。
2. 对于每个 JSON 行（图像），请检查 internal-color-map 字段中的颜色（hex-color）是否与掩码图像中异常标签的颜色相匹配。

   您可以从 *anomaly-mask*-ref 字段中获取掩码图像的位置。将图像下载到您的计算机，然后使用以下代码获取图像中的颜色。

   ```
   from PIL import Image
   img = Image.open('path to local copy of mask file')
   colors = img.convert('RGB').getcolors() #this converts the mode to RGB
   for color in colors:
       print('#%02x%02x%02x' % color[1])
   ```

3. 对于颜色分配不正确的每张图像，更新该图像对应 JSON 行中的 hex-color 字段。
4. 保存更新后的清单文件。
5. 从项目中删除现有数据集。
6. 使用更新后的清单文件，在项目中创建新数据集。
7. 训练模型。

或者，对于步骤 5 和 6，您可以通过调用 Entrities 操作并为要更新的图像提供更新的 JSON 行来更新数据集中的单个图像。UpdateDataset有关代码示例，请参阅 添加更多图像 (SDK)。

掩码图像不是 PNG 格式

如果要训练图像分割模型，则掩码图像必须为 PNG 格式。如果通过清单文件创建数据集，请确保您在 *anomaly-mask*-ref 中引用的掩码图像是 PNG 格式。如果掩码图像不是 PNG 格式，则需要将其转换为 PNG 格式。仅将图像文件的扩展名重命名为 .png 是不够的。

你在 Amazon Lookout for Vision 控制台或使用 Ground Truth 任务创建 SageMaker 的蒙版图像以 PNG 格式创建。您不需要更改这些图像的格式。

更正清单文件中的非 PNG 格式掩码图像

1. 使用文本编辑器，打开用于创建数据集的清单文件。
2. 对于每个 JSON 行（图像），确保 *anomaly-mask*-ref 引用 PNG 格式的图像。有关更多信息，请参阅 创建清单文件。
3. 保存更新后的清单文件。
4. 从项目中删除现有数据集。
5. 使用更新后的清单文件，在项目中创建新数据集。
6. 训练模型。

分割或分类标签不准确或缺失

标签缺失或不准确可能会导致训练失败，或者创建性能不佳的模型。我们建议您标注数据集中的所有图像。如果您不标注所有图像，并且模型训练失败，或者您的模型性能不佳，请添加更多图像。

请检查以下事项：

- 如果要创建分割模型，则掩码必须紧密覆盖数据集图像上的异常。要检查数据集中的掩码，请在项目的数据集图库中查看图像。如有必要，请重绘图像掩码。有关更多信息，请参阅 分割图像（控制台）。

- 确保对数据集图像中的异常图像进行分类。如果要创建图像分割模型，请确保异常图像具有异常标签和图像掩码。

  记住要创建什么类型的模型（分割或分类）非常重要。分类模型在异常图像上不需要图像掩码。请勿向计划用于分类模型的数据集图像添加掩码。

更新缺失的标签

1. 打开项目的数据集图库。
2. 筛选未标注的图像以查看哪些图像没有标签。
3. 请执行以下操作之一：
   - 如果要创建图像分类模型，请对每张未标注的图像进行分类。
   - 如果要创建图像分割模型，请对每张未标注的图像进行分类和分割。
4. 如果要创建图像分割模型，请为所有缺少掩码的已分类异常图像添加掩码。
5. 训练模型。

如果您选择不修复质量不佳或缺失的标签，我们建议您添加更多已标注的图像，或者从数据集中移除受影响的图像。您可以通过控制台或使用UpdateDataset条目操作添加更多内容。有关更多信息，请参阅向您的数据集中添加图像。

如果您选择移除图像，则必须重新创建不含受影响图像的数据集，因为您无法从数据集中删除图像。有关更多信息，请参阅 从数据集中移除图像。

改进您的 Amazon Lookout for Vision 模型

在训练期间，Lookout for Vision 会使用测试数据集测试您的模型，并利用结果创建性能指标。您可以使用性能指标来评估自己模型的性能。如有必要，您可以采取措施改进数据集，然后重新训练模型。

如果您对模型性能感到满意，就可以开始使用模型。有关更多信息，请参阅 运行经过训练的 Amazon Lookout for Vision 模型。

主题

- 步骤 1：评估模型的性能
- 步骤 2：改进您的模型
- 查看性能指标
- 通过试用检测任务验证您的模型

步骤 1：评估模型的性能

您可以通过控制台和 DescribeModel 操作来访问性能指标。Amazon Lookout for Vision 提供了测试数据集的概要性能指标，以及所有个体图像的预测结果。如果您的模型是分割模型，则控制台还会显示每个异常标签的概要指标。

要在控制台中查看性能指标和测试图像预测，请参阅查看性能指标（控制台）。有关通过 `DescribeModel` 操作访问性能指标和测试图像预测的信息，请参阅查看性能指标 (SDK)。

图像分类指标

Amazon Lookout for Vision 为模型在测试期间所做的分类提供了以下概要指标：

- 精度
- 调用
- F1 分数

图像分割模型指标

如果模型是图像分割模型，Amazon Lookout for Vision 会提供概要图像分类指标和每个异常标签的概要性能指标：

- F1 分数
- 平均交并比 (IoU)

精度

精度指标用于解答以下问题：当模型预测图像包含异常时，该预测的正确率是多少？

在假阳性成本很高的情况下，精度是一个有用的指标。例如，从组装好的机器上拆除没有缺陷的机器部件所造成的成本。

Amazon Lookout for Vision 为整个测试数据集提供了一种概要精度指标值。

精度 是预测正确的异常（真阳性）占所有预测异常（真阳性和假阳性）的比例。精度的公式如下所示。

精度值 = 真阳性数/（真阳性数 + 假阳性数）

可能的精度值介于 0-1 之间。Amazon Lookout for Vision 控制台以百分比值 (0-100) 的形式显示精度。

精度值越高，表示正确预测的异常越多。例如，假设您的模型预测 100 张图像异常。如果有 85 个预测正确（真阳性），15 个预测不正确（假阳性），则精度计算方法如下：

85 个真阳性 /（85 个真阳性 + 15 个假阳性）= 0.85 精度值

但是，如果模型在所做的 100 个异常预测中，仅正确预测了 40 张图像，则最终精度值为较低的 0.40（即 40 / (40 + 60) = 0.40）。在这种情况下，您的模型做出的错误预测多于正确预测。要解决这个问题，可以考虑对模型进行改进。有关更多信息，请参阅 步骤 2：改进您的模型。

有关更多信息，请参阅精度和召回率。

调用

召回率指标用于解答以下问题：在测试数据集的异常图像总数中，有多少被正确预测为异常？

在假阴性成本很高的情况下，召回率指标十分有用。例如，当不拆除缺陷零件所造成的成本很高时。Amazon Lookout for Vision 为整个测试数据集提供了一种概要召回率指标值。

召回率 是正确检测出的异常测试图像的比例。当测试数据集的图像中存在异常图像时，它可以衡量模型正确预测异常图像的频率。召回率的计算公式如下所示：

召回率值 = 真阳性数/（真阳性数 + 假阴性数）

召回率介于 0-1 之间。Amazon Lookout for Vision 控制台以百分比值 (0-100) 的形式显示召回率。

召回率值越高，表示正确识别出的异常图像越多。例如，假设测试数据集包含 100 张异常图像。如果模型正确检测出 100 张异常图像中的 90 张，则召回率如下所示：

90 个真阳性 /（90 个真阳性 + 10 个假阴性）= 0.90 召回率值

召回率值为 0.90 表示您的模型正确预测出了测试数据集中的大多数异常图像。如果模型仅正确预测出 20 张异常图像，则召回率为较低的 0.20（即 20 / (20 + 80) = 0.20）。

在此情况下，您应考虑对模型进行改进。有关更多信息，请参阅 步骤 2：改进您的模型。

有关更多信息，请参阅精度和召回率。

F1 分数

Amazon Lookout for Vision 为测试数据集提供了一种平均模型性能分数。具体而言，异常分类的模型性能通过 F1 分数指标来衡量，该分数是精度分数和召回率分数的调和平均值。

F1 分数 是一种同时考虑精度和召回率的综合指标。模型性能分数是介于 0 和 1 之间的值。该值越高，表明模型在召回率和精度方面的表现越好。例如，对于精度为 0.9、召回率为 1.0 的模型，F1 分数为 0.947。

如果模型表现不佳，例如精度低至 0.30，召回率高达 1.0，则 F1 分数为 0.46。同样，如果精度较高 (0.95)，召回率较低 (0.20)，则 F1 分数为 0.33。这两种情况下，F1 分数都较低，表明模型存在问题。

有关更多信息，请参阅 F1 分数。

平均交并比 (IoU)

测试图像中的异常掩码与模型预测这些测试图像具有的异常掩码之间的平均重叠百分比。Amazon Lookout for Vision 会返回每个异常标签的平均 IoU，并且只能通过图像分割模型返回。

较低的百分比值表示，模型预测的标签掩码与测试图像中的掩码不准确匹配。

下面图像的 IoU 较低。橙色掩码是模型所做的预测，它不能紧密覆盖蓝色掩码，即测试图像中的掩码。

下面图像的 IoU 较高。蓝色掩码（测试图像）被橙色掩码（预测的掩码）紧密覆盖。

测试结果

在测试期间，模型会对测试数据集中的每张测试图像预测分类。每个预测的结果会与相应测试图像的标签（正常或异常）进行比较，如下所示：

- 若正确预测图像异常，则视为真阳性。
- 若错误预测图像异常，则视为假阳性。
- 若正确预测图像正常，则视为真阴性。
- 若错误预测图像正常，则视为假阴性。

如果模型是分割模型，则模型还会预测掩码和异常标签，用来表示测试图像上的异常位置。

Amazon Lookout for Vision 使用比较结果来生成绩效指标。

步骤 2：改进您的模型

性能指标可能表明您可以改进模型。例如，如果模型未检测出测试数据集中的所有异常，则表明您的模型的召回率较低（也就是说，召回率指标的值较低）。如果您需要改进模型，请考虑以下事项：

- 检查训练及测试数据集图像是否正确标注。
- 减少光照和物体姿态等图像捕获条件的可变性，并且使用相同类型的对象来训练模型。
- 确保您的图像仅显示所需的内容。例如，如果您的项目要检测机器零件中的异常，请确保图像中不存在其他对象。
- 向您的训练及测试数据集添加更多已标注的图像。如果测试数据集的召回率和精度非常出色，但模型在部署后表现不佳，则说明测试数据集可能代表性不足，您需要对其进行扩展。
- 如果测试数据集导致召回率和精度不佳，请考虑训练及测试数据集中的异常和图像捕获条件的匹配程度。如果训练图像不能代表预期的异常和条件，但测试图像中的图像可以代表，请向训练数据集中添加具有预期异常和条件的图像。如果测试数据集图像不在预期条件下，但训练图像在，请更新测试数据集。

  有关更多信息，请参阅 添加更多图像。向训练数据集中添加已标注图像的另一种方法是，执行试用检测任务并验证结果。然后，您可以将经过验证的图像添加到训练数据集中。有关更多信息，请参阅通过试用检测任务验证您的模型。
- 确保您的训练及测试数据集中具有足够多样的正常图像和异常图像。这些图像必须能够代表您的模型将会遇到的正常图像和异常图像类型。例如，在分析电路板时，您的正常图像应该能够代表电阻器和晶体管等组件的位置和焊接变化。异常图像应该能够代表系统可能会遇到的不同异常类型，如组件错位或缺失。
- 如果您的模型在检测到的异常类型方面具有低平均 IoU，请检查分割模型的掩码输出。在诸如划痕等一些使用场景下，模型输出的划痕可能非常接近测试图像中的真值划痕，但像素重叠度较低。例如，两条相隔 1 个像素的平行线。在这些情况下，平均 IoU 在衡量预测成功率方面是不可靠的指标。
- 如果图像尺寸较小或图像分辨率较低，请考虑以更高的分辨率捕获图像。图像尺寸可以从 64 x 64 像素到最高 4096 X 4096 像素。
- 如果异常尺寸很小，可以考虑将图像分成多个单独的图块，然后使用平铺的图像进行训练和测试。这样，模型就可以看到图像中较大尺寸的缺陷。

在改进训练及测试数据集后，请重新训练并重新评估您的模型。有关更多信息，请参阅 训练您的模型。

如果指标显示您的模型性能可接受，则可以将试用检测任务的结果添加到测试数据集，以此来验证其性能。重新训练后，性能指标应该确认上一次训练中的性能指标。有关更多信息，请参阅 通过试用检测任务验证您的模型。

查看性能指标

您可以从控制台和通过调用 DescribeModel 操作来获取性能指标。

主题

- 查看性能指标（控制台）
- 查看性能指标 (SDK)

查看性能指标（控制台）

训练完成后，控制台会显示性能指标。

Amazon Lookout for Vision 控制台会针对测试期间所做的分类显示以下性能指标：

- 精度
- 调用
- F1 分数

如果模型是分割模型，则控制台还会显示每个异常标签的以下性能指标：

- 从中发现异常标签的测试图像数量。
- F1 分数
- 平均交并比 (IoU)

测试结果概览部分会显示测试数据集中图像的正确预测和错误预测总数。您还可以看到测试数据集中各个图像的预测及实际标签分配情况。

以下过程展示了如何从项目的模型列表视图获取性能指标。

查看性能指标（控制台）

1. 打开 Amazon Lookout for Vision 控制台，网址为 https://console.aws.amazon.com/lookoutvision/。
2. 选择开始使用。
3. 在左侧导航窗格中，选择项目。

4. 在项目视图中，选择包含您要查看的模型版本的项目。
5. 在左侧导航窗格中，于项目名称下选择模型。
6. 在模型列表视图中，选择您要查看的模型的版本。
7. 在模型详细信息页面，查看性能指标选项卡上的性能指标。
8. 请注意以下几点：

    a. 在模型性能指标部分中，包含模型对测试图像所做的分类预测的总体模型指标（精度、召回率、F1 分数）。

    b. 如果模型是图像分割模型，则每标签性能部分中将包含从中发现异常标签的测试图像数量。您还会看到每个异常标签的指标（F1 分数、平均 IoU）。

    c. 测试结果概览部分提供了 Lookout for Vision 用来评估模型的每张测试图像的结果。其中包括以下内容：

    - 所有测试图像的正确（真阳性）和错误（假阴性）分类预测（正常或异常）总数。
    - 每张测试图像的分类预测。如果在图像下方看到正确，表明对图像预测的分类与实际分类一致。否则，表示模型未正确对图像分类。
    - 使用图像分割模型时，您可以看到模型分配给图像的异常标签，以及图像上与异常标签的颜色匹配的掩码。

查看性能指标 (SDK)

您可以使用 DescribeModel 操作获取模型的概要性能指标（分类）、评估清单和模型的评估结果。

获取概要绩效指标

对于模型在测试期间所做的分类预测，它们的概要性能指标（精度、调用和 F1 分数）会在通过调用 `DescribeModel` 返回的 `Performance` 字段中提供。

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

`Performance` 字段不包括由分割模型返回的异常标签性能指标。您可以在 `EvaluationResult` 字段中获取它们。有关更多信息，请参阅 查看评估结果。

有关概要性能指标的信息，请参阅步骤 1：评估模型的性能。有关示例代码，请参阅 查看您的模型 (SDK)。

使用评估清单

评估清单提供了用于测试模型的各个图像的测试预测指标。对于测试数据集中的每张图像，JSON 行中包含图像的原始测试（真值）信息和模型对图像的预测。Amazon Lookout for Vision 将评估清单存储在 Amazon S3 桶中。在 `DescribeModel` 操作的响应中，您可以从 `EvaluationManifest` 字段中获取该位置。

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

文件名格式为 `EvaluationManifest-`*`project name`*`.json`。有关示例代码，请参阅 查看您的模型。

在以下 JSON 示例行中，`class-name` 是图像内容的真值。在此示例中，图像包含异常。`confidence` 字段显示了 Amazon Lookout for Vision 对所做预测的置信度。

```
{
    "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"
    }
}
```

查看评估结果

对于整套测试图像，评估结果中具有以下综合性能指标（分类）：

- 精度
- 调用
- ROC 曲线（控制台中不显示）
- 平均精度（控制台中不显示）
- F1 分数

评估结果还包括用于训练和测试模型的图像数量。

如果模型是分割模型，则评估结果还包括测试数据集中发现的每个异常标签的以下指标：

- 精度
- 调用
- F1 分数
- 平均交并比 (IoU)

Amazon Lookout for Vision 将评估结果存储在 Amazon S3 桶中。在 `DescribeModel` 操作的响应中，您可以查看 `EvaluationResult` 字段的值来获取该位置。

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

```
        }
```

文件名格式为 EvaluationResult-*project name*.json。有关示例，请参阅 查看您的模型。

以下模式显示了评估结果。

```
   {
       "Version": 1,
       "EvaluationDetails":
       {
           "ModelArn": "string", // The Amazon Resource Name (ARN) of the model
version.
           "EvaluationEndTimestamp": "string",   // The UTC 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":
           {                       //Classification 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.
               "F1Score": float,           // The overal F1 score for the model.

               "PixelAnomalyClassMetrics":     //Segmentation metrics.
               [
                   {
                       "Precision": float,       // The precision for the anomaly
label.
                       "Recall": float,          // The recall for the anomaly label.
                       "F1Score": float,         // The F1 score for the anomaly
label.
                       "AIOU" : float,           // The average Intersection Over
Union for the anomaly label.
                       "ClassName": "string"   // The anomaly label.
                   }
               ]
           }
       }
```

```
    }
```

通过试用检测任务验证您的模型

如果要验证或提高模型的质量，您可以执行试用检测任务。试用检测任务可以检测您提供的新图像中的异常。

您可以验证检测结果，并且将经过验证的图像添加到数据集中。如果您有单独的训练数据集和测试数据集，则经过验证的图像会添加到训练数据集中。

您可以验证本地计算机中的图像，或 Amazon S3 桶中的图像。如果要将经过验证的图像添加到数据集中，S3 桶中的图像必须与数据集中的图像位于同一 S3 桶中。

> **Note**
>
> 要运行试用检测任务，请确保您的 S3 桶已启用版本控制。有关更多信息，请参阅使用版本控制。控制台桶是在启用版本控制的情况下创建的。

默认情况下，您的图像使用亚马逊云科技拥有和管理的密钥进行加密。您也可以选择使用自己的 Amazon Key Management Service (KMS) 密钥。有关更多信息，请参阅 Amazon Key Management Service 概念。

主题

- 运行试用检测任务
- 验证试用检测结果
- 使用注释工具更正分割标签

运行试用检测任务

执行以下步骤以运行试用检测任务。

运行试用检测（控制台）

1. 打开 Amazon Lookout for Vision 控制台，网址为 https://console.aws.amazon.com/lookoutvision/。
2. 选择开始使用。

3. 在左侧导航窗格中，选择项目。
4. 在项目视图中，选择包含您要查看的模型版本的项目。
5. 在左侧导航窗格中，于项目名称下选择试用检测。
6. 在试用检测视图中，选择运行试用检测。
7. 在运行试用检测页面，在任务名称中输入您的试用检测任务名称。
8. 在选择模型中，选择要使用的模型的版本。
9. 根据图像的来源导入图像，如下所示：

   - 如果要从 Amazon S3 桶导入源图像，请输入 S3 URI。

     > **Tip**
     >
     > 如果要使用“开始使用”示例图片，请使用 extra_images 文件夹。Amazon S3 URI 为 `s3://your bucket/circuitboard/extra_images`。

   - 如果要从您的计算机上传图像，请在选择检测异常后添加图像。
10. （可选）如果要使用自己的 Amazon KMS 加密密钥，请执行以下操作：

    a. 对于图像数据加密，请选择自定义加密设置（高级）。

    b. 在 encryption.aws_kms_key 中，输入您的密钥的 Amazon 资源名称 (ARN)，或者选择现有的 Amazon KMS 密钥。要创建新密钥，请选择创建 Amazon KMS 密钥。
11. 选择检测异常，然后选择运行试用检测以启动试用检测任务。
12. 在试用检测视图中查看当前状态。试用检测可能需要一段时间才能完成。

验证试用检测结果

验证试用检测结果可以帮助您改进模型。

如果性能指标不佳，请运行试用检测，然后将经过验证的图像添加到数据集（如果有单独的数据集，则添加到训练数据集），从而改进您的模型。

如果模型的性能指标良好，但试用检测的结果不佳，则可以将经过验证的图像添加到数据集（训练数据集），从而改进您的模型。如果您有单独的测试数据集，请考虑向测试数据集添加更多图像。

在将经过验证的图像添加到数据集后，请重新训练并重新评估您的模型。有关更多信息，请参阅 训练您的模型。

验证试用检测的结果

1. 打开 Amazon Lookout for Vision 控制台，网址为 https://console.aws.amazon.com/lookoutvision/。
2. 在左侧导航窗格中，选择项目。
3. 在项目页面上，选择要使用的项目。此时将显示您的项目的控制面板。
4. 在左侧导航窗格中，选择试用检测。
5. 选择要验证的试用检测。
6. 在试用检测页面上，选择验证机器预测。
7. 选中选择此页面上的所有图像。
8. 如果预测正确，请选择验证为正确。否则，请选择验证为不正确。每张图像下方会显示所做的预测和预测置信度分数。
9. 如果需要更改图像的标签，请执行以下操作：
   a. 在图像下方选择正确或不正确。
   b. 如果无法确定图像的正确标签，请选择图库中的图像以放大图像。

> **Note**
> 在筛选条件部分，您可以选择所需的标签或标签状态以筛选图像标签。在排序选项部分，您可以按置信度分数进行排序。

10. 如果您的模型是分割模型，并且图像的掩码或异常标签错误，请选择图像下方的异常区域并打开注释工具。通过执行以下操作：使用注释工具更正分割标签，更新分割信息。
11. 必要时在每页上重复步骤 7-10，直到所有图像都经过验证。
12. 选择将经过验证的图像添加到数据集。如果您有单独的数据集，则这些图像会添加到训练数据集中。
13. 重新训练您的模型。有关更多信息，请参阅 the section called “训练您的模型”。

使用注释工具更正分割标签

您可以使用注释工具，通过用掩码来标记异常区域以分割图像。

使用注释工具更正图像的分割标签

1. 在数据集图库中，选择图像下方的异常区域以打开注释工具。
2. 如果掩码的异常标签不正确，请选择该掩码，然后在异常标签下选择正确的异常标签。如有必要，请选择添加异常标签以添加新的异常标签。
3. 如果掩码不正确，请在页面底部选择绘图工具，然后为异常标签绘制紧密覆盖其异常区域的掩码。下面的图像是掩码紧密覆盖异常区域的示例。

下面的图像是掩码不佳，未紧密覆盖异常区域的示例。

4. 如果还有更多图像需要更正，请选择下一步，然后重复步骤 2 和步骤 3。
5. 选择提交并关闭，完成图像更新。

运行经过训练的 Amazon Lookout for Vision 模型

要使用您的模型检测图像中的异常，必须先使用 StartModel 操作来启动模型。Amazon Lookout for Vision 控制台提供了 AWS CLI 命令，可用于启动和停止您的模型。本节包括您可以使用的示例代码。

模型启动后，您可以使用 DetectAnomalies 操作来检测图像中的异常。有关更多信息，请参阅 检测图像中的异常。

主题

- 推理单位
- 可用区
- 启动您的 Amazon Lookout for Vision 模型
- 停止您的 Amazon Lookout for Vision 模型

推理单位

当您启动模型后，Amazon Lookout for Vision 会预调配至少一个计算资源，称为一个推理单位。您应该在 StartModel API 的 MinInferenceUnits 输入参数中指定要使用的推理单位数量。模型的默认分配值为 1 个推理单位。

> **Important**
>
> 您需要按照模型运行的小时数和模型在运行时使用的推理单位数付费，具体取决于您如何配置模型的运行。例如，如果使用两个推理单位启动模型并使用模型 8 小时，则需要支付 16 个推理小时（8 小时运行时间 x 两个推理单位）的费用。有关更多信息，请参阅 Amazon Lookout for Vision 定价。如果不通过调用 StopModel 明确停止模型，则即使您未主动使用模型来分析图像，也需要付费。

单个推理单位支持的每秒事务数 (TPS) 受以下因素影响：

- Lookout for Vision 用来训练模型的算法。当您训练模型时，多个模型会受到训练。Lookout for Vision 会根据数据集的大小及其正常图像和异常图像组成情况，从中选择性能最好的模型。
- 图像分辨率越高，需要的分析时间越长。
- 小图像（以 MB 为测量单位）的分析速度比大图像快。

使用推理单位管理吞吐量

根据您的应用需求，您可以提高或降低模型的吞吐量。要提高吞吐量，请使用更多推理单位。每增加一个推理单位，您的处理速度就会增加一个推理单位。有关计算所需推理单位数的信息，请参阅计算 Amazon Rekognition Custom Labels 模型和 Amazon Lookout for Vision 模型的推理单位数量。如果要更改模型支持的吞吐量，您有两种选择：

手动添加或移除推理单位

停止模型，然后使用所需数量的推理单位重新启动模型。这种方法的缺点是，模型在重新启动时无法接收请求，也不能用于应对需求高峰。如果您的模型具有稳定的吞吐量，并且您的使用场景可以容忍 10-20 分钟的停机时间，请使用此方法。例如，您想使用每周计划对模型进行批量调用。

自动扩缩推理单位数量

如果您的模型必须适应需求高峰，Amazon Lookout for Vision 可以自动扩缩模型使用的推理单位数量。随着需求的增加，Amazon Lookout for Vision 会向模型添加更多推理单位，并在需求下降时将其移除。

要允许 Lookout for Vision 自动为模型扩缩推理单位，请启动模型，然后使用 `MaxInferenceUnits` 参数设置模型可使用的最大推理单位数量。通过设置最大推理单位数，您可以限制可供模型使用的推理单位数量，以此来管理模型的运行成本。如果不指定最大单位数，Lookout for Vision 将不会自动扩缩模型，而只会使用您启动模型时所用的推理单位数量。有关最大推理单位数量的信息，请参阅服务限额。

您也可使用 `MinInferenceUnits` 参数指定最小推理单位数量。这可让您为模型指定最小吞吐量，其中一个推理单位代表 1 小时的处理时间。

> **Note**
>
> 您无法使用 Lookout for Vision 控制台设置最大推理单位数量，而应通过为 `StartModel` 操作指定 `MaxInferenceUnits` 输入参数来设置。

Lookout for Vision 提供了以下 Amazon CloudWatch Logs 指标，您可以使用它们来确定模型当前的自动扩缩状态。

指标	描述
DesiredInferenceUnits	Lookout for Vision 扩大或缩小到的推理单位数量。
InServiceInferenceUnits	模型正在使用的推理单位数量。

如果 DesiredInferenceUnits = InServiceInferenceUnits，则 Lookout for Vision 当前不会扩缩推理单位的数量。

如果 DesiredInferenceUnits > InServiceInferenceUnits，则 Lookout for Vision 会扩大到 DesiredInferenceUnits 的值。

如果 DesiredInferenceUnits < InServiceInferenceUnits，则 Lookout for Vision 会缩小到 DesiredInferenceUnits 的值。

有关 Lookout for Vision 返回的指标和筛选维度的更多信息，请参阅使用 Amazon CloudWatch 监控 Lookout for Vision。

要查明您为模型请求的最大推理单位数量，请调用 DescribeModel 并检查其响应中的 MaxInferenceUnits 字段。

可用区

Amazon Lookout for Vision 会在一个 AWS 区域内的多个可用区之间分配推理单位，以提供更高的可用性。有关更多信息，请参阅可用区。为帮助保护生产模型免受可用区中断和推理单位故障的影响，请使用至少两个推理单位启动生产模型。

如果可用区中断，则可用区中的所有推理单位都将无法使用，模型容量也会减少。对 DetectAnomalies 的调用将在剩余的推理单位之间重新分配。如果此类调用不超过剩余推理单位支持的每秒事务数 (TPS)，则它们会成功。在 AWS 修复可用区后，推理单位将重新启动，模型也将恢复其全部容量。

如果单个推理单位出现故障，Amazon Lookout for Vision 会自动在同一可用区内启动新的推理单位。在新推理单位启动之前，模型容量会降低。

启动您的 Amazon Lookout for Vision 模型

在使用 Amazon Lookout for Vision 模型进行异常检测之前，必须先启动该模型。您可以通过调用 StartModel API 并传递以下项来启动模型：

- ProjectName：包含要启动的模型的项目的名称。
- ModelVersion：要启动的模型版本。
- MinInferenceUnits：推理单位的最小数量。有关更多信息，请参阅 推理单位。
- （可选）MaxInferenceUnits：Amazon Lookout for Vision 可用来自动扩缩模型的最大推理单位数量。有关更多信息，请参阅 自动扩缩推理单位数量。

Amazon Lookout for Vision 控制台提供了示例代码，可用于启动和停止模型。

> **Note**
>
> 您需要按照模型的运行时间量付费。要停止正在运行的模型，请参阅停止您的 Amazon Lookout for Vision 模型。
> 您可以使用 AWS SDK，查看在所有可使用 Lookout for Vision 的 AWS 区域中运行的模型。有关示例代码，请参阅 find_running_models.py。

主题

- 启动您的模型（控制台）
- 启动您的 Amazon Lookout for Vision 模型 (SDK)

启动您的模型（控制台）

Amazon Lookout for Vision 控制台提供了 AWS CLI 命令，可用于启动模型。模型启动后，您便可以开始检测图像中的异常。有关更多信息，请参阅 检测图像中的异常。

启动模型（控制台）

1. 安装并配置 AWS CLI 和 AWS SDK（如果尚未如此）。有关更多信息，请参阅 步骤 4：设置 AWS CLI 和 AWS 软件开发工具包。
2. 打开 Amazon Lookout for Vision 控制台，网址为 https://console.aws.amazon.com/lookoutvision/。
3. 选择开始使用。

4. 在左侧导航窗格中，选择项目。
5. 在项目资源页面上，选择包含要启动的已训练模型的项目。
6. 在模型部分，选择要启动的模型。
7. 在模型的详细信息页面上，选择使用模型，然后选择将 API 集成到云。

> **Tip**
>
> 如果要将您的模型部署到边缘设备，请选择创建模型打包任务。有关更多信息，请参阅 将您的 Amazon Lookout for Vision 模型打包。

8. 在 Amazon CLI 命令下，复制用于调用 `start-model` 的 AWS CLI 命令。
9. 在命令提示符处，输入您在上一步中复制的 `start-model` 命令。如果您使用 `lookoutvision` 配置文件来获取凭证，请添加 `--profile lookoutvision-access` 参数。
10. 在控制台中，选择左侧导航页面中的模型。
11. 查看状态列以了解模型的当前状态，当状态为已托管时，您便可以使用模型来检测图像中的异常。有关更多信息，请参阅 检测图像中的异常。

启动您的 Amazon Lookout for Vision 模型 (SDK)

您可以通过调用 StartModel 操作来启动模型。

模型可能需要一段时间才能启动。您可以通过调用 DescribeModel 来检查当前状态。有关更多信息，请参阅 查看您的模型。

启动您的模型 (SDK)

1. 安装并配置 AWS CLI 和 AWS SDK（如果尚未如此）。有关更多信息，请参阅 步骤 4：设置 AWS CLI 和 AWS 软件开发工具包。
2. 使用以下示例代码启动模型。

   CLI

   更改以下值：

   - `project-name` 更改为包含要启动的模型的项目的名称。
   - `model-version` 更改为要启动的模型版本。
   - `--min-inference-units` 更改为要使用的推理单位数。

- （可选）--max-inference-units 更改为 Amazon Lookout for Vision 可用来自动扩缩模型的最大推理单位数量。

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

Python

此代码取自 AWS 文档 SDK 示例 GitHub 存储库。请在此处查看完整示例。

```
    @staticmethod
    def start_model(
            lookoutvision_client, project_name, model_version,
min_inference_units, max_inference_units = None):
        """
        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.
        :param max_inference_units: (Optional) The maximum number of inference
units that
        Lookout for Vision can use to automatically scale the model.
        """
        try:
            logger.info(
                "Starting model version %s for project %s", model_version,
project_name)

            if max_inference_units is None:
                lookoutvision_client.start_model(
                    ProjectName = project_name,
                    ModelVersion = model_version,
```

```
                MinInferenceUnits = min_inference_units)

        else:
            lookoutvision_client.start_model(
                ProjectName = project_name,
                ModelVersion = model_version,
                MinInferenceUnits = min_inference_units,
                MaxInferenceUnits = max_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
```

Java V2

此代码取自 AWS 文档 SDK 示例 GitHub 存储库。请在此处查看完整示例。

```
/**
 * Starts hosting an Amazon Lookout for Vision model. Returns when the model has
 * started or if hosting fails. You are charged for the amount of time that a
 * model is hosted. To stop hosting a model, use the StopModel operation.
 *
 * @param lfvClient   An Amazon Lookout for Vision client.
 * @param projectName The name of the project that contains the model that you
 *                    want to host.
 * @modelVersion The version of the model that you want to host.
 * @minInferenceUnits The number of inference units to use for hosting.
 * @maxInferenceUnits The maximum number of inference units that Lookout for
 *                    Vision can use for automatically scaling the model. If the
 *                    value is null, automatic scaling doesn't happen.
 * @return ModelDescription The description of the model, which includes the
 *         model hosting status.
 */
public static ModelDescription startModel(LookoutVisionClient lfvClient, String
 projectName, String modelVersion,
        Integer minInferenceUnits, Integer maxInferenceUnits) throws
 LookoutVisionException, InterruptedException {

    logger.log(Level.INFO, "Starting Model version {0} for project {1}.",
            new Object[] { modelVersion, projectName });

    StartModelRequest startModelRequest = null;

    if (maxInferenceUnits == null) {

        startModelRequest =
 StartModelRequest.builder().projectName(projectName).modelVersion(modelVersion)
                .minInferenceUnits(minInferenceUnits).build();
    } else {
        startModelRequest =
 StartModelRequest.builder().projectName(projectName).modelVersion(modelVersion)

 .minInferenceUnits(minInferenceUnits).maxInferenceUnits(maxInferenceUnits).build();
    }

    // Start hosting the model.
    lfvClient.startModel(startModelRequest);

    DescribeModelRequest describeModelRequest =
 DescribeModelRequest.builder().projectName(projectName)
```

```
            .modelVersion(modelVersion).build();

   ModelDescription modelDescription = null;

   boolean finished = false;
   // Wait until model is hosted or failure occurs.
   do {

       modelDescription =
lfvClient.describeModel(describeModelRequest).modelDescription();

       switch (modelDescription.status()) {

       case HOSTED:
           logger.log(Level.INFO, "Model version {0} for project {1} is
running.",
                   new Object[] { modelVersion, projectName });
           finished = true;
           break;

       case STARTING_HOSTING:
           logger.log(Level.INFO, "Model version {0} for project {1} is
starting.",
                   new Object[] { modelVersion, projectName });

           TimeUnit.SECONDS.sleep(60);

           break;
       case HOSTING_FAILED:
           logger.log(Level.SEVERE, "Hosting failed for model version {0} for
project {1}.",
                   new Object[] { modelVersion, projectName });
           finished = true;
           break;

       default:
           logger.log(Level.SEVERE, "Unexpected error when hosting model
version {0} for project {1}: {2}.",
                   new Object[] { projectName, modelVersion,
modelDescription.status() });
           finished = true;
           break;

       }
```

```
    } while (!finished);

    logger.log(Level.INFO, "Finished starting model version {0} for project {1}
 status: {2}",
            new Object[] { modelVersion, projectName,
 modelDescription.statusMessage() });

    return modelDescription;

}
```

3. 如果代码的输出为 `Model is hosted and ready for use`，则可以使用模型来检测图像中的异常。有关更多信息，请参阅 检测图像中的异常。

停止您的 Amazon Lookout for Vision 模型

要停止正在运行的模型，您可以调用 `StopModel` 操作并传递以下项：

- 项目：包含要停止的模型的项目的名称。
- ModelVersion：要停止的模型版本。

Amazon Lookout for Vision 控制台提供了示例代码，可用于停止模型。

> **Note**
>
> 您需要按照模型的运行时间量付费。

主题

- 停止您的模型（控制台）
- 停止您的 Amazon Lookout for Vision 模型 (SDK)

停止您的模型（控制台）

执行以下过程中的步骤，可以使用控制台停止您的模型。

停止您的模型（控制台）

1. 安装并配置 AWS CLI 和 AWS SDK（如果尚未如此）。有关更多信息，请参阅 步骤 4：设置 AWS CLI 和 AWS 软件开发工具包。
2. 打开 Amazon Lookout for Vision 控制台，网址为 https://console.aws.amazon.com/lookoutvision/。
3. 选择开始使用。
4. 在左侧导航窗格中，选择项目。
5. 在项目资源页面上，选择包含要停止的运行中模型的项目。
6. 在模型部分，选择要停止的模型。
7. 在模型的详细信息页面上，选择使用模型，然后选择将 API 集成到云。
8. 在 Amazon CLI 命令下，复制用于调用 `stop-model` 的 AWS CLI 命令。
9. 在命令提示符处，输入您在上一步中复制的 `stop-model` 命令。如果您使用 `lookoutvision` 配置文件来获取凭证，请添加 `--profile lookoutvision-access` 参数。
10. 在控制台，选择左侧导航页面中的模型。
11. 查看状态列以了解模型的当前状态。当状态列的值为训练完成时，表示模型已停止。

停止您的 Amazon Lookout for Vision 模型 (SDK)

您可以通过调用 StopModel 操作来停止模型。

模型可能需要一段时间才能停止。要检查当前状态，请使用 `DescribeModel`。

停止您的模型 (SDK)

1. 安装并配置 AWS CLI 和 AWS SDK（如果尚未如此）。有关更多信息，请参阅 步骤 4：设置 AWS CLI 和 AWS 软件开发工具包。
2. 使用以下示例代码停止正在运行的模型。

   CLI

   更改以下值：

   - `project-name` 更改为包含要停止的模型的项目的名称。
   - `model-version` 更改为要停止的模型版本。

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

Python

此代码取自 AWS 文档 SDK 示例 GitHub 存储库。请在此处查看完整示例。

```
    @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
```

Java V2

此代码取自 AWS 文档 SDK 示例 GitHub 存储库。请在此处查看完整示例。

```
/**
 * Stops the hosting an Amazon Lookout for Vision model. Returns when model has
 * stopped or if hosting fails.
 *
 * @param lfvClient   An Amazon Lookout for Vision client.
 * @param projectName The name of the project that contains the model that you
 *                    want to stop hosting.
 * @modelVersion The version of the model that you want to stop hosting.
 * @return ModelDescription The description of the model, which includes the
 *         model hosting status.
 */

public static ModelDescription stopModel(LookoutVisionClient lfvClient, String
 projectName,
                String modelVersion) throws LookoutVisionException,
 InterruptedException {

        logger.log(Level.INFO, "Stopping Model version {0} for project {1}.",
                        new Object[] { modelVersion, projectName });

        StopModelRequest stopModelRequest = StopModelRequest.builder()
```

```
                        .projectName(projectName)
                        .modelVersion(modelVersion)
                        .build();

        // Stop hosting the model.

        lfvClient.stopModel(stopModelRequest);

        DescribeModelRequest describeModelRequest =
DescribeModelRequest.builder()
                        .projectName(projectName)
                        .modelVersion(modelVersion)
                        .build();

        ModelDescription modelDescription = null;

        boolean finished = false;
        // Wait until model is stopped or failure occurs.
        do {

                modelDescription =
lfvClient.describeModel(describeModelRequest).modelDescription();

                switch (modelDescription.status()) {

                        case TRAINED:
                                logger.log(Level.INFO, "Model version {0} for
project {1} has stopped.",
                                                new Object[] { modelVersion,
projectName });
                                finished = true;
                                break;

                        case STOPPING_HOSTING:
                                logger.log(Level.INFO, "Model version {0} for
project {1} is stopping.",
                                                new Object[] { modelVersion,
projectName });

                                TimeUnit.SECONDS.sleep(60);

                                break;

                        default:
```

```
                                logger.log(Level.SEVERE,
                                                "Unexpected error when stopping
 model version {0} for project {1}: {2}.",
                                                new Object[] { projectName,
 modelVersion,

 modelDescription.status() });
                                finished = true;
                                break;

                }

        } while (!finished);

        logger.log(Level.INFO, "Finished stopping model version {0} for project
 {1} status: {2}",
                        new Object[] { modelVersion, projectName,
 modelDescription.statusMessage() });

        return modelDescription;

}
```

检测图像中的异常

要使用经过训练的 Amazon Lookout for Vision 模型来检测图像中的异常，您需要调用 DetectAnomalies 操作。DetectAnomalies 的结果中包括一个布尔预测值，该预测会按照图像包含一个异常或多个异常来对其进行分类，此外还包括对所做预测的置信度值。如果模型是图像分割模型，则结果中还包括有色掩码，用于显示不同类型异常的位置。

在宽度和高度尺寸上，您提供给 DetectAnomalies 的图像必须与用于训练模型的图像相同。

DetectAnomalies 可以接受 PNG 或 JPG 格式的图像。我们建议图像的编码和压缩格式与用于训练模型的图像相同。例如，如果您使用 PNG 格式的图像来训练模型，请使用 PNG 格式的图像调用 DetectAnomalies。

在调用 DetectAnomalies 之前，必须使用 StartModel 操作来启动模型。有关更多信息，请参阅启动您的 Amazon Lookout for Vision 模型。您需要按照模型的运行时间量（以分钟为单位）以及模型使用的异常检测单位数量付费。如果不再使用模型，请使用 StopModel 操作停止您的模型。有关更多信息，请参阅 停止您的 Amazon Lookout for Vision 模型。

主题

- 调用 DetectAnomalies
- 了解 DetectAnomalies 的响应
- 确定图像是否异常
- 显示分类和分割信息
- 使用 AWS Lambda 函数查找异常

调用 DetectAnomalies

要调用 DetectAnomalies，请指定以下内容：

- 项目：包含要使用的模型的项目的名称。
- ModelVersion：要使用的模型版本。
- ContentType：要分析的图像类型。有效值为 image/png（PNG 格式图像）和 image/jpeg（JPG 格式图像）。
- 主体：用来表示图像的未编码二进制字节。

  图像必须与用于训练模型的图像尺寸相同。

以下示例显示了如何调用 DetectAnomalies。您可以使用 Python 和 Java 示例中的函数响应，在确定图像是否异常中调用函数。

AWS CLI

此 AWS CLI 命令显示 DetectAnomalies CLI 操作的 JSON 输出。更改以下输入参数的值：

- project name 更改为要使用的项目的名称。
- model version 更改为要使用的模型版本。
- content type 更改为要使用的图像类型。有效值为 image/png（PNG 格式图像）和 image/jpeg（JPG 格式图像）。
- file name 更改为要使用的图像的路径和文件名。确保文件类型与 content-type 的值相匹配。

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

Python

有关完整的代码示例，请参阅 GitHub。

```
    def detect_anomalies(lookoutvision_client, project_name, model_version, photo):
        """
        Calls DetectAnomalies using the supplied project, model version, and image.
        :param lookoutvision_client: A Lookout for Vision Boto3 client.
        :param project: 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 photo that you want to analyze.
        :return: The DetectAnomalyResult object that contains the analysis results.
        """

        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}")


        # Get images bytes for call to detect_anomalies
        with open(photo, "rb") as image:
            response = lookoutvision_client.detect_anomalies(
                ProjectName=project_name,
                ContentType=content_type,
                Body=image.read(),
                ModelVersion=model_version)

        return response['DetectAnomalyResult']
```

Java V2

```
  public static DetectAnomalyResult detectAnomalies(LookoutVisionClient lfvClient,
String projectName,
            String modelVersion,
            String photo) throws IOException, LookoutVisionException {
        /**
         * Creates an Amazon Lookout for Vision dataset from a manifest file.
         * Returns after Lookout for Vision creates the dataset.
         *
         * @param lfvClient    An Amazon Lookout for Vision client.
         * @param projectName  The name of the project in which you want to create a
         *                     dataset.
         * @param modelVersion The version of the model that you want to use.
         *
         * @param photo        The photo that you want to analyze.
         *
         * @return DetectAnomalyResult The analysis result from DetectAnomalies.
         */

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

        // Get image bytes.

        InputStream sourceStream = new FileInputStream(new File(photo));
        SdkBytes imageSDKBytes = SdkBytes.fromInputStream(sourceStream);
        byte[] imageBytes = imageSDKBytes.asByteArray();
```

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

        // 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: Anomalous";
        }

        // Convert confidence to percentage.
        NumberFormat defaultFormat = NumberFormat.getPercentInstance();
        defaultFormat.setMinimumFractionDigits(1);
        String confidence = String.format("Confidence: %s",
defaultFormat.format(result.confidence()));

        // Log classification result.
        String photoPath = "File: " + photo;
        String[] imageLines = { photoPath, prediction, confidence };
        logger.log(Level.INFO, "Image: {0}\nAnomalous: {1}\nConfidence {2}",
imageLines);

        return result;

    }

    // Gets the image mime type. Supported formats are image/jpeg and image/png.
    private static 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));
    }
```

了解 DetectAnomalies 的响应

根据您训练的模型类型（分类模型或分割模型），DetectAnomalies 的响应会有所不同。两种情况下，响应都是 DetectAnomalyResult 对象。

分类模型

如果您的模型是图像分类模型，则 DetectAnomalies 的响应会包含以下内容：

- IsAnomalous：一种用于表示图像包含一个或多个异常的布尔指示符。
- 置信度：Amazon Lookout for Vision 对异常预测 (IsAnomalous) 准确度的信心。Confidence 是介于 0 和 1 之间的浮点值。值越高表示置信度越高。
- 来源：与传递给 DetectAnomalies 的图像相关的信息。

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

通过检查 IsAnomalous 字段并确认 Confidence 值是否高到足以满足您的需求，您可以确定图像是否存在异常。

如果您发现 DetectAnomalies 返回的置信度值过低，请考虑重新训练模型。有关示例代码，请参阅分类。

分割模型

如果您的模型是图像分割模型，则响应中将包括分类信息和分割信息，如图像掩码和异常类型。分类信息与分割信息分开进行计算，您不应假设它们之间存在关系。如果未在响应中获取分割信息，请检查是否已安装最新版本的 AWS SDK（如果使用的是 AWS CLI，则检查 AWS Command Line Interface）。有关示例代码，请参阅分段和显示分类和分割信息。

- IsAnomalous（分类）：一种用于将图像归类为正常或异常的布尔指示符。
- 置信度（分类）：Amazon Lookout for Vision 对图像分类 (IsAnomalous) 准确度的信心。Confidence 是介于 0 和 1 之间的浮点值。值越高表示置信度越高。
- 来源：与传递给 DetectAnomalies 的图像相关的信息。
- AnomalyMask（分割）：一种像素掩码，用于覆盖所分析图像中发现的异常。图像上可能存在多个异常。掩码贴图的颜色表示异常的类型。掩码颜色对应训练数据集中分配给各异常类型的颜色。要通过掩码颜色查找异常类型，请在 Anomalies 列表返回的每个异常的 PixelAnomaly 字段中检查 Color。有关示例代码，请参阅 显示分类和分割信息。
- 异常（分割）：图像中发现的异常列表。每个异常都包括异常类型 (Name) 和像素信息 (PixelAnomaly)。TotalPercentageArea 是图像中异常所覆盖的区域百分比。Color 是异常的掩码颜色。

  列表中的第一个元素始终是表示图像背景 (BACKGROUND) 的异常类型，不应将其视为异常。Amazon Lookout for Vision 会自动将背景异常类型添加到响应中。您无需在数据集中声明背景异常类型。

```
{
    "DetectAnomalyResult": {
        "Source": {
            "Type": "direct"
        },
        "IsAnomalous": true,
        "Confidence": 0.9996814727783203,
        "Anomalies": [
            {
                "Name": "background",
```

```
                "PixelAnomaly": {
                    "TotalPercentageArea": 0.998999834060669,
                    "Color": "#FFFFFF"
                }
            },
            {
                "Name": "scratch",
                "PixelAnomaly": {
                    "TotalPercentageArea": 0.0004034999874420464,
                    "Color": "#7ED321"
                }
            },
            {
                "Name": "dent",
                "PixelAnomaly": {
                    "TotalPercentageArea": 0.0005966666503809392,
                    "Color": "#4DD8FF"
                }
            }
        ],
        "AnomalyMask": "iVBORw0....."
    }
}
```

确定图像是否异常

您可以通过多种方式确定图像是否异常。选择哪种方法取决于您的使用场景和模型类型。以下是可能的解决方案。

主题

- 分类
- 分段

分类

IsAnomalous 可以将图像归类为异常，使用 Confidence 字段有助于确定该图像是否真的异常。值越高表示置信度越高。例如，只有当置信度超过 80% 时，您才可能会确定产品存在缺陷。您可以对分类模型或图像分割模型分析的图像进行分类。

Python

有关完整的代码示例，请参阅 GitHub。

```
   def reject_on_classification(image, prediction, confidence_limit):
       """
       Returns True if the anomaly confidence is greater than or equal to
       the supplied confidence limit.
       :param image: The name of the image file that was analyzed.
       :param prediction: The DetectAnomalyResult object returned from
DetectAnomalies
       :param confidence_limit: The minimum acceptable confidence. Float value
between 0 and 1.
       :return: True if the error condition indicates an anomaly, otherwise False.
       """

       reject = False

       logger.info("Checking classification for %s", image)

       if prediction['IsAnomalous'] and prediction['Confidence'] >=
confidence_limit:
           reject = True
           reject_info=(f"Rejected: Anomaly confidence
({prediction['Confidence']:.2%}) is greater"
                   f" than limit ({confidence_limit:.2%})")
           logger.info("%s", reject_info)

       if not reject:
           logger.info("No anomalies found.")
       return reject
```

Java V2

```
   public static boolean rejectOnClassification(String image, DetectAnomalyResult
prediction, float minConfidence) {
       /**
        * Rejects an image based on its anomaly classification and prediction
        * confidence
        *
        * @param image         The file name of the analyzed image.
        * @param prediction    The prediction for an image analyzed with
        *                      DetectAnomalies.
```

```
         * @param minConfidence The minimum acceptable confidence for the prediction
         *                      (0-1).
         *
         * @return boolean True if the image is anomalous, otherwise False.
         */

        Boolean reject = false;

        logger.log(Level.INFO, "Checking classification for {0}", image);

        String[] logParameters = { prediction.confidence().toString(),
String.valueOf(minConfidence) };

        if (Boolean.TRUE.equals(prediction.isAnomalous()) && prediction.confidence()
>= minConfidence) {
            logger.log(Level.INFO, "Rejected: Anomaly confidence {0} is greater than
confidence limit {1}",
                    logParameters);
            reject = true;
        }
        if (Boolean.FALSE.equals(reject))
            logger.log(Level.INFO, ": No anomalies found.");

        return reject;

    }
```

分段

如果您的模型是图像分割模型，则可以使用分割信息来确定图像是否包含异常。您还可以使用图像分割模型对图像进行分类。有关获取和显示图像掩码的示例代码，请参阅显示分类和分割信息

异常区域

使用异常在图像上的覆盖百分比 (TotalPercentageArea)。例如，如果异常区域大于图像的 1%，则您可以确定产品存在缺陷。

Python

有关完整的代码示例，请参阅 GitHub。

```
   def reject_on_coverage(image, prediction, confidence_limit, anomaly_label,
 coverage_limit):
       """
       Checks if the coverage area of an anomaly is greater than the coverage limit
 and if
       the prediction confidence is greater than the confidence limit.
       :param image: The name of the image file that was analyzed.
       :param prediction: The DetectAnomalyResult object returned from
 DetectAnomalies
       :param confidence_limit: The minimum acceptable confidence (float 0-1).
       :anomaly_label: The anomaly label for the type of anomaly that you want to
 check.
       :coverage_limit: The maximum acceptable percentage coverage of an anomaly
 (float 0-1).
       :return: True if the error condition indicates an anomaly, otherwise False.
       """

       reject = False

       logger.info("Checking coverage for %s", image)

       if prediction['IsAnomalous'] and prediction['Confidence'] >=
 confidence_limit:
           for anomaly in prediction['Anomalies']:
               if (anomaly['Name'] == anomaly_label and
                       anomaly['PixelAnomaly']['TotalPercentageArea'] >
 (coverage_limit)):
                   reject = True
                   reject_info=(f"Rejected: Anomaly confidence
 ({prediction['Confidence']:.2%}) "
                       f"is greater than limit ({confidence_limit:.2%}) and
 {anomaly['Name']} "
                       f"coverage ({anomaly['PixelAnomaly']
['TotalPercentageArea']:.2%}) "
                       f"is greater than limit ({coverage_limit:.2%})")

                   logger.info("%s", reject_info)

       if not reject:
           logger.info("No anomalies found.")

       return reject
```

Java V2

```
   public static Boolean rejectOnCoverage(String image, DetectAnomalyResult
prediction, float minConfidence,
           String anomalyType, float maxCoverage) {
       /**
        * Rejects an image based on a maximum allowable coverage area for an
anomaly
        * type.
        *
        * @param image         The file name of the analyzed image.
        * @param prediction    The prediction for an image analyzed with
        *                      DetectAnomalies.
        * @param minConfidence The minimum acceptable confidence for the prediction
        *                      (0-1).
        * @param anomalyTypes  The anomaly type to check.
        * @param maxCoverage   The maximum allowable coverage area of the anomaly
type.
        *                      (0-1).
        *
        * @return boolean True if the coverage area of the anomaly type exceeds the
        *         maximum allowed, otherwise False.
        */

       Boolean reject = false;

       logger.log(Level.INFO, "Checking coverage for {0}", image);

       if (Boolean.TRUE.equals(prediction.isAnomalous()) && prediction.confidence()
>= minConfidence) {
           for (Anomaly anomaly : prediction.anomalies()) {

               if (Objects.equals(anomaly.name(), anomalyType)
                       && anomaly.pixelAnomaly().totalPercentageArea() >=
maxCoverage) {

                   String[] logParameters = { prediction.confidence().toString(),
                           String.valueOf(minConfidence),

String.valueOf(anomaly.pixelAnomaly().totalPercentageArea()),
                           String.valueOf(maxCoverage) };
                   logger.log(Level.INFO,
                           "Rejected: Anomaly confidence {0} is greater than
confidence limit {1} and " +
```

```
                                    "{2} anomaly type coverage is higher than
coverage limit {3}\n",
                            logParameters);
                    reject = true;

                }
            }
        }

        if (Boolean.FALSE.equals(reject))
            logger.log(Level.INFO, ": No anomalies found.");

        return reject;
    }
```

异常类型数量

使用图像上发现的不同异常类型 (Name) 的计数。例如，如果存在两种以上的异常，则您可以确定产品存在缺陷。

Python

有关完整的代码示例，请参阅 GitHub。

```
    def reject_on_anomaly_types(image, prediction, confidence_limit,
anomaly_types_limit):
        """
        Checks if the number of anomaly types is greater than than the anomaly types
        limit and if the prediction confidence is greater than the confidence limit.
        :param image: The name of the image file that was analyzed.
        :param prediction: The DetectAnomalyResult object returned from
DetectAnomalies
        :param confidence: The minimum acceptable confidence. Float value between 0
and 1.
        :param anomaly_types_limit: The maximum number of allowable anomaly types
(Integer).
        :return: True if the error condition indicates an anomaly, otherwise False.
        """

        logger.info("Checking number of anomaly types for %s",image)

        reject = False
```

```
       if prediction['IsAnomalous'] and prediction['Confidence'] >=
confidence_limit:

           anomaly_types = {anomaly['Name'] for anomaly in prediction['Anomalies']\
               if anomaly['Name'] != 'background'}

           if len (anomaly_types) > anomaly_types_limit:
                   reject = True
                   reject_info = (f"Rejected: Anomaly confidence
({prediction['Confidence']:.2%}) "
                   f"is greater than limit ({confidence_limit:.2%}) and "
                   f"the number of anomaly types ({len(anomaly_types)-1}) is "
                   f"greater than the limit ({anomaly_types_limit})")

                   logger.info("%s", reject_info)

       if not reject:
           logger.info("No anomalies found.")
       return reject
```

Java V2

```
   public static Boolean rejectOnAnomalyTypeCount(String image, DetectAnomalyResult
prediction,
           float minConfidence, Integer maxAnomalyTypes) {

       /**
        * Rejects an image based on a maximum allowable number of anomaly types.
        *
        * @param image           The file name of the analyzed image.
        * @param prediction      The prediction for an image analyzed with
        *                        DetectAnomalies.
        * @param minConfidence   The minimum acceptable confidence for the
predictio
        *                        (0-1).
        * @param maxAnomalyTypes The maximum allowable number of anomaly types.
        *
        * @return boolean True if the image contains more than the maximum allowed
        *         anomaly types, otherwise False.
        */

       Boolean reject = false;
```

```
        logger.log(Level.INFO, "Checking coverage for {0}", image);

        Set<String> defectTypes = new HashSet<>();

        if (Boolean.TRUE.equals(prediction.isAnomalous()) && prediction.confidence()
>= minConfidence) {
            for (Anomaly anomaly : prediction.anomalies()) {
                defectTypes.add(anomaly.name());
            }
            // Reduce defect types by one to account for 'background' anomaly type.
            if ((defectTypes.size() - 1) > maxAnomalyTypes) {
                String[] logParameters = { prediction.confidence().toString(),
                        String.valueOf(minConfidence),
                        String.valueOf(defectTypes.size()),
                        String.valueOf(maxAnomalyTypes) };
                logger.log(Level.INFO, "Rejected: Anomaly confidence {0} is >=
minimum confidence {1} and " +
                        "the number of anomaly types {2} > the allowable number of
anomaly types {3}\n", logParameters);
                reject = true;
            }

        }

        if (Boolean.FALSE.equals(reject))
            logger.log(Level.INFO, ": No anomalies found.");

        return reject;
    }
```

显示分类和分割信息

此示例显示所分析的图像并叠加分析结果。如果响应中包括异常掩码，则掩码将以关联的异常类型的颜色显示。

显示图像分类和图像分割信息

1. 执行以下操作(如果尚未这样做)：

   a. 安装并配置 AWS CLI 和 AWS SDK（如果尚未如此）。有关更多信息，请参阅 步骤 4：设置 AWS CLI 和 AWS 软件开发工具包。

b. 训练您的模型。

c. 启动您的模型。

2. 确保调用 DetectAnomalies 的用户有权访问您要使用的模型版本。有关更多信息，请参阅 设置 SDK 权限。

3. 使用以下代码。

Python

以下示例代码可检测您提供的图像中是否存在异常。该示例采用以下命令行选项：

- project：要使用的项目的名称。
- version：项目中要使用的模型版本。
- image：本地图像文件（JPEG 或 PNG 格式）的路径和文件。

```
# Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
# SPDX-License-Identifier: Apache-2.0
"""
Shows how to detect and show anomalies in an image using a trained Amazon
 Lookout
for Vision model. The script displays the analysed image and overlays mask and
 analysis
output.
"""

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

from botocore.exceptions import ClientError

logger = logging.getLogger(__name__)


class ShowAnomalies:
    """
    Class to detect and show anomalies in an image analyzed by detect_anomalies.
    """
```

```
    @staticmethod
    def draw_line(draw, text, fnt,  y_coordinate, color):
        """
        Draws a line of text on the supplied drawing surface.
        :param draw: The surface on which to draw the text.
        :param text: The text to draw in the drawing surface.
        :param fnt: The font for the text.
        :param y_coordinate: The y position for the text.
        :param color: The color for the text.
        :returns The y coordinate for the next line of text.
        """
        text_width, text_height = draw.textsize(text, fnt)
        draw.rectangle([(10, y_coordinate), (text_width + 10,
                                             y_coordinate + text_height)],
 fill="black")
        draw.text((10, y_coordinate), text, fill=color, font=fnt)

        y_coordinate += text_height

        return y_coordinate

    @staticmethod
    def draw_analysis_text(image, analysis):
        """
        Draws classification and segmentation info onto supplied image
        overlay analysis results on an image analyzed by detect_anomalies.
        :param analysis: The response from a call to detect_anomalies.
        :returns Nothing
        """

        ## Calculate a reasonable font size based on image width.
        font_size = int(image.size[0]/32)

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

        draw = ImageDraw.Draw(image)

        y_coordinate = 0

        # Draw classification information.
        prediction = "Anomalous" if analysis["DetectAnomalyResult"]
["IsAnomalous"] \
            else "Normal"
```

```
        confidence = analysis["DetectAnomalyResult"]["Confidence"]
        found_anomalies = analysis["DetectAnomalyResult"]['Anomalies']
        segmentation_info = False

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

        y_coordinate = 0
        y_coordinate = ShowAnomalies.draw_line(
            draw, "Classification", fnt, y_coordinate, "white")
        y_coordinate = ShowAnomalies.draw_line(
            draw, f"  Prediction: {prediction}", fnt, y_coordinate, "white")
        y_coordinate = ShowAnomalies.draw_line(
            draw, f"  Confidence: {confidence:.2%}", fnt, y_coordinate, "white")

        # Draw segmentation information, if present.
        if (len(found_anomalies)) > 1:
            logger.info("Anomalies:")

            y_coordinate = ShowAnomalies.draw_line(
                draw, "Segmentation:", fnt, y_coordinate, "white")
            for i in range(1, len(found_anomalies)):

                # Only display info if more than 0% coverage found.
                percent_coverage = found_anomalies[i]['PixelAnomaly']
['TotalPercentageArea']
                if percent_coverage > 0:
                    segmentation_info = True
                    logger.info("  %s", found_anomalies[i]['Name'])
                    logger.info("    Color: %s",
                                found_anomalies[i]['PixelAnomaly']['Color'])
                    logger.info("    Area: %s", percent_coverage)
                    y_coordinate = ShowAnomalies.draw_line(
                        draw,
                        f"  Anomaly: {found_anomalies[i]['Name']}. Area:
 {percent_coverage:.2%}",
                        fnt,
                        y_coordinate,
                        found_anomalies[i]['PixelAnomaly']['Color'])

            if not segmentation_info:
                y_coordinate = ShowAnomalies.draw_line(
```

```
                    draw, "No segmentation information found.", fnt,
y_coordinate, "white")




   @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:

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

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

           # Check that image type is valid.
           if image_type not in ("image/jpeg", "image/png"):
               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
            )

            # Overlay mask onto analyzed image.
            image_mask_bytes = response["DetectAnomalyResult"]["AnomalyMask"]
            image_mask = Image.open(io.BytesIO(image_mask_bytes))

            final_img = Image.blend(image, image_mask, 0.5) \
                if response["DetectAnomalyResult"]["IsAnomalous"] else image

            # Overlay analysis output on image.
            ShowAnomalies.draw_analysis_text(final_img, response)

            final_img.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")

        session = boto3.Session(
            profile_name='lookoutvision-access')

        lookoutvision_client = session.client("lookoutvision")

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

        add_arguments(parser)

        args = parser.parse_args()

        # Analyze the image and show results.
        ShowAnomalies.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

以下示例代码可检测您提供的图像中是否存在异常。该示例采用以下命令行选项：

- `project`：要使用的项目的名称。
- `version`：项目中要使用的模型版本。
- `image`：本地图像文件（JPEG 或 PNG 格式）的路径和文件。

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


package com.example.lookoutvision;

import software.amazon.awssdk.auth.credentials.ProfileCredentialsProvider;
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.Anomaly;
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 ShowAnomalies extends JPanel {
```

```
/**
 * Finds and displays anomalies on a supplied image.
 */

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

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

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

       maskImage = null;

       // 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();


        if (result.anomalyMask() != null){
            SdkBytes maskSDKBytes = result.anomalyMask();

            ByteArrayInputStream maskInputStream = new
ByteArrayInputStream(maskSDKBytes.asByteArray());
            maskImage = ImageIO.read(maskInputStream);
        }

        drawImageInfo(result);

    }

    // 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;
        dimension.height = (int) dimension.getHeight() / 2;

        if (image.getWidth() < dimension.width || image.getHeight() <
dimension.height) {
            dimension.width = image.getWidth();
            dimension.height = image.getHeight();
        }
        setPreferredSize(dimension);

    }

    private int drawLine(Graphics2D g2d, String line, FontMetrics metrics, int
yPos, Color color) {
    /**
    * Draws a line of text at the spsecified y position and color.
    * confidence
    *
    * @param g2D The Graphics2D object for the image.
    * @param line The line of text to draw.
    * @param metrics The font information to use.
```

```
    * @param yPos The y position for the line of text.
    *
    * @return  The yPos for the next line of text.
    */

        int indent = 10;

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

        int y2Pos = (yPos + textHeight) - descent;

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

        // Draw text.
        g2d.setColor(color);
        g2d.drawString(line, indent, y2Pos);

        yPos += textHeight;

        return yPos;

    }

    public void drawImageInfo(DetectAnomalyResult result) {
    /**
     * Draws the results from DetectAnomalies onto the output image.
     *
     * @param result The response from a call to
     *               DetectAnomalies.
     *
     */

        // Set up drawing.
        Graphics2D g2d = image.createGraphics();

        if (result.anomalyMask() != null){
            Composite composite = g2d.getComposite();
            g2d.setComposite(AlphaComposite.SrcOver.derive(0.5f));
```

```
            int x = (image.getWidth() - maskImage.getWidth()) / 2;
            int y = (image.getHeight() - maskImage.getHeight()) / 2;
            g2d.drawImage(maskImage, x, y, null);
            // Set composite for overlaying text.
            g2d.setComposite(composite);
        }

        //Calculate font size based on arbitary 32 pixel image width.
        int fontSize = (image.getWidth() / 32);


        g2d.setFont(new Font("Tahoma", Font.PLAIN, fontSize));
        Font font = g2d.getFont();
        FontMetrics metrics = g2d.getFontMetrics(font);

        // Get classification information.

        String prediction = "Prediction: Normal";

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

        // Convert prediction to percentage.
        NumberFormat defaultFormat = NumberFormat.getPercentInstance();
        defaultFormat.setMinimumFractionDigits(1);
        String confidence = String.format("Confidence: %s",
defaultFormat.format(result.confidence()));

        // Draw classification information.
        int yPos = 0;

        yPos = drawLine(g2d, "Classification:", metrics, yPos, Color.WHITE);
        yPos = drawLine(g2d, prediction, metrics, yPos, Color.WHITE);
        yPos = drawLine(g2d, confidence, metrics, yPos, Color.WHITE);

        // Draw segmentation info.
        yPos = drawLine(g2d, "Segmentation:", metrics, yPos, Color.WHITE);

        // Ignore background label, so size must be > 1
        if (result.anomalies().size() > 1) {
            for (Anomaly anomaly : result.anomalies()) {
                if (anomaly.name().equals("background"))
                    continue;
```

```
                String label = String.format("Anomaly: %s. Area: %s",
anomaly.name(),

defaultFormat.format(anomaly.pixelAnomaly().totalPercentageArea()));
                Color anomalyColor =
Color.decode((anomaly.pixelAnomaly().color()));
                yPos = drawLine(g2d, label, metrics, yPos, anomalyColor);

            }

        } else {
            drawLine(g2d, "None found.", metrics, yPos, Color.WHITE);

        }

        g2d.dispose();

    }

    @Override
    public void paintComponent(Graphics g)
    /**
     * Draws the image and analysis results.
     *
     * @param g The Graphics context object for drawing.
     *          DetectAnomalies.
     *
     */
    {

        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
    /**
     * Gets the file type of a supplied image. Raises an exception if the image
     * isn't compatible with with Amazon Lookout for Vision.
     *
```

```
     * @param image The image that you want to check.
     *
     * @return String The type of the image.
     */

    {
        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];
            ShowAnomalies panel = null;

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

.credentialsProvider(ProfileCredentialsProvider.create("lookoutvision-access"))
                .build();

            // Create frame and panel.
            JFrame frame = new JFrame(photo);
            frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

            panel = new ShowAnomalies(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}:
{1}",
                    new Object[] { lfvError.awsErrorDetails().errorCode(),
                            lfvError.awsErrorDetails().errorMessage() });
            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. 如果不打算继续使用您的模型，请停止您的模型。

使用 AWS Lambda 函数查找异常

AWS Lambda 是一项计算服务，可使您无需预配置或管理服务器即可运行代码。例如，您可以从移动应用程序分析所提交的图像，而不必创建服务器来托管应用程序代码。以下说明展示了如何在 Python 中创建用来调用 DetectAnomalies 的 Lambda 函数。该函数会分析所提供的图像，并返回分类结果，表明该图像中是否存在异常。这些说明中包括示例 Python 代码，用于展示如何使用 Amazon S3 桶中的图像或本地计算机提供的图像调用 Lambda 函数。

主题

- 步骤 1：创建 AWS Lambda 函数（控制台）
- 步骤 2：（可选）创建层（控制台）
- 步骤 3：添加 Python 代码（控制台）
- 步骤 4：试用您的 Lambda 函数

步骤 1：创建 AWS Lambda 函数（控制台）

在此步骤中，您将创建一个空 AWS 函数，以及一个允许您的函数调用 `DetectAnomalies` 操作的 IAM 执行角色。针对存储供分析图像的 Amazon S3 桶，它还可以授予对该桶的访问权限。您还可以为以下项指定环境变量：

- 您希望 Lambda 函数使用的 Amazon Lookout for Vision 项目和模型版本。
- 您希望模型使用的置信限。

随后，您可以向 Lambda 函数添加源代码，也可以选择添加一个层。

创建 AWS Lambda 函数（控制台）

1. 登录到 AWS Management Console，然后通过以下网址打开 AWS Lambda 控制台：https://console.aws.amazon.com/lambda/。
2. 选择 Create function（创建函数）。有关更多信息，请参阅使用控制台创建 Lambda 函数。

3. 选择以下选项。
   - 选择从头开始编写。
   - 为函数名称输入一个值。
   - 对于运行时，请选择 Python 3.10。
4. 选择创建函数以创建 AWS Lambda 函数。
5. 在函数页面上，选择配置选项卡。
6. 在环境变量窗格中，选择编辑。
7. 添加以下环境变量。对于每个变量，请选择添加环境变量，然后输入变量键和值。

键	Value
PROJECT_NAME	包含您要使用的模型的 Lookout for Vision 项目。
MODEL_VERSION	您要使用的模型版本。
CONFIDENCE	对于预测结果为异常，模型的置信度最小值 (0-100)。如果置信度较低，则分类视为正常。

8. 选择保存以保存环境变量。
9. 在权限窗格的角色名称下，选择执行角色以在 IAM 控制台中打开角色。
10. 在权限选项卡中，选择添加权限，然后选择创建内联策略。
11. 选择 JSON，将现有策略替换为以下策略。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Action": "lookoutvision:DetectAnomalies",
            "Resource": "*",
            "Effect": "Allow",
            "Sid": "DetectAnomaliesAccess"
        }
    ]
}
```

12. 选择 Next（下一步）。
13. 在策略详细信息中，输入策略的名称，如 DetectAnomalies-access。
14. 选择 Create policy（创建策略）。
15. 如果供分析的图像存储在 Amazon S3 桶中，请重复步骤 10-14。

    a. 在步骤 11 中，使用以下策略。将 *bucket/folder path* 替换为要分析的图像所在的 Amazon S3 桶和文件夹路径。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "S3Access",
            "Effect": "Allow",
            "Action": "s3:GetObject",
            "Resource": "arn:aws:s3:::bucket/folder path/*"
        }
    ]
}
```

    b. 在步骤 13 中，选择其他策略名称，如 S3Bucket-access。

步骤 2：（可选）创建层（控制台）

要运行此示例，您无需执行此步骤。`DetectAnomalies` 操作会作为适用于 Python 的 AWS SDK (Boto3) 的一部分，包含在默认 Lambda Python 环境中。如果 Lambda 函数的其他部分需要最新的 AWS 服务更新，而这些更新不在默认 Lambda Python 环境中，请执行此步骤，以便将最新的 Boto3 SDK 版本作为一个层添加到函数中。

首先，创建包含 Boto3 SDK 的 .zip 文件归档。然后，创建一个层并将 .zip 文件归档添加到该层。有关更多信息，请参阅组合使用层与 Lambda 函数。

创建并添加层（控制台）

1. 打开命令提示符，然后输入以下命令。

```
pip install boto3 --target python/.
zip boto3-layer.zip -r python/
```

2. 记下压缩文件的名称 (boto3-layer.zip)。您需要在本过程的步骤 6 中使用它。
3. 打开 AWS Lambda 控制台，地址：https://console.aws.amazon.com/lambda/。
4. 在导航窗格中，选择 Layers（层）。
5. 选择 Create layer（创建层）。
6. 为 Name（名称）和 Description（描述）输入值。
7. 选择上传 .zip 文件，然后选择上传。
8. 在对话框中，选择您在本过程步骤 1 中创建的 .zip 文件归档 (boto3-layer.zip)。
9. 对于兼容的运行时，请选择 Python 3.9。
10. 选择创建以创建层。
11. 选择导航窗格菜单图标。
12. 在导航窗格中，选择函数。
13. 在资源列表中，选择您在步骤 1：创建 AWS Lambda 函数（控制台）中创建的函数。
14. 选择节点选项卡。
15. 在层部分，选择添加层。
16. 选择自定义层。
17. 在自定义层中，选择您在步骤 6 中输入的层名称。
18. 在版本中，选择层版本，该版本应为 1。
19. 选择 Add（添加）。

步骤 3：添加 Python 代码（控制台）

在此步骤中，您将使用 Lambda 控制台代码编辑器，向您的 Lambda 函数添加 Python 代码。该代码可以使用 `DetectAnomalies` 分析所提供的图像，并返回分类结果（如果图像异常，则返回 true；如果图像正常，则返回 false）。所提供的图像可以位于 Amazon S3 桶中，或者以 byte64 编码图像字节的形式提供。

添加 Python 代码（控制台）

1. 如果您不在 Lambda 控制台中，请执行以下操作：
   a. 打开 AWS Lambda 控制台，地址：https://console.aws.amazon.com/lambda/。
   b. 打开您在步骤 1：创建 AWS Lambda 函数（控制台）中创建的 Lambda 函数。
2. 选择节点选项卡。

3. 在代码源中，将 lambda_function.py 中的代码替换为以下代码：

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

"""
Purpose
An AWS lambda function that analyzes images with an Amazon Lookout for Vision
 model.
"""
import base64
import imghdr
from os import environ
from io import BytesIO
import logging
import boto3

from botocore.exceptions import ClientError


logger = logging.getLogger(__name__)


# Get the model and confidence.
project_name = environ['PROJECT_NAME']
model_version = environ['MODEL_VERSION']
min_confidence = int(environ.get('CONFIDENCE', 50))

lookoutvision_client = boto3.client('lookoutvision')


def lambda_handler(event, context):
    """
    Lambda handler function
    param: event: The event object for the Lambda function.
    param: context: The context object for the lambda function.
    return: The labels found in the image passed in the event
    object.
    """

    try:

        file_name = ""
```

```
        # Determine image source.
        if 'image' in event:
            # Decode the encoded image
            image_bytes = event['image'].encode('utf-8')
            img_b64decoded = base64.b64decode(image_bytes)
            image_type = get_image_type(img_b64decoded)
            image = BytesIO(img_b64decoded)
            file_name = event['filename']

        elif 'S3Object' in event:
            bucket = boto3.resource('s3').Bucket(event['S3Object']['Bucket'])
            image_object = bucket.Object(event['S3Object']['Name'])
            image = image_object.get().get('Body').read()
            image_type = get_image_type(image)
            file_name = f"s3://{event['S3Object']['Bucket']}/{event['S3Object']
['Name']}"

        else:
            raise ValueError(
                'Invalid image source. Only base 64 encoded image bytes or images
 in S3 buckets are supported.')

        # Analyze the image.
        response = lookoutvision_client.detect_anomalies(
            ProjectName=project_name,
            ContentType=image_type,
            Body=image,
            ModelVersion=model_version)

        reject = reject_on_classification(
            response['DetectAnomalyResult'],
 confidence_limit=float(environ['CONFIDENCE'])/100)

        status = "anomalous" if reject else "normal"

        lambda_response = {
            "statusCode": 200,
            "body": {
                "Reject": reject,
                "RejectMessage": f"Image {file_name} is {status}."
            }
        }
```

```
    except ClientError as err:
        error_message = f"Couldn't analyze {file_name}. " + \
            err.response['Error']['Message']

        lambda_response = {
            'statusCode': 400,
            'body': {
                "Error": err.response['Error']['Code'],
                "ErrorMessage": error_message,
                "Image": file_name
            }
        }
        logger.error("Error function %s: %s",
                     context.invoked_function_arn, error_message)

    except ValueError as val_error:

        lambda_response = {
            'statusCode': 400,
            'body': {
                "Error": "ValueError",
                "ErrorMessage": format(val_error),
                "Image": event['filename']
            }
        }
        logger.error("Error function %s: %s",
                     context.invoked_function_arn, format(val_error))

    return lambda_response

def get_image_type(image):
    """
    Gets the format of the image. Raises an error
    if the type is not PNG or JPEG.
    :param image: The image that you want to check.
    :return The type of the image.

    """
    image_type = imghdr.what(None, image)

    if image_type == "jpeg":
        content_type = "image/jpeg"
    elif image_type == "png":
        content_type = "image/png"
```

```
    else:
        logger.info("Invalid image type")
        raise ValueError(
            "Invalid file format. Supply a jpeg or png format file.")
    return content_type


def reject_on_classification(prediction, confidence_limit):
    """
    Returns True if the anomaly confidence is greater than or equal to
    the supplied confidence limit.
    :param image: The name of the image file that was analyzed.
    :param prediction: The DetectAnomalyResult object returned from DetectAnomalies
    :param confidence_limit: The minimum acceptable confidence. Float value between
 0 and 1.
    :return: True if the error condition indicates an anomaly, otherwise False.
    """

    reject = False

    if prediction['IsAnomalous'] and prediction['Confidence'] >= confidence_limit:
        reject = True
        reject_info = (f"Rejected: Anomaly confidence
 ({prediction['Confidence']:.2%}) is greater"
                       f" than limit ({confidence_limit:.2%})")
        logger.info("%s", reject_info)

    if not reject:
        logger.info("No anomalies found.")
    return reject
```

4. 选择部署以部署您的 Lambda 函数。

步骤 4：试用您的 Lambda 函数

在此步骤中，您将使用计算机上的 Python 代码，将本地图像或 Amazon S3 桶中的图像传递给您的 Lambda 函数。从本地计算机传递的图像必须小于 6291456 字节。如果图像较大，请将图像上传到 Amazon S3 桶，然后使用该图像的 Amazon S3 路径调用脚本。有关将图像文件上传到 Amazon S3 桶的更多信息，请参阅上传对象。

确保您运行代码的 AWS 区域与创建 Lambda 函数时所在的区域相同。您可以在 Lambda 控制台的函数详细信息页面的导航栏中，查看 Lambda 函数的AWS区域。

如果 AWS Lambda 函数返回超时错误，请延长 Lambda 函数的超时时间。有关更多信息，请参阅配置函数超时（控制台）。

有关从您的代码调用 Lambda 函数的更多信息，请参阅调用 AWS Lambda 函数。

试用您的 Lambda 函数

1. 执行以下操作(如果尚未这样做)：

   a. 确保使用客户端代码的用户具有 `lambda:InvokeFunction` 权限。您可以使用以下权限。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "LambdaPermission",
            "Effect": "Allow",
            "Action": "lambda:InvokeFunction",
            "Resource": "ARN for lambda function"
        }
    ]
}
```

   您可以从 Lambda 控制台中的函数概览，获取 Lambda 函数的 ARN。

   要提供访问权限，请为您的用户、组或角色添加权限：

   - AWS IAM Identity Center 中的用户和组：

     创建权限集合。按照《AWS IAM Identity Center 用户指南》中 创建权限集 的说明进行操作。

   - 通过身份提供商在 IAM 中托管的用户：

     创建适用于身份联合验证的角色。按照《IAM 用户指南》中为第三方身份提供商创建角色（联合身份验证）的说明进行操作。

   - IAM 用户：
     - 创建您的用户可以担任的角色。按照《IAM 用户指南》中为 IAM 用户创建角色的说明进行操作。
     - （不推荐使用）将策略直接附加到用户或将用户添加到用户组。按照IAM 用户指南 中向用户添加权限（控制台）中的说明进行操作。

b. 安装并配置适用于 Python 的 AWS SDK。有关更多信息，请参阅 步骤 4：设置 AWS CLI 和 AWS 软件开发工具包。

c. 启动模型，即您在步骤 1：创建 AWS Lambda 函数（控制台）的步骤 7 中指定的模型。

2. 将以下代码保存到名为 client.py 的文件中。

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

"""
Purpose: Shows how to call the anomaly detection
AWS Lambda function.
"""
from botocore.exceptions import ClientError

import argparse
import logging
import base64
import json
import boto3
from os import environ

logger = logging.getLogger(__name__)

def analyze_image(function_name, image):
    """
    Analyzes an image with an AWS Lambda function.
    :param image: The image that you want to analyze.
    :return The status and classification result for
    the image analysis.
    """

    lambda_client = boto3.client('lambda')

    lambda_payload = {}

    if image.startswith('s3://'):
        logger.info("Analyzing image from S3 bucket: %s", image)
        bucket, key = image.replace("s3://", "").split("/", 1)
        s3_object = {
            'Bucket': bucket,
            'Name': key
        }
```

```
        lambda_payload = {"S3Object": s3_object}

    # Call the lambda function with the image.
    else:
        with open(image, 'rb') as image_file:
            logger.info("Analyzing local image image: %s ", image)
            image_bytes = image_file.read()
            data = base64.b64encode(image_bytes).decode("utf8")
            lambda_payload = {"image": data, "filename": image}

    response = lambda_client.invoke(FunctionName=function_name,
                                    Payload=json.dumps(lambda_payload))
    return json.loads(response['Payload'].read().decode())


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

    parser.add_argument(
        "function", help="The name of the AWS Lambda function "
        "that you want to use to analyze the image.")
    parser.add_argument(
        "image", help="The local image that you want to analyze.")


def main():
    """
    Entrypoint for script.
    """
    try:
        logging.basicConfig(level=logging.INFO,
                            format="%(levelname)s: %(message)s")

        # Get command line arguments.
        parser = argparse.ArgumentParser(usage=argparse.SUPPRESS)
        add_arguments(parser)
        args = parser.parse_args()

        # Analyze image and display results.

        result = analyze_image(args.function, args.image)
```

```
        status = result['statusCode']

        if status == 200:
            classification = result['body']
            print(f"classification: {classification['Reject']}")
            print(f"Message: {classification['RejectMessage']}")
        else:
            print(f"Error: {result['statusCode']}")
            print(f"Message: {result['body']}")

    except ClientError as error:
        logging.error(error)
        print(error)


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

3. 运行该代码。对于命令行参数，请提供 Lambda 函数名称和要分析的本地图像的路径。例如：

```
python client.py function_name /bucket/path/image.jpg
```

如果成功，则会针对图像中发现的异常输出分类结果。如果未返回分类结果，请考虑降低您在步骤 1：创建 AWS Lambda 函数（控制台）的步骤 7 中设置的置信度值。

4. 如果已结束 Lambda 函数并且模型未被其他应用程序使用，请停止模型。下次要使用 Lambda 函数时，请记得启动模型。

在边缘设备上使用您的 Amazon Lookout for Vision 模型

您可以在由 AWS IoT Greengrass Version 2 管理的边缘设备上使用 Amazon Lookout for Vision 模型。Amazon IoT Greengrass 是一项开源物联网 (IoT) 边缘运行时和云服务。您可以使用它在自己的设备上构建、部署和管理 IoT 应用程序。有关更多信息，请参阅AWS IoT Greengrass。

您可以将已在云端完成训练的同一 Amazon Lookout for Vision 模型部署到兼容 AWS IoT Greengrass V2 的边缘设备上。然后，您可以使用所部署的模型在本地执行异常检测，如工厂车间，无需将数据持续流式传输到云端。这样，您就可以尽可能降低带宽成本，并通过实时图像分析在本地检测异常。

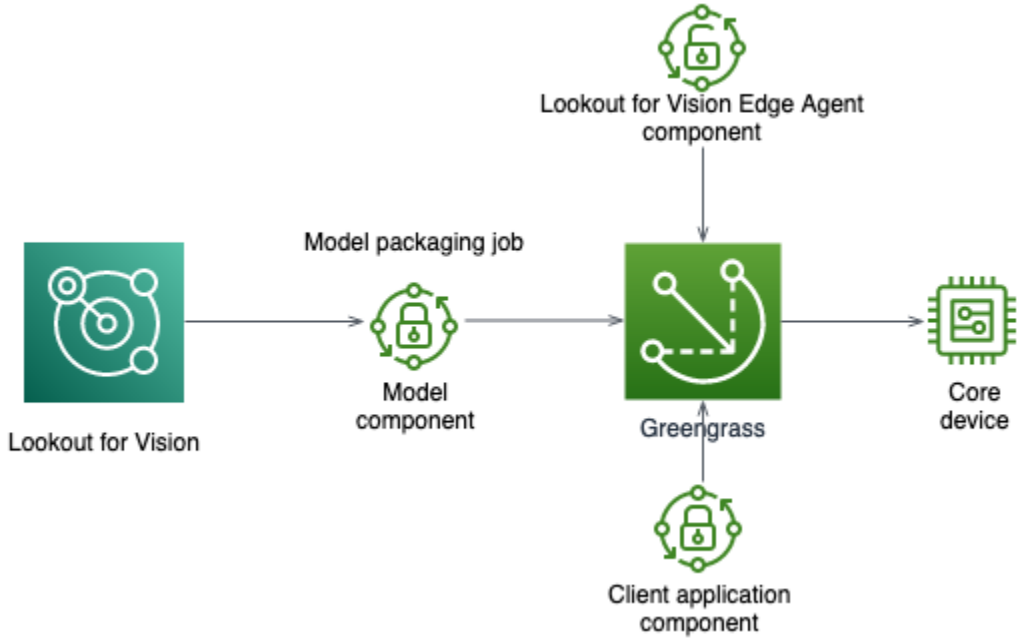
> **Tip**
>
> 在使用 AWS IoT Greengrass 部署 Lookout for Vision 模型之前，我们建议您阅读 AWS IoT Greengrass Version 2 开发者指南。有关更多信息，请参阅什么是 Amazon IoT Greengrass？。

要在 AWS IoT Greengrass V2 核心设备上使用 Lookout for Vision 模型，您需要将模型和辅助软件作为组件部署到核心设备。组件是在 Greengrass 核心设备上运行的软件模块，如 Lookout for Vision 模型。组件有两种形式。自定义组件是由您创建组件，只有您可以访问。它也称为私有组件。AWS 提供的组件是由 AWS 提供的预构建组件。它也称为公有组件。有关更多信息，请参阅 https://docs.aws.amazon.com/greengrass/v2/developerguide/public-components.html。

在核心设备上，您需要为 Lookout for Vision 模型和辅助软件部署的组件包括：

- 模型组件。一种包含 Lookout for Vision 模型的自定义组件。要创建模型组件，您应使用 Lookout for Vision 创建模型打包作业。模型打包作业会为模型创建一个组件，并使之作为自定义组件在 AWS IoT Greengrass V2 中可供使用。有关更多信息，请参阅 将您的 Amazon Lookout for Vision 模型打包。
- 客户端应用程序组件。一种由您创建的自定义组件，用于实现您的业务需求所对应的代码。例如，从组装后拍摄的图像中发现异常电路板。有关更多信息，请参阅 编写您的客户端应用程序组件。
- Amazon Lookout for Vision Edge Agent 组件。一种由 AWS 提供的组件，可以提供一个 API 以使用和管理您的模型。例如，您的客户端应用程序组件中的代码可以使用 `DetectAnomalies` API 来检测图像中的异常。Lookout for Vision Edge Agent 组件是模型组件的依赖项。部署模型组件时，它会自动安装在核心设备上。有关更多信息，请参阅 Amazon Lookout for Vision Edge Agent API 参考。

创建模型组件和客户端应用程序组件后，您可以使用 AWS IoT Greengrass V2 将组件和依赖项部署到核心设备。有关更多信息，请参阅 将您的组件部署到设备。



> **Important**
>
> 模型在核心设备上使用 `DetectAnomalies` 所做的预测，可能与使用云端托管的同一模型所做的预测不同。我们建议您在生产环境中使用模型之前，先在核心设备上测试模型。
> 为了减少设备托管模型与云托管模型之间预测结果不匹配的情况，我们建议增加训练数据集中的正常图像和异常图像数量。我们不建议重复使用现有图像来增加训练数据集的大小。

将模型和客户端应用程序组件部署到 AWS IoT Greengrass Version 2 核心设备

在 AWS IoT Greengrass Version 2 核心设备上部署 Amazon Lookout for Vision 模型和客户端应用程序组件的过程如下：

1. 使用 AWS IoT Greengrass Version 2 设置您的核心设备。
2. 通过使用 Lookout for Vision，创建模型打包作业。该作业将创建您的模型组件。
3. 编写客户端应用程序组件。该组件用于实现您的业务逻辑。
4. 通过使用 AWS IoT Greengrass V2，将模型组件和客户端应用程序组件部署到核心设备。

在将组件和依赖项部署到核心设备后，您便可以在核心设备上使用该模型。

> **Note**
>
> 您必须使用相同的 AWS 区域和 AWS 账户，来创建和部署您的 Lookout for Vision 模型和客户端应用程序组件。

AWS IoT Greengrass Version 2 核心设备要求

为了在 AWS IoT Greengrass Version 2 核心设备上使用 Amazon Lookout for Vision 模型，您的模型对核心设备有各种不同的要求。

主题

- 经过测试的设备、芯片架构和操作系统
- 核心设备内存和存储
- 必需的软件

经过测试的设备、芯片架构和操作系统

我们预计 Amazon Lookout for Vision 将在以下硬件上运行：

- CPU 架构
  - X86_64（64 位版本的 x86 指令集）
  - Aarch64（ARMv8 64 位 CPU）
- （仅限 GPU 加速推理）具有足够内存容量的 NVIDIA GPU 加速器（一个运行中的模型至少需要 6.0 GB）。

Amazon Lookout for Vision 团队已在以下设备、芯片架构和操作系统上测试 Lookout for Vision 模型。

设备

设备	操作系统	架构	Accelerator	编译器选项
jetson_xavier (NVIDIA® Jetson AGX Xavier)	Linux	Aarch64	NVIDIA	{"gpu-code": "sm_72", "trt-ver": "7.1.3", "cuda-ver": "10.2"} {"gpu-code": "sm_72", "trt-ver": "8.2.1", "cuda-ver": "10.2"}
g4dn.xlarge [配备 NVIDIA T4 Tensor Core GPU 的 EC2 实例 (G4)]	Linux	X86_64/X86-64	NVIDIA	{"gpu-code": "sm_75", "trt-ver": "7.1.3", "cuda-ver": "10.2"}
g5.xlarge [配备 NVIDIA A10G Tensor Core GPU 的 EC2 实例 (G5)]	Linux	X86_64/X86-64	NVIDIA	{"gpu-code": "sm_80", "trt-ver": "8.2.0", "cuda-ver": "11.2"}

设备	操作系统	架构	Accelerator	编译器选项
c5.2xlarge（Amazon EC2 C5 实例）	Linux	X86_64/X86-64	CPU	`{"mcpu": "core-avx2"}`

核心设备内存和存储

为了运行单个模型和 Amazon Lookout for Vision Edge Agent，您的核心设备需要满足以下内存和存储要求。您的客户端应用程序组件可能需要更多内存和存储。

- 存储：至少 1.5 GB。
- 内存：一个运行中的模型至少需要 6.0 GB。

必需的软件

核心设备需要具有以下软件。

Jetson 设备

如果您的核心设备是 Jetson 设备，则需要在核心设备上安装以下软件。

软件	支持的版本
Jetpack SDK	4.4 到 4.6.1
Python 和适用于 Lookout for Vision Edge Agent 版本 1.x 的 Python 虚拟环境	3.8 或 3.9

X86 硬件

如果您的核心设备使用 x86 硬件，则需要在核心设备上安装以下软件。

CPU 推理

软件	支持的版本
Python 和适用于 Lookout for Vision Edge Agent 版本 1.x 的 Python 虚拟环境	3.8 或 3.9

GPU 加速推理

软件版本因您使用的 NVIDIA GPU 的微架构而异。

采用 Ampere 之前微架构的 NVIDIA GPU（计算能力低于 8.0）

采用 Ampere 之前微架构的 NVIDIA GPU（计算能力低于 8.0）所需的软件。gpu-code 必须小于 sm_80。

软件	支持的版本
NVIDIA CUDA	10.2
NVIDIA TensorRT	至少 7.1.3 且低于 8.0.0
Python 和适用于 Lookout for Vision Edge Agent 版本 1.x 的 Python 虚拟环境	3.8 或 3.9

采用 Ampere 微架构的 NVIDIA GPU（计算能力 8.0）

采用 Ampere 微架构的 NVIDIA GPU（计算能力为 8.0）所需的软件。gpu-code 必须是 sm_80。

软件	支持的版本
NVIDIA CUDA	11.2
NVIDIA TensorRT	8.2.0
Python 和适用于 Lookout for Vision Edge Agent 版本 1.x 的 Python 虚拟环境	3.8 或 3.9

设置您的 AWS IoT Greengrass Version 2 核心设备

Amazon Lookout for Vision 使用 AWS IoT Greengrass Version 2，可以更简单地将模型组件、Amazon Lookout for Vision Edge Agent 组件和客户端应用程序组件部署到您的 AWS IoT Greengrass V2 核心设备。有关可以使用的设备和硬件的信息，请参阅 AWS IoT Greengrass Version 2 核心设备要求。

设置您的核心设备

使用以下信息来设置您的核心设备。

设置您的核心设备

1. 设置您的 GPU 库。如果不使用 GPU 加速推理，请勿执行此步骤。
   a. 确认您具有支持 CUDA 的 GPU。有关更多信息，请参阅验证您具有支持 CUDA 的 GPU。
   b. 通过执行以下操作之一，在您的设备上设置 CUDA、cuDNN 和 TensorRT：
      - 如果使用的是 Jetson 设备，请安装 JetPack 版本 4.4-4.6.1。有关更多信息，请参阅 JetPack 归档。
      - 如果使用的是基于 x86 的硬件，并且您的 NVIDIA GPU 微架构早于 Ampere（计算能力低于 8.0），请执行以下操作：
        1. 按照 Linux 系统 NVIDIA CUDA 安装指南中的说明，设置 CUDA 版本 10.2。
        2. 按照 NVIDIA cuDNN 文档中的说明，安装 cuDNN。
        3. 按照 NVIDIA TENSORRT 文档中的说明，设置 TensorRT（版本 7.1.3 或更高版本，但低于 8.0.0）。
      - 如果使用的是基于 x86 的硬件，并且您的 NVIDIA GPU 微架构是 Ampere（计算能力为 8.0），请执行以下操作：
        1. 按照 Linux 系统 NVIDIA CUDA 安装指南中的说明，设置 CUDA（版本 11.2）。
        2. 按照 NVIDIA cuDNN 文档中的说明，安装 cuDNN。
        3. 按照 NVIDIA TENSORRT 文档中的说明，设置 TensorRT（版本 8.2.0）。
2. 在您的核心设备上安装 AWS IoT Greengrass Version 2 核心软件。有关更多信息，请参阅 AWS IoT Greengrass Version 2 开发者指南 中的安装 Amazon IoT Greengrass 核心软件。
3. 要从存储模型的 Amazon S3 桶中执行读取操作，请将权限附加到您在 AWS IoT Greengrass Version 2 设置期间创建的 IAM 角色（令牌交换角色）。有关更多信息，请参阅允许访问 S3 桶中的组件构件。

4. 在命令提示符处，输入以下命令，将 Python 和 Python 虚拟环境安装到核心设备上。

```
sudo apt install python3.8 python3-venv python3.8-venv
```

5. 使用以下命令，将 Greengrass 用户添加到视频组。这将允许 Greengrass 部署的组件访问 GPU：

```
sudo usermod -a -G video ggc_user
```

6. （可选）如果要从其他用户调用 Lookout for Vision Edge Agent API，请将该用户添加到 ggc_group 中。这将允许用户通过 Unix 域套接字与 Lookout for Vision Edge Agent 进行通信：

```
sudo usermod -a -G ggc_group $(whoami)
```

将您的 Amazon Lookout for Vision 模型打包

模型打包作业会将 Amazon Lookout for Vision 模型打包为模型组件。

要创建模型打包作业，您需要选择要打包的模型，并且为该作业创建的模型组件提供设置。您只能将已成功训练的模型打包。

您可以使用 Lookout for Vision 控制台或 AWS SDK 来创建模型打包作业。您还可以获取与您创建的模型打包作业相关的信息。有关更多信息，请参阅 获取有关模型打包作业的信息。您可以使用 AWS IoT Greengrass V2 控制台或 AWS SDK，将组件部署到 AWS IoT Greengrass Version 2 核心设备。

主题

- 包设置
- 将您的模型打包（控制台）
- 将您的模型打包 (SDK)
- 获取有关模型打包作业的信息

包设置

使用以下信息可决定模型打包作业的包设置。

要创建模型打包作业，请参阅 将您的模型打包（控制台） 或 将您的模型打包 (SDK)。

主题

- 目标硬件
- 组件设置

目标硬件

您可以为模型选择目标设备或目标平台，但不能同时选择两者。有关更多信息，请参阅 经过测试的设备、芯片架构和操作系统。

目标设备

模型的目标设备，如 NVIDIA® Jetson AGX Xavier。您不需要指定编译器选项。

目标平台

Amazon Lookout for Vision 支持以下平台配置：

- X86_64（64 位版本的 x86 指令集）和 Aarch64（ARMv8 64 位 CPU）架构。
- Linux 操作系统。
- 使用 NVIDIA 或 CPU 加速器进行推理。

您需要为目标平台指定正确的编译器选项。

编译器选项

编译器选项使您能够为自己的 AWS IoT Greengrass Version 2 核心设备指定目标平台。目前，您可以指定以下编译器选项。

NVIDIA 加速器

- `gpu-code`：指定运行模型组件的核心设备的 GPU 代码。
- `trt-ver`：以 x.y.z. 格式指定 TensorRT 版本。
- `cuda-ver`：以 x.y 格式指定 CUDA 版本。

CPU 加速器

- （可选）`mcpu`：指定指令集。例如 `core-avx2`。如果不提供一个值，则 Lookout for Vision 将使用值 `core-avx2`。

您应以 JSON 格式指定这些选项。例如：

```
{"gpu-code": "sm_75", "trt-ver": "7.1.3", "cuda-ver": "10.2"}
```

有关更多示例，请参阅 经过测试的设备、芯片架构和操作系统：

组件设置

模型打包作业会创建一个包含您的模型的模型组件。该作业将创建一些构件，供 AWS IoT Greengrass V2 用于将模型组件部署到核心设备。

您不能使用与现有组件具有相同组件名称和组件版本的模型组件。

组件名称

Lookout for Vision 在模型打包期间创建的模型组件的名称。您指定的组件名称会显示在 AWS IoT Greengrass V2 控制台中。在为客户端应用程序组件创建的配方中，您应使用该组件名称。有关更多信息，请参阅 创建客户端应用程序组件。

组件描述

（可选）对模型组件的描述。

组件版本

模型组件的版本号。您可以接受默认版本号，也可以选择自己的版本号。版本号必须遵循语义版本号系统：major.minor.patch。例如，版本 1.0.0 表示组件的第一个主要版本。有关更多信息，请参阅语义版本控制 2.0.0。如果不提供一个值，则 Lookout for Vision 将使用您的模型的版本号为您生成一个版本。

组件位置

您希望模型打包作业用来保存模型组件构件的 Amazon S3 位置。Amazon S3 桶所在的亚马逊云科技区域及账户必须与您使用 AWS IoT Greengrass Version 2 所在的区域和账户相同。要创建 Amazon S3 桶，请参阅创建桶。

标签

您可以通过使用标签，识别、整理、搜索和筛选您的组件。每个标签都是由用户定义的键和值组成的标签。当模型打包作业在 Greengrass 中创建模型组件时，这些标签会附加到模型组件上。一个组件就是

一个 Amazon IoT Greengrass V2 资源。这些标签不会附加到任何 Lookout for Vision 资源，如您的模型。有关更多信息，请参阅标记亚马逊云科技资源。

将您的模型打包（控制台）

通过使用 Amazon Lookout for Vision 控制台，您可以创建模型打包作业。

有关包设置的信息，请参阅包设置。

将模型打包（控制台）

1. 创建一个 Amazon S3 桶，或重复使用现有的桶，供 Lookout for Vision 用来存储打包作业的构件（模型组件）。
2. 打开 Amazon Lookout for Vision 控制台，网址为 https://console.aws.amazon.com/lookoutvision/。
3. 选择开始使用。
4. 在左侧导航窗格中，选择项目。
5. 在项目部分，选择包含要打包的模型的项目。
6. 在左侧导航窗格中，于项目名称下选择边缘模型包。
7. 在模型包装任务部分，选择创建模型打包任务。
8. 输入包的设置。有关更多信息，请参阅 包设置。
9. 选择创建模型打包任务。
10. 等待打包作业完成。当作业状态为成功时，表示该作业已完成。
11. 选择模型包装任务部分中的打包作业。
12. 选择继续在 Greengrass 中部署，以便继续在 AWS IoT Greengrass Version 2 中部署您的模型组件。有关更多信息，请参阅 将您的组件部署到设备。

将您的模型打包 (SDK)

您应通过创建模型打包作业，将模型打包为模型组件。要创建模型打包作业，您应调用 StartModelPackagingJob API。该作业可能需要一段时间才能完成。要查看当前的状态，请调用 DescribeModelPackagingJob 并检查其响应中的 `Status` 字段。

有关包设置的信息，请参阅包设置。

以下过程展示了如何使用 AWS CLI 启动打包作业。您可以将模型打包以用于目标平台或目标设备。有关 Java 示例代码，请参阅 StartModelPackagingJob。

将您的模型打包 (SDK)

1. 安装并配置 AWS CLI 和 AWS SDK（如果尚未如此）。有关更多信息，请参阅 步骤 4：设置 AWS CLI 和 AWS 软件开发工具包。
2. 确保您拥有启动模型打包作业所需的正确权限。有关更多信息，请参阅 StartModelPackagingJob。
3. 使用以下 CLI 命令，将您的模型打包以用于目标设备或目标平台。

   Target platform

   以下 CLI 命令展示了如何将模型打包，以用于采用 NVIDIA 加速器的目标平台。

   更改以下值：

   - `project_name` 更改为包含要打包的模型的项目的名称。
   - `model_version` 更改为要打包的模型版本。
   - （可选）`description` 更改为对模型打包作业的描述。
   - `architecture` 更改为运行模型组件的 AWS IoT Greengrass Version 2 核心设备的架构（`ARM64` 或 `X86_64`）。
   - `gpu_code` 更改为运行模型组件的核心设备的 GPU 代码。
   - `trt_ver` 更改为在核心设备上安装的 TensorRT 版本。
   - `cuda_ver` 更改为在核心设备上安装的 CUDA 版本。
   - `component_name` 更改为要在 AWS IoT Greengrass V2 上创建的模型组件的名称。
   - （可选）`component_version` 更改为打包作业创建的模型组件的版本。采用格式 `major.minor.patch`。例如，1.0.0 表示组件的第一个主要版本。
   - `bucket` 更改为打包作业用来存储模型组件构件的 Amazon S3 桶。
   - `prefix` 更改为 Amazon S3 桶中供打包作业用来存储模型组件构件的位置。
   - （可选）`component_description` 更改为对模型组件的描述。
   - （可选）`tag_key1` 和 `tag_key2` 更改为附加到模型组件的标签的键。
   - （可选）`tag_value1` 和 `tag_value2` 更改为附加到模型组件的标签的键值。

```
aws lookoutvision start-model-packaging-job \
```

```
  --project-name project_name \
  --model-version model_version \
  --description="description" \
  --configuration
 "Greengrass={TargetPlatform={Os='LINUX',Arch='architecture',Accelerator='NVIDIA'},Compi
code\": \"gpu_code\", \"trt-ver\": \"trt_ver\", \"cuda-ver\":
 \"cuda_ver\"}',S3OutputLocation={Bucket='bucket',Prefix='prefix'},ComponentName='Compon
 {Key='tag_key2',Value='tag_value2'}]}" \
  --profile lookoutvision-access
```

例如：

```
aws lookoutvision start-model-packaging-job \
  --project-name test-project-01 \
  --model-version 1 \
  --description="Model Packaging Job for G4 Instance using TargetPlatform
 Option" \
  --configuration
 "Greengrass={TargetPlatform={Os='LINUX',Arch='X86_64',Accelerator='NVIDIA'},CompilerOpt
code\": \"sm_75\", \"trt-ver\": \"7.1.3\", \"cuda-ver\":
 \"10.2\"}',S3OutputLocation={Bucket='bucket',Prefix='test-project-01/
folder'},ComponentName='SampleComponentNameX86TargetPlatform',ComponentVersion='0.1.0',C
 is my component',Tags=[{Key='modelKey0',Value='modelValue'},
 {Key='modelKey1',Value='modelValue'}]}" \
  --profile lookoutvision-access
```

Target Device

使用以下 CLI 命令，将模型打包以用于目标设备。

更改以下值：

- `project_name` 更改为包含要打包的模型的项目的名称。
- `model_version` 更改为要打包的模型版本。
- （可选）`description` 更改为对模型打包作业的描述。
- `component_name` 更改为要在 AWS IoT Greengrass V2 上创建的模型组件的名称。
- （可选）`component_version` 更改为打包作业创建的模型组件的版本。采用格式 `major.minor.patch`。例如，1.0.0 表示组件的第一个主要版本。
- `bucket` 更改为打包作业用来存储模型组件构件的 Amazon S3 桶。
- `prefix` 更改为 Amazon S3 桶中供打包作业用来存储模型组件构件的位置。

- （可选）component_description 更改为对模型组件的描述。
- （可选）tag_key1 和 tag_key2 更改为附加到模型组件的标签的键。
- （可选）tag_value1 和 tag_value2 更改为附加到模型组件的标签的键值。

```
aws lookoutvision start-model-packaging-job \
  --project-name project_name \
  --model-version model_version \
  --description="description" \
  --configuration
 "Greengrass={TargetDevice='jetson_xavier',S3OutputLocation={Bucket='bucket',Prefix='pre
 {Key='tag_key2',Value='tag_value2'}]}" \
  --profile lookoutvision-access
```

例如：

```
aws lookoutvision start-model-packaging-job \
  --project-name project_01 \
  --model-version 1 \
  --description="description" \
  --configuration
 "Greengrass={TargetDevice='jetson_xavier',S3OutputLocation={Bucket='bucket',Prefix='com
 model component',Tags=[{Key='tag_key1',Value='tag_value1'},
 {Key='tag_key2',Value='tag_value2'}]}" \
  --profile lookoutvision-access
```

4. 记下响应中的 JobName 值。您在下一个步骤中需要用到它。例如：

```
{
    "JobName": "6bcfd0ff-90c3-4463-9a89-6b4be3daf972"
}
```

5. 使用 DescribeModelPackagingJob 获取作业的当前状态。更改以下项：

- project_name 更改为要使用的项目的名称。
- job_name 更改为您在上一步中记录的作业名称。

```
aws lookoutvision describe-model-packaging-job \
    --project-name project_name \
```

```
    --job-name job_name \
    --profile lookoutvision-access
```

如果 Status 的值为 SUCCEEDED，表示模型打包作业已完成。如果是其他值，请稍等片刻，然后重试。

6. 继续使用 AWS IoT Greengrass V2 进行部署。有关更多信息，请参阅 将您的组件部署到设备。

获取有关模型打包作业的信息

您可以使用 Amazon Lookout for Vision 控制台或 AWS SDK，获取与您创建的模型打包作业相关的信息。

主题

- 获取模型打包作业信息（控制台）
- 获取模型打包作业信息 (SDK)

获取模型打包作业信息（控制台）

获取模型打包作业信息（控制台）

1. 打开 Amazon Lookout for Vision 控制台，网址为 https://console.aws.amazon.com/lookoutvision/。
2. 选择开始使用。
3. 在左侧导航窗格中，选择项目。
4. 在项目部分，选择包含要查看的模型打包作业的项目。
5. 在左侧导航窗格中，于项目名称下选择边缘模型包。
6. 在模型打包作业部分，选择要查看的模型打包作业。此时将显示该模型打包作业的详细信息页面。

获取模型打包作业信息 (SDK)

您可以使用 AWS SDK 列出项目中的模型打包作业，并获取与特定模型打包作业相关的信息。

列出模型打包作业

您可以通过调用 ListModelPackagingJobs API，列出项目中的模型打包作业。响应中包含一个 ModelPackagingJobMetadata 对象列表，其中提供了有关每个模型打包作业的信息。此外还包括一个分页标记，如果列表不完整，您可以用它来获取下一组结果。

列出您的模型打包作业

1. 安装并配置 AWS CLI 和 AWS SDK（如果尚未如此）。有关更多信息，请参阅 步骤 4：设置 AWS CLI 和 AWS 软件开发工具包。
2. 使用以下 CLI 命令。将 `project_name` 的值更改为要使用的项目名称。

```
aws lookoutvision list-model-packaging-jobs \
  --project-name project_name \
  --profile lookoutvision-access
```

描述模型打包作业

使用 DescribeModelPackagingJob API，获取有关模型打包作业的信息。它的响应是一个 ModelPackagingDescription 对象，其中包含作业的当前状态和其他信息。

描述包

1. 安装并配置 AWS CLI 和 AWS SDK（如果尚未如此）。有关更多信息，请参阅 步骤 4：设置 AWS CLI 和 AWS 软件开发工具包。
2. 使用以下 CLI 命令。更改以下项：

   - `project_name` 更改为要使用的项目的名称。
   - `job_name` 更改为作业的名称。当您调用 StartModelPackagingJob 时，将会得到作业名称 (`JobName`)。

```
aws lookoutvision describe-model-packaging-job \
    --project-name project_name \
    --job-name job_name \
    --profile lookoutvision-access
```

编写您的客户端应用程序组件

客户端应用程序组件是由您编写的自定义 AWS IoT Greengrass Version 2 组件。它用于实现在 AWS IoT Greengrass Version 2 核心设备上使用 Amazon Lookout for Vision 模型所需的业务逻辑。

要访问模型，您的客户端应用程序组件需要使用 Lookout for Vision Edge Agent 组件。Lookout for Vision Edge Agent 组件提供了一个 API，通过该 API，可使用模型分析图像并管理核心设备上的模型。

Lookout for Vision Edge Agent API 使用 gRPC 实现，后者是一种用于进行远程过程调用的协议。有关更多信息，请参阅 gRPC。要编写代码，您可以使用 gRPC 支持的任何语言。我们提供了 Python 示例代码。有关更多信息，请参阅 在您的客户端应用程序组件中使用模型。

> **Note**
>
> Lookout for Vision Edge Agent 组件是您部署的模型组件的依赖项。当您将模型组件部署到核心设备时，它会自动部署到核心设备上。

要编写客户端应用程序组件，您应执行以下操作。

1. 设置您的环境，以便使用 gRPC 并安装第三方库。
2. 编写代码以使用模型。
3. 将代码作为自定义组件部署到核心设备。

如需通过一个示例客户端应用程序组件，展示如何在自定义 GStreamer 管道中执行异常检测，请参阅 https://github.com/awslabs/aws-greengrass-labs-lookoutvision-gstreamer。

设置环境

要编写客户端代码，您的开发环境需要远程连接到您已部署 Amazon Lookout for Vision 模型组件和依赖项的 AWS IoT Greengrass Version 2 核心设备。或者，您可以在核心设备上编写代码。有关更多信息，请参阅 Amazon IoT Greengrass 开发工具和开发 Amazon IoT Greengrass 组件。

您的客户端代码应使用 gRPC 客户端来访问 Amazon Lookout for Vision Edge Agent。本部分介绍如何使用 gRPC 设置开发环境，并安装 `DetectAnomalies` 示例代码所需的第三方依赖项。

编写完客户端代码后，您可以创建自定义组件并将该自定义组件部署到自己的边缘设备。有关更多信息，请参阅 创建客户端应用程序组件。

主题

- 设置 gRPC
- 添加第三方依赖项

设置 gRPC

在开发环境中，您需要一个 gRPC 客户端，用来在代码中调用 Lookout for Vision Edge Agent API。为此，您应使用 Lookout for Vision Edge Agent 的 `.proto` 服务定义文件，创建一个 gRPC 存根。

> **Note**
>
> 您也可以从 Lookout for Vision Edge Agent 应用程序捆绑包中获取该服务定义文件。在将 Lookout for Vision Edge Agent 组件作为模型组件的依赖项进行安装时，将会安装该应用程序捆绑包。该应用程序捆绑包位于 `/greengrass/v2/packages/artifacts-unarchived/aws.iot.lookoutvision.EdgeAgent/`*`edge_agent_version`*`/lookoutvision_edge_agent`。`edge_agent_version` 应替换为您正在使用的 Lookout for Vision Edge Agent 的版本。要获取该应用程序捆绑包，您需要将 Lookout for Vision Edge Agent 部署到核心设备。

设置 gRPC

1. 下载压缩文件 proto.zip。该压缩文件中包含 .proto 服务定义文件 (`edge-agent.proto`)。
2. 对内容进行解压缩。
3. 打开命令提示符，导航至包含 `edge-agent.proto` 的文件夹。
4. 使用以下命令生成 Python 客户端接口。

   ```
   %%bash
   python3 -m pip install grpcio
   python3 -m pip install grpcio-tools
   python3 -m grpc_tools.protoc --proto_path=. --python_out=. --grpc_python_out=.
    edge-agent.proto
   ```

   如果这些命令成功执行，就会在工作目录中创建 `edge_agent_pb2.py` 存根和 `edge_agent_pb2_grpc.py`。
5. 编写使用您的模型的客户端代码。有关更多信息，请参阅 在您的客户端应用程序组件中使用模型。

添加第三方依赖项

DetectAnomalies 示例代码使用 Pillow 库来处理图像。有关更多信息，请参阅 使用图像字节检测异常。

使用以下命令可安装 Pillow 库。

```
python3 -m pip install Pillow
```

在您的客户端应用程序组件中使用模型

从客户端应用程序组件使用模型时，步骤与使用云端托管的模型类似。

1. 开始运行模型。
2. 检测图像中的异常。
3. 停止模型（如果不再需要）。

Amazon Lookout for Vision Edge Agent 提供了 API 来启动模型、检测图像中的异常和停止模型。您还可以使用该 API 列出设备上的模型，并获取与已部署的模型相关的信息。有关更多信息，请参阅 Amazon Lookout for Vision Edge Agent API 参考。

您可以通过检查 gRPC 状态代码获取错误信息。有关更多信息，请参阅 获取错误信息。

要编写代码，您可以使用 gRPC 支持的任何语言。我们提供了 Python 示例代码。

主题

- 在客户端应用程序组件中使用存根
- 启动模型
- 检测异常
- 停止模型
- 列出设备上的模型
- 描述模型
- 获取错误信息

在客户端应用程序组件中使用存根

使用以下代码，可设置通过 Lookout for Vision Edge Agent 来访问模型。

```
import grpc
from edge_agent_pb2_grpc import EdgeAgentStub
import edge_agent_pb2 as pb2

# Creating stub.
 with grpc.insecure_channel("unix:///tmp/aws.iot.lookoutvision.EdgeAgent.sock") as
 channel:
    stub = EdgeAgentStub(channel)
    # Add additional code that works with Edge Agent in this block to prevent resources
 leakage
```

启动模型

您应通过调用 StartModel API 来启动模型。模型可能需要一段时间才能启动。您可以通过调用 DescribeModel 来检查当前的状态。如果 `status` 字段的值为 Running，表示模型正在运行。

示例代码

component_name 应替换为您的模型组件的名称。

```
import time

import grpc
from edge_agent_pb2_grpc import EdgeAgentStub
import edge_agent_pb2 as pb2


model_component_name = "component_name"


def start_model_if_needed(stub, model_name):
    # Starting model if needed.
    while True:
        model_description_response =
 stub.DescribeModel(pb2.DescribeModelRequest(model_component=model_name))
        print(f"DescribeModel() returned {model_description_response}")
        if model_description_response.model_description.status == pb2.RUNNING:
            print("Model is already running.")
            break
        elif model_description_response.model_description.status == pb2.STOPPED:
            print("Starting the model.")
            stub.StartModel(pb2.StartModelRequest(model_component=model_name))
            continue
```

```
        elif model_description_response.model_description.status == pb2.FAILED:
            raise Exception(f"model {model_name} failed to start")
        print(f"Waiting for model to start.")
        if model_description_response.model_description.status != pb2.STARTING:
            break
        time.sleep(1.0)


# Creating stub.
with grpc.insecure_channel("unix:///tmp/aws.iot.lookoutvision.EdgeAgent.sock") as
 channel:
    stub = EdgeAgentStub(channel)
    start_model_if_needed(stub, model_component_name)
```

检测异常

您应使用 DetectAnomalies API 来检测图像中的异常。

DetectAnomalies 操作期望图像位图以 RGB888 压缩格式进行传递。第一个字节代表红色通道，第二个字节代表绿色通道，第三个字节代表蓝色通道。如果您以其他格式（如 BGR）提供图像，则 DetectAnomalies 所做的预测会不正确。

默认情况下，OpenCV 对图像位图使用 BGR 格式。如果要使用 OpenCV 捕获图像以供 DetectAnomalies 分析，则必须先将图像转换为 RGB888 格式，然后才能将其传递给 DetectAnomalies。

在宽度和高度尺寸上，您提供给 DetectAnomalies 的图像必须与用于训练模型的图像相同。

使用图像字节检测异常

您可以通过以图像字节的形式提供图像，来检测图像中的异常。在下面的示例中，图像字节是从存储在本地文件系统中的图像中获取的。

sample.jpg 应替换为要分析的图像文件的名称。*component_name* 应替换为您的模型组件的名称。

```
import time

from PIL import Image
import grpc
from edge_agent_pb2_grpc import EdgeAgentStub
import edge_agent_pb2 as pb2
```

```
model_component_name = "component_name"

....
# Detecting anomalies.
def detect_anomalies(stub, model_name, image_path):
    image = Image.open(image_path)
    image = image.convert("RGB")
    detect_anomalies_response = stub.DetectAnomalies(
        pb2.DetectAnomaliesRequest(
            model_component=model_name,
            bitmap=pb2.Bitmap(
                width=image.size[0],
                height=image.size[1],
                byte_data=bytes(image.tobytes())
            )
        )
    )
    print(f"Image is anomalous -
 {detect_anomalies_response.detect_anomaly_result.is_anomalous}")
    return detect_anomalies_response.detect_anomaly_result


# Creating stub.
with grpc.insecure_channel("unix:///tmp/aws.iot.lookoutvision.EdgeAgent.sock") as
 channel:
    stub = EdgeAgentStub(channel)
    start_model_if_needed(stub, model_component_name)
    detect_anomalies(stub, model_component_name, "sample.jpg")
```

通过使用共享内存段检测异常

您可以通过在 POSIX 共享内存段中以图像字节的形式提供图像，来检测图像中的异常。为获得最佳性能，我们建议对 DetectAnomalies 请求使用共享内存。有关更多信息，请参阅 DetectAnomalies。

停止模型

如果您不再使用模型，请使用 StopModel API 停止运行模型。

```
stop_model_response = stub.StopModel(
    pb2.StopModelRequest(
         model_component=model_component_name
    )
```

```
)
print(f"New status of the model is {stop_model_response.status}")
```

列出设备上的模型

您可以使用 the section called “ListModels” API 列出部署到设备的模型。

```
models_list_response = stub.ListModels(
    pb2.ListModelsRequest()
)
for model in models_list_response.models:
    print(f"Model Details {model}")
```

描述模型

您可以通过调用 DescribeModel API，获取与部署到设备的模型相关的信息。使用 `DescribeModel` 有助于获取模型的当前状态。例如，您需要先知道模型是否正在运行，然后才能调用 `DetectAnomalies`。有关示例代码，请参阅 启动模型。

获取错误信息

gRPC 状态码用于报告 API 结果。

您可以通过捕获 `RpcError` 异常来获取错误信息，如以下示例所示。有关错误状态码的信息，请参阅 API 的参考主题。

```
# Error handling.
try:
    stub.DetectAnomalies(detect_anomalies_request)
except grpc.RpcError as e:
    print(f"Error code: {e.code()}, Status: {e.details()}")
```

创建客户端应用程序组件

生成 gRPC 存根并准备好客户端应用程序代码后，您就可以创建客户端应用程序组件。您创建的组件是一种自定义组件，需要使用 AWS IoT Greengrass V2 将其部署到 AWS IoT Greengrass Version 2 核心设备。您创建的配方用于描述您的自定义组件。该配方中包含所有也需部署的依赖项。在此例中，您应指定在将您的 Amazon Lookout for Vision 模型打包中创建的模型组件。有关组件配方的更多信息，请参阅 AWS IoT Greengrass Version 2 组件配方参考。

本主题的相关过程展示了如何通过配方文件创建客户端应用程序组件，并将其作为 AWS IoT Greengrass V2 自定义组件发布。您可以使用 AWS IoT Greengrass V2 控制台或 AWS SDK 来发布组件。

有关创建自定义组件的详细信息，请参阅 AWS IoT Greengrass V2 文档中的以下内容。

- 在您的设备上开发和测试组件
- 创建 AWS IoT Greengrass 组件
- 发布组件以部署到您的核心设备

主题

- 发布客户端应用程序组件所需的 IAM 权限
- 创建配方
- 发布客户端应用程序组件（控制台）
- 发布客户端应用程序组件 (SDK)

发布客户端应用程序组件所需的 IAM 权限

要创建和发布客户端应用程序组件，您需要以下 IAM 权限：

- `greengrass:CreateComponentVersion`
- `greengrass:DescribeComponent`
- `s3:PutObject`

创建配方

在此过程中，您将为一个简单的客户端应用程序组件创建配方。`lookoutvision_edge_agent_example.py` 中的代码可以列出部署到设备的模型，在您将组件部署到核心设备后，它会自动运行。要查看输出，请在部署组件后查看组件日志。有关更多信息，请参阅 将您的组件部署到设备。准备就绪后，应使用此过程为用于实现业务逻辑的代码创建配方。

您应将配方创建为 JSON 或 YAML 格式的文件。您还要将客户端应用程序代码上载到 Amazon S3 桶。

创建客户端应用程序组件配方

1. 创建 gRPC 存根文件（如果尚未如此）。有关更多信息，请参阅 设置 gRPC。

2. 将以下代码保存到名为 lookoutvision_edge_agent_example.py 的文件中

```
import grpc
from edge_agent_pb2_grpc import EdgeAgentStub
import edge_agent_pb2 as pb2

# Creating stub.
with grpc.insecure_channel("unix:///tmp/aws.iot.lookoutvision.EdgeAgent.sock") as
 channel:
    stub = EdgeAgentStub(channel)
    # Add additional code that works with Edge Agent in this block to prevent
 resources leakage

    models_list_response = stub.ListModels(
        pb2.ListModelsRequest()
    )
    for model in models_list_response.models:
        print(f"Model Details {model}")
```

3. 创建一个 Amazon S3 桶（或使用现有的桶），以存储您的客户端应用程序组件的源文件。该桶必须位于您的 AWS 账户中，并且它所在的 AWS 区域必须与您使用 AWS IoT Greengrass Version 2 和 Amazon Lookout for Vision 时所在的区域相同。
4. 将 lookoutvision_edge_agent_example.py、edge_agent_pb2_grpc.py and edge_agent_pb2.py 文件上传到您在上一步中创建的 Amazon S3 桶。记下每个文件的 Amazon S3 路径。您在设置 gRPC 中创建了 edge_agent_pb2_grpc.py 和 edge_agent_pb2.py。
5. 在编辑器中，创建以下 JSON 或 YAML 配方文件。

   - model_component 更改为您的模型组件的名称。有关更多信息，请参阅 组件设置。
   - 将 URI 条目更改为 lookoutvision_edge_agent_example.py、edge_agent_pb2_grpc.py 和 edge_agent_pb2.py 的 S3 路径。

   JSON

```
{
  "RecipeFormatVersion": "2020-01-25",
  "ComponentName": "com.lookoutvision.EdgeAgentPythonExample",
  "ComponentVersion": "1.0.0",
  "ComponentType": "aws.greengrass.generic",
```

```
  "ComponentDescription": "Lookout for Vision Edge Agent Sample Application",
  "ComponentPublisher": "Sample App Publisher",
  "ComponentDependencies": {
    "model_component": {
      "VersionRequirement": ">=1.0.0",
      "DependencyType": "HARD"
    }
  },
  "Manifests": [
    {
      "Platform": {
        "os": "linux"
      },
      "Lifecycle": {
        "install": "pip3 install grpcio grpcio-tools protobuf Pillow",
        "run": {
          "script": "python3 {artifacts:path}/
lookoutvision_edge_agent_example.py"
        }
      },
      "Artifacts": [
        {
          "Uri": "S3 path to lookoutvision_edge_agent_example.py"
        },
        {
          "Uri": "S3 path to edge_agent_pb2_grpc.py"
        },
        {
          "Uri": "S3 path to edge_agent_pb2.py"
        }
      ]
    }
  ],
  "Lifecycle": {}
}
```

YAML

```
---
RecipeFormatVersion: 2020-01-25
ComponentName: com.lookoutvison.EdgeAgentPythonExample
ComponentVersion: 1.0.0
ComponentDescription: Lookout for Vision Edge Agent Sample Application
```

```
ComponentPublisher: Sample App Publisher
ComponentDependencies:
  model_component:
    VersionRequirement: '>=1.0.0'
    DependencyType: HARD
Manifests:
  - Platform:
      os: linux
    Lifecycle:
      install: |-
        pip3 install grpcio
        pip3 install grpcio-tools
        pip3 install protobuf
        pip3 install Pillow
      run:
        script: |-
          python3 {artifacts:path}/lookout_vision_agent_example.py
    Artifacts:
      - URI: S3 path to lookoutvision_edge_agent_example.py
      - URI: S3 path to edge_agent_pb2_grpc.py
      - URI: S3 path to edge_agent_pb2.py
```

6. 将该 JSON 或 YAML 文件保存到您的计算机。
7. 通过执行以下操作之一创建客户端应用程序组件：

   - 如果要使用 AWS IoT Greengrass 控制台，请执行以下操作：发布客户端应用程序组件（控制台）。
   - 如果要使用 AWS SDK，请执行以下操作：发布客户端应用程序组件 (SDK)。

发布客户端应用程序组件（控制台）

您可以使用 AWS IoT Greengrass V2 控制台发布客户端应用程序组件。

发布客户端应用程序组件

1. 通过执行以下操作：创建配方，为客户端应用程序组件创建配方（如果尚未如此）。
2. 打开 AWS IoT Greengrass 控制台，网址为 https://console.aws.amazon.com/iot/。
3. 在左侧导航窗格中，于 Greengrass 下选择组件。
4. 在我的组件下，选择创建组件。

5. 在创建组件页面上，如果要使用 JSON 格式的配方，请选择以 JSON 格式输入配方。如果要使用 YAML 格式的配方，请选择以 YAML 格式输入配方。
6. 在配方下，将现有配方替换为您在创建配方中创建的 JSON 或 YAML 配方。
7. 选择创建组件。
8. 接下来，部署您的客户端应用程序组件。

发布客户端应用程序组件 (SDK)

您可以使用 CreateComponentVersion API 来发布客户端应用程序组件。

发布客户端应用程序组件 (SDK)

1. 通过执行以下操作：创建配方，为客户端应用程序组件创建配方（如果尚未如此）。
2. 在命令提示符处，输入以下命令以创建客户端应用程序组件。`recipe-file` 应替换为您在创建配方中创建的配方文件的名称。

   ```
   aws greengrassv2 create-component-version --inline-recipe fileb://recipe-file
   ```

   记下响应中组件的 ARN。您在下一个步骤中需要用到它。
3. 使用以下命令获取客户端应用程序组件的状态。`component-arn` 应替换为您在上一步中记下的 ARN。如果 `componentState` 的值为 `DEPLOYABLE`，表示客户端应用程序组件已准备就绪。

   ```
   aws greengrassv2 describe-component --arn component-arn
   ```
4. 接下来，部署您的客户端应用程序组件。

将您的组件部署到设备

要将模型组件和客户端应用程序组件部署到 AWS IoT Greengrass Version 2 核心设备，您应使用 AWS IoT Greengrass V2 控制台或使用 CreateDeployment API。有关更多信息，请参阅 AWS IoT Greengrass Version 2 开发者指南 中的创建部署。有关对部署到核心设备的组件进行更新的信息，请参阅修订部署。

主题

- 部署组件所需的 IAM 权限
- 部署您的组件（控制台）

- 部署组件 (SDK)

部署组件所需的 IAM 权限

要使用 AWS IoT Greengrass V2 来部署组件，您需要以下权限：

- `greengrass:ListComponents`
- `greengrass:ListComponentVersions`
- `greengrass:ListCoreDevices`
- `greengrass:CreateDeployment`
- `greengrass:GetDeployment`
- `greengrass:ListDeployments`

`CreateDeployment` 和 `GetDeployment` 具有依赖操作。有关更多信息，请参阅 Amazon IoT Greengrass V2 定义的操作。

有关更改 IAM 权限的更多信息，请参阅更改用户的权限。

部署您的组件（控制台）

使用以下过程可将客户端应用程序组件部署到核心设备。客户端应用程序依赖于模型组件（后者又依赖于 Lookout for Vision Edge Agent）。部署客户端应用程序组件时，还会启动模型组件和 Lookout for Vision Edge Agent 的部署。

> **Note**
>
> 您可以将组件添加到现有部署中。您也可以将组件部署到事物组。

要运行此过程，必须事先配置 AWS IoT Greengrass V2 核心设备。有关更多信息，请参阅 设置您的 AWS IoT Greengrass Version 2 核心设备。

将您的组件部署到设备

1. 打开 AWS IoT Greengrass 控制台，网址为 https://console.aws.amazon.com/iot/。
2. 在左侧导航窗格中，于 Greengrass 下选择部署。

3. 在部署下，选择创建。
4. 在指定目标页面上，执行以下操作：
   1. 在部署信息下，输入或修改部署的友好名称。
   2. 在部署目标下，选择核心设备并输入目标名称。
   3. 选择 Next（下一步）。
5. 在选择用户页面上，执行以下操作：
   1. 在我的组件下，选择您的客户端应用程序组件的名称（`com.lookoutvison.EdgeAgentPythonExample`）。
   2. 选择 Next（下一步）。
6. 在配置组件页面上，保留当前配置并选择下一步。
7. 在配置高级设置页面上，保留当前的设置，然后选择下一步。
8. 在查看页面上，选择部署以开始部署您的组件。

检查部署状态（控制台）

您可以通过 AWS IoT Greengrass V2 控制台查看部署的状态。如果您的客户端应用程序组件使用the section called “创建客户端应用程序组件”中的示例配方和代码，请在部署完成后查看客户端应用程序组件日志。如果部署成功，则日志中将包含部署到组件的 Lookout for Vision 模型的列表。

有关使用 AWS SDK 检查部署状态的信息，请参阅检查部署状态。

检查部署状态

1. 打开 AWS IoT Greengrass 控制台，网址为 https://console.aws.amazon.com/iot/。
2. 在左侧导航窗格中，选择核心设备。
3. 在 Greengrass 核心设备下，选择您的核心设备。
4. 选择部署选项卡以查看当前的部署状态。
5. 部署成功后（状态为已完成），在核心设备上打开一个终端窗口，通过以下位置查看客户端应用程序组件日志：`/greengrass/v2/logs/com.lookoutvison.EdgeAgentPythonExample.log`。如果您的部署使用示例配方和代码，则日志中将包含 `lookoutvision_edge_agent_example.py` 的输出。例如：

```
Model Details model_component:"ModelComponent"
```

部署组件 (SDK)

使用下面的过程，可以将客户端应用程序组件、模型组件和 Amazon Lookout for Vision Edge Agent 组件部署到您的核心设备。

1. 创建一个 deployment.json 文件来为定义组件的部署配置。此文件应类似于以下示例。

```
{
  "targetArn":"targetArn",
  "components": {
    "com.lookoutvison.EdgeAgentPythonExample": {
      "componentVersion": "1.0.0",
      "configurationUpdate": {
      }
    }
  }
}
```

   - 在 targetArn 字段中，按以下格式将 *targetArn* 替换为部署目标的事物或事物组的 Amazon 资源名称 (ARN)：
     - 事物：arn:aws:iot:*region*:*account-id*:thing/*thingName*
     - 事物组：arn:aws:iot:*region*:*account-id:*thinggroup/*thingGroupName*
2. 检查部署目标是否有要修改的现有部署。执行以下操作：

   a. 运行以下命令，列出部署目标的部署。targetArn 应替换为目标 AWS IoT 事物或事物组的 Amazon 资源名称 (ARN)。要获取当前亚马逊云科技区域中的事物 ARN，请使用命令 aws iot list-things。

```
aws greengrassv2 list-deployments --target-arn targetArn
```

   响应中包含目标最新部署的列表。如果响应为空，则表示目标没有现有部署，您可以跳到步骤 3。否则，请复制响应中的 deploymentId，以便在下一步中使用。

   b. 运行以下命令，获取部署的详细信息。这些详细信息包括元数据、组件和作业配置。deploymentId 应替换为上一步中的 ID。

```
aws greengrassv2 get-deployment --deployment-id deploymentId
```

c. 从上一个命令的响应中，将以下任何键值对复制到 deployment.json 中。您可以为新部署更改这些值。

- `deploymentName`：部署的名称。
- `components`：部署的组件。要卸载组件，请将其从该对象中移除。
- `deploymentPolicies`：部署的策略。
- `tags`：部署的标签。

3. 运行以下命令以在设备上部署组件。记下响应中 `deploymentId` 的值。

```
aws greengrassv2 create-deployment \
    --cli-input-json file://path/to/deployment.json
```

4. 运行以下命令以获取部署的状态。`deployment-id` 应更改为您在上一步中记录的值。如果 `deploymentStatus` 的值为 `COMPLETED`，表示已成功完成部署。

```
aws greengrassv2 get-deployment --deployment-id deployment-id
```

5. 部署成功后，在核心设备上打开一个终端窗口，通过以下位置查看客户端应用程序组件日志：`/greengrass/v2/logs/com.lookoutvison.EdgeAgentPythonExample.log`。如果您的部署使用示例配方和代码，则日志中将包含 `lookoutvision_edge_agent_example.py` 的输出。例如：

```
Model Details model_component:"ModelComponent"
```

Amazon Lookout for Vision Edge Agent API 参考

本部分是 Amazon Lookout for Vision Edge Agent 的 API 参考。

使用模型检测异常

您可以使用 DetectAnomalies API，通过在 AWS IoT Greengrass Version 2 核心设备上运行的模型来检测图像中的异常。

获取模型信息

以下 API 用于获取与部署到核心设备的模型相关的信息。

- ListModels

- DescribeModel

运行模型

以下 API 用于启动和停止部署到核心设备的 Amazon Lookout for Vision 模型。

- StartModel
- StopModel

DetectAnomalies

检测所提供图像中的异常。

DetectAnomalies 的响应中包括一个布尔预测值（预测图像中包含一个异常或多个异常），以及一个对所做预测的置信度值。如果模型是分割模型，则响应中包括以下内容：

- 一个掩码图像，其中会分别用一种独特的颜色覆盖每种异常类型。您可以使 DetectAnomalies 将掩码图像存储在共享内存中，也可以用图像字节的形式返回掩码。
- 异常类型覆盖的图像区域百分比。
- 掩码图像上异常类型的十六进制颜色。

> **Note**
>
> 与 DetectAnomalies 一起使用的模型必须处于运行状态。您可以通过调用 DescribeModel 来获取当前的状态。要开始运行模型，请参阅 StartModel。

DetectAnomalies 支持交错 RGB888 格式的压缩位图（图像）。第一个字节代表红色通道，第二个字节代表绿色通道，第三个字节代表蓝色通道。如果您以其他格式（如 BGR）提供图像，则 DetectAnomalies 所做的预测会不正确。

默认情况下，OpenCV 对图像位图使用 BGR 格式。如果要使用 OpenCV 捕获图像以供 DetectAnomalies 分析，则必须先将图像转换为 RGB888 格式，然后才能将其传递给 DetectAnomalies。

支持的最小图像尺寸为 64 x 64 像素。支持的最大图像尺寸为 4096 x 4096 像素。

您可以通过 protobuf 消息或通过共享内存段发送图像。如果将大尺寸图像序列化到 protobuf 消息中，可能会显著增加调用 DetectAnomalies 时的延迟。为了尽可能降低延迟，我们建议您使用共享内存。

```
rpc DetectAnomalies(DetectAnomaliesRequest) returns (DetectAnomaliesResponse);
```

DetectAnomaliesRequest

DetectAnomalies 的输入参数。

```
message Bitmap {
  int32 width = 1;
  int32 height = 2;
  oneof data {
    bytes byte_data = 3;
    SharedMemoryHandle shared_memory_handle = 4;
  }
}
```

```
message SharedMemoryHandle {
  string name = 1;
  uint64 size = 2;
  uint64 offset = 3;
}
```

```
message AnomalyMaskParams {
SharedMemoryHandle shared_memory_handle = 2;
}
```

```
message DetectAnomaliesRequest {
string model_component = 1;
Bitmap bitmap = 2;
AnomalyMaskParams anomaly_mask_params = 3;
}
```

Bitmap

您要使用 DetectAnomalies 分析的图像。

width

以像素为单位的图像宽度。

height

以像素为单位的图像高度。

byte_data

在 protobuf 消息中传递的图像字节。

shared_memory_handle

在共享内存段中传递的图像字节。

SharedMemoryHandle

代表一个 POSIX 共享内存段。

name

POSIX 内存段的名称。有关创建共享内存的信息，请参阅 shm_open。

size

从偏移量开始的图像缓冲区大小（以字节为单位）。

offset

从共享内存段开始，到图像缓冲区开头的偏移量（以字节为单位）。

AnomalyMaskParams

用于输出异常掩码的参数。（分割模型）。

shared_memory_handle

如果未提供 shared_memory_handle，则包含掩码的图像字节。

DetectAnomaliesRequest

model_component

包含要使用的模型的 AWS IoT Greengrass V2 组件的名称。

bitmap

您要使用 DetectAnomalies 分析的图像。

anomaly_mask_params

用于输出掩码的可选参数。（分割模型）。

DetectAnomaliesResponse

DetectAnomalies 的响应。

```
message DetectAnomalyResult {
bool is_anomalous = 1;
float confidence = 2;
Bitmap anomaly_mask = 3;
repeated Anomaly anomalies = 4;
float anomaly_score = 5;
float anomaly_threshold = 6;
}
```

```
message Anomaly {
string name = 1;
PixelAnomaly pixel_anomaly = 2;
```

```
message PixelAnomaly {
float total_percentage_area = 1;
string hex_color = 2;
}
```

```
message DetectAnomaliesResponse {
  DetectAnomalyResult detect_anomaly_result = 1;
}
```

异常

表示在图像上发现的异常。（分割模型）。

name

在图像中发现的异常类型的名称。name 对应训练数据集中的异常类型。该服务会自动将背景异常类型插入到 DetectAnomalies 的响应中。

pixel_anomaly

与覆盖异常类型的像素掩码相关的信息。

PixelAnomaly

与覆盖异常类型的像素掩码相关的信息。（分割模型）。

total_percentage_area

异常类型覆盖的图像区域百分比。

hex_color

一个十六进制颜色值，表示图像上的异常类型。颜色对应训练数据集中使用的异常类型的颜色。

DetectAnomalyResult

is_anomalous

指示图像是否包含异常。true 表示图像包含异常。false 表示图像正常。

confidence

DetectAnomalies 对预测准确度的置信度。confidence 是介于 0 和 1 之间的浮点值。

anomaly_mask

如果未提供 shared_memory_handle，则包含掩码的图像字节。（分割模型）。

异常

在输入图像中发现的 0 个或多个异常的列表。（分割模型）。

anomaly_score

一个数字，用于量化所预测的图像异常与无异常图像的偏差程度。anomaly_score 是一个浮点值，范围介于 0.0（与正常图像的偏差最小）到 1.0（与正常图像的偏差最大）之间。即使预测图像为正常，Amazon Lookout for Vision 也会返回一个 anomaly_score 值。

anomaly_threshold

一个数字（浮点数），用于确定图像的预测分类什么情况下为正常或异常。如果图像的 anomaly_score 等于或大于 anomaly_threshold 的值，则视为异常。如果 anomaly_score 值低于 anomaly_threshold，则表示图像正常。您在训练模型时，Amazon Lookout for Vision 会计算供模型使用的 anomaly_threshold 值。您无法设置或更改 anomaly_threshold 的值。

状态代码

代码	数字	描述
OK	0	DetectAnomalies 成功做出了预测
UNKNOWN	2	发生了未知错误。
INVALID_ARGUMENT	3	一个或多个输入参数无效。有关更多详细信息，请查看错误消息。
NOT_FOUND	5	未找到具有指定名称的模型。
RESOURCE_EXHAUSTED	8	没有足够的资源来执行此操作。例如，Lookout for Vision Edge Agent 无法跟上对 DetectAnomalies 的调用速度。有关更多详细信息，请查看错误消息。
FAILED_PRECONDITION	9	对未处于 RUNNING 状态的模型调用了 DetectAnomalies 。
INTERNAL	13	出现内部错误。

DescribeModel

描述部署到 AWS IoT Greengrass Version 2 核心设备的 Amazon Lookout for Vision 模型。

```
rpc DescribeModel(DescribeModelRequest) returns (DescribeModelResponse);
```

DescribeModelRequest

```
message DescribeModelRequest {
  string model_component = 1;
}
```

model_component

包含要描述的模型的 AWS IoT Greengrass V2 组件的名称。

DescribeModelResponse

```
message ModelDescription {
  string model_component = 1;
  string lookout_vision_model_arn = 2;
  ModelStatus status = 3;
  string status_message = 4;
}
```

```
message DescribeModelResponse {
  ModelDescription model_description = 1;
}
```

ModelDescription

model_component

包含 Amazon Lookout for Vision 模型的 AWS IoT Greengrass Version 2 组件的名称。

lookout_vision_model_arn

用于生成 AWS IoT Greengrass V2 组件的 Amazon Lookout for Vision 模型的 Amazon 资源名称 (ARN)。

状态

模型的当前状态。有关更多信息，请参阅 ModelStatus。

status_message

模型的状态消息。

状态代码

代码	数字	描述
OK	0	调用已成功。
UNKNOWN	2	发生了未知错误。
INVALID_ARGUMENT	3	一个或多个输入参数无效。有关更多详细信息，请查看错误消息。
NOT_FOUND	5	未找到具有所提供名称的模型。
INTERNAL	13	出现内部错误。

ListModels

列出部署到 AWS IoT Greengrass Version 2 核心设备的模型。

```
rpc ListModels(ListModelsRequest) returns (ListModelsResponse);
```

ListModelsRequest

```
message ListModelsRequest {}
```

ListModelsResponse

```


message ModelMetadata {
  string model_component = 1;
  string lookout_vision_model_arn = 2;
  ModelStatus status = 3;
  string status_message = 4;
```

```
}
```

```
message ListModelsResponse {
  repeated ModelMetadata models = 1;
}
```

ModelMetadata

model_component

包含 Amazon Lookout for Vision 模型的 AWS IoT Greengrass Version 2 组件的名称。

lookout_vision_model_arn

用于生成 AWS IoT Greengrass V2 组件的 Amazon Lookout for Vision 模型的 Amazon 资源名称 (ARN)。

状态

模型的当前状态。有关更多信息，请参阅 ModelStatus。

status_message

模型的状态消息。

状态代码

代码	数字	描述
OK	0	调用已成功。
UNKNOWN	2	发生了未知错误。
INTERNAL	13	出现内部错误。

StartModel

启动在 AWS IoT Greengrass Version 2 核心设备上运行的模型。模型可能需要一段时间才能开始运行。要检查当前状态，请调用 DescribeModel。如果 `Status` 字段为 `RUNNING`，表示模型正在运行。

可同时运行的模型数量取决于核心设备的硬件规格。

```
rpc StartModel(StartModelRequest) returns (StartModelResponse);
```

StartModelRequest

```
message StartModelRequest {
  string model_component = 1;
}
```

model_component

包含要启动的模型的 AWS IoT Greengrass Version 2 组件的名称。

StartModelResponse

```
message StartModelResponse {
  ModelStatus status = 1;
}
```

状态

模型的当前状态。如果调用成功，则响应为 STARTING。有关更多信息，请参阅 ModelStatus。

状态代码

代码	数字	描述
OK	0	模型正在启动。
UNKNOWN	2	发生了未知错误。
INVALID_ARGUMENT	3	一个或多个输入参数无效。有关更多详细信息，请查看错误消息。
NOT_FOUND	5	未找到具有所提供名称的模型。

代码	数字	描述
RESOURCE_EXHAUSTED	8	没有足够的资源来执行此操作。例如，内存不足，无法加载模型。有关更多详细信息，请查看错误消息。
FAILED_PRECONDITION	9	对未处于 STOPPED 或 FAILED 状态的模型调用了此方法。
INTERNAL	13	出现内部错误。

StopModel

停止在 AWS IoT Greengrass Version 2 核心设备上运行的模型。`StopModel` 会在模型停止后返回响应。如果响应中的 `Status` 字段为 `STOPPED`，则表示模型已成功停止。

```
  rpc StopModel(StopModelRequest) returns (StopModelResponse);
```

StopModelRequest

```
message StopModelRequest {
  string model_component = 1;
}
```

model_component

包含要停止的模型的 AWS IoT Greengrass Version 2 组件的名称。

StopModelResponse

```
message StopModelResponse {
  ModelStatus status = 1;
}
```

状态

模型的当前状态。如果调用成功，则响应为 STOPPED。有关更多信息，请参阅 ModelStatus。

状态代码

代码	数字	描述
OK	0	模型正在停止。
UNKNOWN	2	发生了未知错误。
INVALID_ARGUMENT	3	一个或多个输入参数无效。有关更多详细信息，请查看错误消息。
NOT_FOUND	5	未找到具有所提供名称的模型。
FAILED_PRECONDITION	9	对未处于 RUNNING 状态的模型调用了此方法。
INTERNAL	13	出现内部错误。

ModelStatus

部署到 AWS IoT Greengrass Version 2 核心设备的模型的状态。要获取当前状态，请调用 DescribeModel。

```
enum ModelStatus {
  STOPPED = 0;
  STARTING = 1;
  RUNNING = 2;
  FAILED = 3;
  STOPPING = 4;
}
```

使用 Amazon Lookout for Vision 控制面板

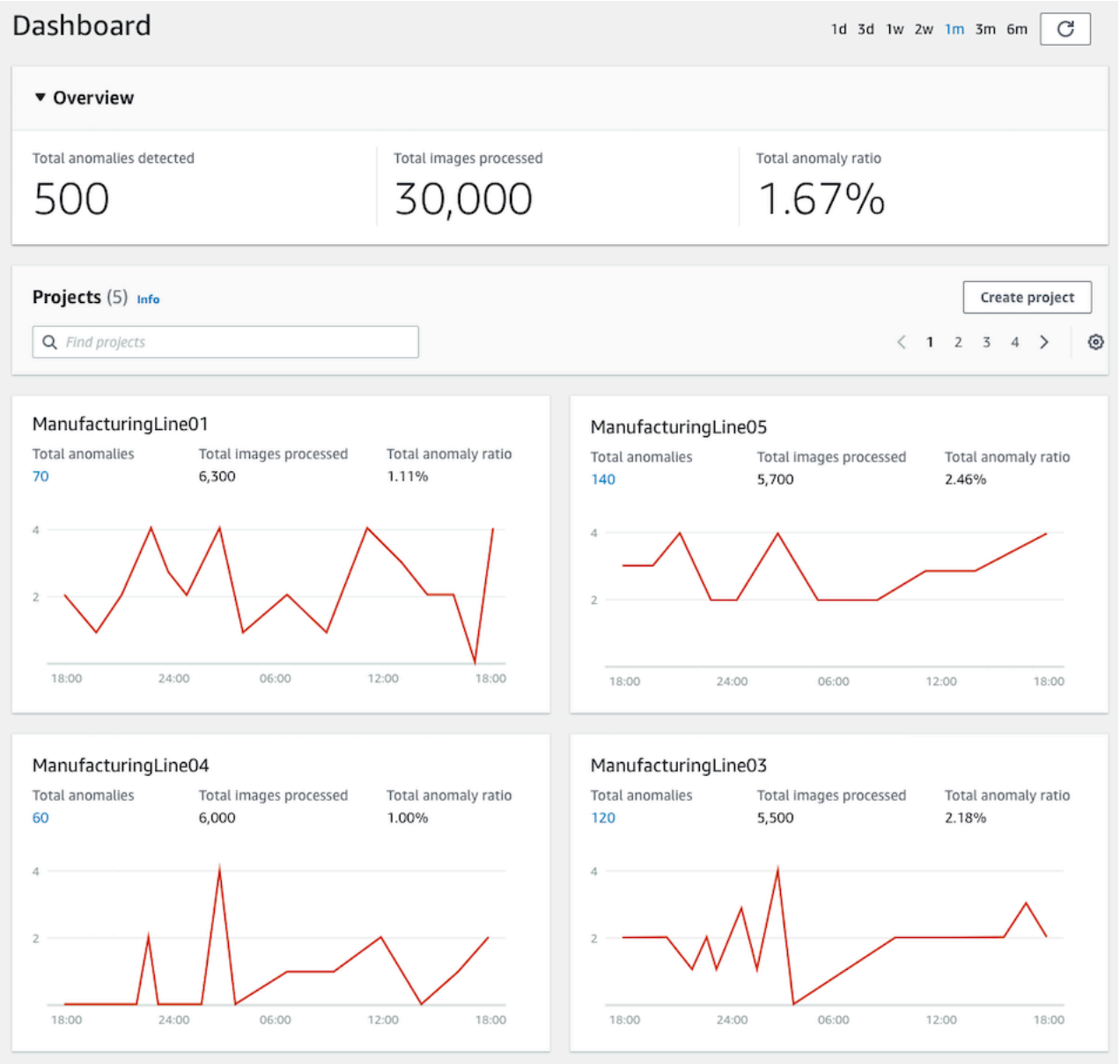
控制面板可以提供 Amazon Lookout for Vision 项目的指标概览，如上周检测到的异常总数。借助控制面板，您可以获取自己所有项目的概览，以及各个项目的概览。您可以选择所显示指标的时间线。您还可以使用控制面板创建新项目。

概览部分显示项目总数、图像总数，以及您的所有项目检测的图像总数。

项目部分显示各个项目的以下概览信息：

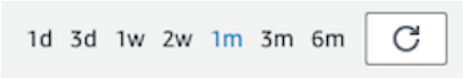
- 检测到的异常总数。
- 已处理的图像总数。
- 总异常比（即检测到异常的图像的百分比）。
- 通过一个图形显示所选时间范围内的异常检测情况。

您还可以获取与项目有关的进一步信息。

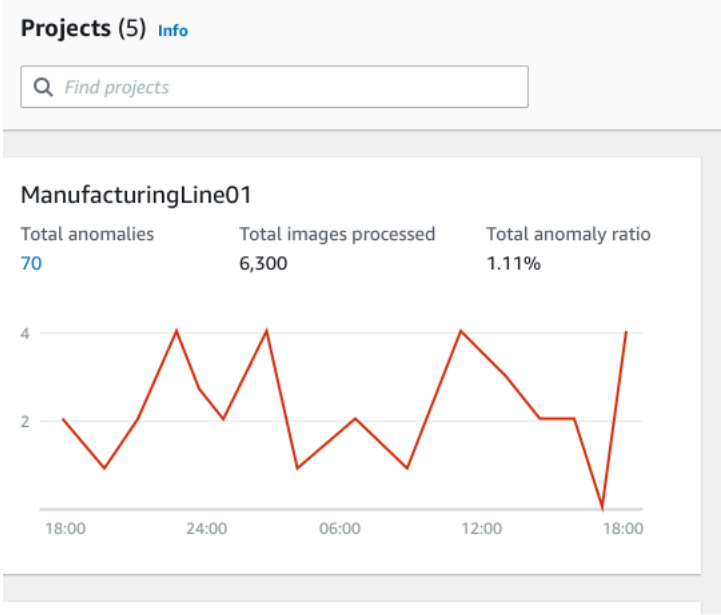


使用控制面板

1. 打开 Amazon Lookout for Vision 控制台，网址为 https://console.aws.amazon.com/lookoutvision/。
2. 选择开始使用。
3. 在左侧导航窗格中，选择控制面板。
4. 要查看特定时间范围的指标，请执行以下操作：
   a. 在控制面板右上角选择时间范围。
   b. 选择刷新按钮，显示新时间线下的控制面板。



5. 要获取与项目有关的进一步详细信息，请在项目部分选择项目名称（例如，ManufacturingLine01）。



6. 要创建项目，请在项目部分选择创建项目。

管理您的 Amazon Lookout for Vision 资源

您可以使用控制台或 AWS SDK 来管理您的 Amazon Lookout for Vision 资源。Amazon Lookout for Vision 具有以下资源：

- 项目
- 数据集
- 模型
- 试用检测

> **Note**
>
> 您无法删除试用检测任务。此外，您也无法使用 AWS SDK 管理试用检测。

主题

- 查看您的项目
- 删除项目
- 查看您的数据集
- 向您的数据集中添加图像
- 从数据集中移除图像
- 删除数据集
- 从项目中导出数据集 (SDK)
- 查看您的模型
- 删除模型
- 标记模型
- 查看您的试用检测任务

查看您的项目

您可以从控制台或使用 AWS SDK，获取 Amazon Lookout for Vision 项目列表以及有关各个项目的信息。

> **Note**
>
> 项目列表具有最终一致性。如果创建或删除项目，您可能需要稍等片刻，然后项目列表才会达到最新状态。

查看您的项目（控制台）

执行以下过程中的步骤，可以在控制台中查看您的项目。

查看您的项目

1. 打开 Amazon Lookout for Vision 控制台，网址为 https://console.aws.amazon.com/lookoutvision/。
2. 选择开始使用。
3. 在左侧导航窗格中，选择项目。随后将显示项目视图。
4. 选择一个项目名称，查看该项目的详细信息。

查看您的项目 (SDK)

一个项目管理着单个使用场景的数据集和模型。例如，检测机器零件中存在的异常。以下示例将调用 `ListProjects` 来获取您的项目列表。

查看您的项目 (SDK)

1. 安装并配置 AWS CLI 和 AWS SDK（如果尚未如此）。有关更多信息，请参阅 步骤 4：设置 AWS CLI 和 AWS 软件开发工具包。
2. 使用以下示例代码查看您的项目。

   CLI

   使用 `list-projects` 命令列出您账户中的项目。

   ```
   aws lookoutvision list-projects \
     --profile lookoutvision-access
   ```

   使用 `describe-project` 命令获取与项目有关的信息。

将 project-name 的值更改为要描述的项目的名称。

```
aws lookoutvision describe-project --project-name project_name \
  --profile lookoutvision-access
```

Python

此代码取自 AWS 文档 SDK 示例 GitHub 存储库。请在此处查看完整示例。

```
    @staticmethod
    def list_projects(lookoutvision_client):
        """
        Lists information about the projects that are in in your AWS account
        and in the current AWS Region.

        :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
```

Java V2

此代码取自 AWS 文档 SDK 示例 GitHub 存储库。请在此处查看完整示例。

```
/**
 * Lists the Amazon Lookout for Vision projects in the current AWS account and
 AWS
 * Region.
 *
 * @param lfvClient   An Amazon Lookout for Vision client.
 * @param projectName The name of the project that you want to create.
 * @return List<ProjectMetadata> Metadata for each project.
 */
public static List<ProjectMetadata> listProjects(LookoutVisionClient lfvClient)
                throws LookoutVisionException {

        logger.log(Level.INFO, "Getting projects:");
        ListProjectsRequest listProjectsRequest = ListProjectsRequest.builder()
                        .maxResults(100)
                        .build();

        List<ProjectMetadata> projectMetadata = new ArrayList<>();

        ListProjectsIterable projects =
 lfvClient.listProjectsPaginator(listProjectsRequest);
```

```
        projects.stream().flatMap(r -> r.projects().stream())
                        .forEach(project -> {
                                projectMetadata.add(project);
                                logger.log(Level.INFO, project.projectName());
                        });

        logger.log(Level.INFO, "Finished getting projects.");

        return projectMetadata;

}
```

删除项目

您可以从控制台的项目视图页面中删除项目，或者使用 DeleteProject 操作来删除项目。

项目数据集引用的图像不会被删除。

删除项目（控制台）

通过以下过程可删除项目。如果使用控制台操作过程，可以为您删除关联的模型版本和数据集。

删除项目

1. 打开 Amazon Lookout for Vision 控制台，网址为 https://console.aws.amazon.com/lookoutvision/。
2. 选择开始使用。
3. 在左侧导航窗格中，选择项目。
4. 在项目页面上，选择要删除的项目。
5. 在页面的顶部，选择 Delete (删除)。
6. 在删除对话框中，输入 delete 以确认要删除该项目。
7. 如有必要，请选择删除所有关联的数据集和模型。
8. 选择 Delete project (删除项目)。

删除项目 (SDK)

您可以调用 DeleteProject 并提供要删除的项目的名称，从而删除 Amazon Lookout for Vision 项目。

在删除项目之前，必须先删除项目中的所有模型。有关更多信息，请参阅 删除模型 (SDK)。您还必须删除与模型关联的数据集。有关更多信息，请参阅 删除数据集。

项目可能需要几分钟才能删除。在此期间，项目状态为 DELETING。如果随后对 DeleteProject 的调用不包括您删除的项目，则表示该项目已删除。

删除项目 (SDK)

1. 安装并配置 AWS CLI 和 AWS SDK（如果尚未如此）。有关更多信息，请参阅 步骤 4：设置 AWS CLI 和 AWS 软件开发工具包。
2. 使用以下代码删除项目。

   AWS CLI

   将 project-name 的值更改为要删除的项目的名称。

```
aws lookoutvision delete-project --project-name project_name \
  --profile lookoutvision-access
```

   Python

   此代码取自 AWS 文档 SDK 示例 GitHub 存储库。请在此处查看完整示例。

```
    @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
```

Java V2

此代码取自 AWS 文档 SDK 示例 GitHub 存储库。请在此处查看完整示例。

```
/**
 * Deletes an Amazon Lookout for Vision project.
 *
 * @param lfvClient   An Amazon Lookout for Vision client.
 * @param projectName The name of the project that you want to create.
 * @return String The ARN of the deleted project.
 */
public static String deleteProject(LookoutVisionClient lfvClient, String
 projectName)
                throws LookoutVisionException {

        logger.log(Level.INFO, "Deleting project: {0}", projectName);

        DeleteProjectRequest deleteProjectRequest =
 DeleteProjectRequest.builder()
                        .projectName(projectName)
                        .build();

        DeleteProjectResponse response =
 lfvClient.deleteProject(deleteProjectRequest);

        logger.log(Level.INFO, "Deleted project: {0} ARN: {1}",
                        new Object[] { projectName, response.projectArn() });

        return response.projectArn();

}
```

查看您的数据集

一个项目可以有单个数据集，用于训练和测试模型。或者，您可以使用单独的训练数据集和测试数据集。您可以使用控制台查看自己的数据集。您还可以使用 DescribeDataset 操作来获取与数据集（训练或测试）有关的信息。

查看项目中的数据集（控制台）

执行以下过程中的步骤，可以在控制台中查看项目的数据集。

查看您的数据集（控制台）

1. 打开 Amazon Lookout for Vision 控制台，网址为 https://console.aws.amazon.com/lookoutvision/。
2. 选择开始使用。
3. 在左侧导航窗格中，选择项目。
4. 在项目页面上，选择包含您要查看的数据集的项目。
5. 在左侧导航窗格中，选择数据集以查看数据集详细信息。如果有训练数据集和测试数据集，则每个数据集各显示一个选项卡。

查看项目中的数据集 (SDK)

您可以使用 `DescribeDataset` 操作，获取与项目关联的训练或测试数据集的信息。

查看您的数据集 (SDK)

1. 安装并配置 AWS CLI 和 AWS SDK（如果尚未如此）。有关更多信息，请参阅 步骤 4：设置 AWS CLI 和 AWS 软件开发工具包。
2. 使用以下示例代码查看数据集。

   CLI

   更改以下值：

   - `project-name` 更改为包含要查看的模型的项目的名称。
   - `dataset-type` 更改为要查看的数据集类型（`train` 或 `test`）。

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

   Python

   此代码取自 AWS 文档 SDK 示例 GitHub 存储库。请在此处查看完整示例。

```
    @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.")
```

Java V2

此代码取自 AWS 文档 SDK 示例 GitHub 存储库。请在此处查看完整示例。

```
/**
 * Gets the description for a Amazon Lookout for Vision dataset.
 *
```

```
 * @param lfvClient   An Amazon Lookout for Vision client.
 * @param projectName The name of the project in which you want to describe a
 *                    dataset.
 * @param datasetType The type of the dataset that you want to describe (train
 *                    or test).
 * @return DatasetDescription A description of the dataset.
 */
public static DatasetDescription describeDataset(LookoutVisionClient lfvClient,
                String projectName,
                String datasetType) throws LookoutVisionException {

        logger.log(Level.INFO, "Describing {0} dataset for project {1}",
                        new Object[] { datasetType, projectName });

        DescribeDatasetRequest describeDatasetRequest =
 DescribeDatasetRequest.builder()
                        .projectName(projectName)
                        .datasetType(datasetType)
                        .build();

        DescribeDatasetResponse describeDatasetResponse =
 lfvClient.describeDataset(describeDatasetRequest);
        DatasetDescription datasetDescription =
 describeDatasetResponse.datasetDescription();

        logger.log(Level.INFO, "Project: {0}\n"
                        + "Created: {1}\n"
                        + "Type: {2}\n"
                        + "Total: {3}\n"
                        + "Labeled: {4}\n"
                        + "Normal: {5}\n"
                        + "Anomalous: {6}\n",
                        new Object[] {
                                        datasetDescription.projectName(),
                                        datasetDescription.creationTimestamp(),
                                        datasetDescription.datasetType(),

 datasetDescription.imageStats().total().toString(),

 datasetDescription.imageStats().labeled().toString(),

 datasetDescription.imageStats().normal().toString(),

 datasetDescription.imageStats().anomaly().toString(),
```

```
                        });

        return datasetDescription;

}
```

向您的数据集中添加图像

创建数据集后，您可能会希望向数据集中添加更多图像。例如，如果模型评估表明模型性能不佳，则可以通过添加更多图像来提高模型的质量。如果您创建了测试数据集，添加更多图像可以提高模型性能指标的准确度。

更新数据集后重新训练您的模型。

主题

- 添加更多图像
- 添加更多图像 (SDK)

添加更多图像

您可以从本地计算机上传图像，从而向您的数据集中添加更多图像。要使用 SDK 添加更多已标注的图像，请使用 UpdateDatasetEntries 操作。

向数据集中添加更多图像（控制台）

1. 选择操作，然后选择要向其添加图像的数据集。
2. 选择要上传到数据集的图像。可以从本地计算机拖动或选择要上传的图像。一次最多可以上传 30 张图像。
3. 选择上传图像。
4. 选择 Save changes（保存更改）。

添加完更多图像后，您需要标注它们，以便它们可用来训练模型。有关更多信息，请参阅 图像分类（控制台）。

添加更多图像 (SDK)

要使用 SDK 添加更多已标注的图像，请使用 UpdateDatasetEntries 操作。您需要提供一个清单文件，其中包含要添加的图像。您还可以在清单文件中 JSON 行的 `source-ref` 字段指定图像，从而更新现有图像。有关更多信息，请参阅 创建清单文件。

向数据集中添加更多图像 (SDK)

1. 安装并配置 AWS CLI 和 AWS SDK（如果尚未如此）。有关更多信息，请参阅 步骤 4：设置 AWS CLI 和 AWS 软件开发工具包。
2. 使用以下示例代码向数据集中添加更多图像。

   CLI

   更改以下值：

   - `project-name` 更改为包含要更新的数据集的项目的名称。
   - `dataset-type` 更改为要更新的数据集类型（`train` 或 `test`）。
   - `changes` 更改为包含数据集更新的清单文件的位置。

```
aws lookoutvision update-dataset-entries\
    --project-name project\
    --dataset-type train or test\
    --changes fileb://manifest file \
    --profile lookoutvision-access
```

   Python

   此代码取自 AWS 文档 SDK 示例 GitHub 存储库。请在此处查看完整示例。

```
    @staticmethod
    def update_dataset_entries(lookoutvision_client, project_name, dataset_type,
updates_file):
        """
        Adds dataset entries to an Amazon Lookout for Vision dataset.
        :param lookoutvision_client: The Amazon Rekognition Custom Labels Boto3
client.
        :param project_name: The project that contains the dataset that you want
to update.
```

```
        :param dataset_type: The type of the dataset that you want to update
 (train or test).
        :param updates_file: The manifest file of JSON Lines that contains the
 updates.
        """

        try:
            status = ""
            status_message = ""
            manifest_file = ""

            # Update dataset entries
            logger.info(f"""Updating {dataset_type} dataset for project
 {project_name}
with entries from {updates_file}.""")

            with open(updates_file) as f:
                manifest_file = f.read()

            lookoutvision_client.update_dataset_entries(
                ProjectName=project_name,
                DatasetType=dataset_type,
                Changes=manifest_file,
            )

            finished = False
            while finished == False:

                dataset =
 lookoutvision_client.describe_dataset(ProjectName=project_name,

 DatasetType=dataset_type)

                status = dataset['DatasetDescription']['Status']
                status_message = dataset['DatasetDescription']['StatusMessage']

                if status == "UPDATE_IN_PROGRESS":
                    logger.info(
                        (f"Updating {dataset_type} dataset for project
 {project_name}."))
                    time.sleep(5)
                    continue

                if status == "UPDATE_FAILED_ROLLBACK_IN_PROGRESS":
```

```
                    logger.info(
                        (f"Update failed, rolling back {dataset_type} dataset
for project {project_name}."))
                    time.sleep(5)
                    continue

                if status == "UPDATE_COMPLETE":
                    logger.info(
                        f"Dataset updated: {status} : {status_message} :
{dataset_type} dataset for project {project_name}.")
                    finished = True
                    continue

                if status == "UPDATE_FAILED_ROLLBACK_COMPLETE":
                    logger.info(
                        f"Rollback complated after update failure: {status} :
{status_message} : {dataset_type} dataset for project {project_name}.")
                    finished = True
                    continue

                logger.exception(
                    f"Failed. Unexpected state for dataset update: {status} :
{status_message} : {dataset_type} dataset for project {project_name}.")
                raise Exception(
                    f"Failed. Unexpected state for dataset update: {status} :
{status_message} :{dataset_type} dataset for project {project_name}.")

            logger.info(f"Added entries to dataset.")

            return status, status_message

        except ClientError as err:
            logger.exception(
                f"Couldn't update dataset: {err.response['Error']['Message']}")
            raise
```

Java V2

此代码取自 AWS 文档 SDK 示例 GitHub 存储库。请在此处查看完整示例。

```
/**
 * Updates an Amazon Lookout for Vision dataset from a manifest file.
 * Returns after Lookout for Vision updates the dataset.
```

```
 *
 * @param lfvClient    An Amazon Lookout for Vision client.
 * @param projectName  The name of the project in which you want to update a
 *                     dataset.
 * @param datasetType  The type of the dataset that you want to update (train or
 *                     test).
 * @param manifestFile The name and location of a local manifest file that you
 want to
 * use to update the dataset.
 * @return DatasetStatus The status of the updated dataset.
 */

public static DatasetStatus updateDatasetEntries(LookoutVisionClient lfvClient,
 String projectName,
                String datasetType, String updateFile) throws
 FileNotFoundException, LookoutVisionException,
                InterruptedException {

        logger.log(Level.INFO, "Updating {0} dataset for project {1}",
                        new Object[] { datasetType, projectName });

        InputStream sourceStream = new FileInputStream(updateFile);
        SdkBytes sourceBytes = SdkBytes.fromInputStream(sourceStream);

        UpdateDatasetEntriesRequest updateDatasetEntriesRequest =
 UpdateDatasetEntriesRequest.builder()
                        .projectName(projectName)
                        .datasetType(datasetType)
                        .changes(sourceBytes)
                        .build();

        lfvClient.updateDatasetEntries(updateDatasetEntriesRequest);

        boolean finished = false;
        DatasetStatus status = null;

        // Wait until update completes.

        do {

                DescribeDatasetRequest describeDatasetRequest =
 DescribeDatasetRequest.builder()
                                .projectName(projectName)
                                .datasetType(datasetType)
```

```
                                .build();
                DescribeDatasetResponse describeDatasetResponse = lfvClient
                                .describeDataset(describeDatasetRequest);

                DatasetDescription datasetDescription =
describeDatasetResponse.datasetDescription();

                status = datasetDescription.status();

                switch (status) {

                        case UPDATE_COMPLETE:
                                logger.log(Level.INFO, "{0} Dataset updated for
project {1}.",
                                                new Object[] { datasetType,
projectName });
                                finished = true;
                                break;

                        case UPDATE_IN_PROGRESS:

                                logger.log(Level.INFO, "{0} Dataset update for
project {1} in progress.",
                                                new Object[] { datasetType,
projectName });
                                TimeUnit.SECONDS.sleep(5);

                                break;

                        case UPDATE_FAILED_ROLLBACK_IN_PROGRESS:
                                logger.log(Level.SEVERE,
                                                "{0} Dataset update failed for
project {1}. Rolling back",
                                                new Object[] { datasetType,
projectName });

                                TimeUnit.SECONDS.sleep(5);

                                break;

                        case UPDATE_FAILED_ROLLBACK_COMPLETE:
                                logger.log(Level.SEVERE,
                                                "{0} Dataset update failed for
project {1}. Rollback completed.",
```

```
                                                    new Object[] { datasetType,
projectName });
                                finished = true;
                                break;

                        default:
                                logger.log(Level.SEVERE,
                                                    "{0} Dataset update failed for
project {1}. Unexpected error returned.",
                                                    new Object[] { datasetType,
projectName });
                                finished = true;

                }

        } while (!finished);

        return status;

}
```

3. 重复上一步并提供其他数据集类型的值。

从数据集中移除图像

您无法直接从数据集中删除图像。您必须删除现有数据集，然后创建一个不包含要移除的图像的新数据集。如何移除图像取决于您是如何将图像导入现有数据集的（清单文件、Amazon S3 桶或本地计算机）。

您还可以使用 AWS SDK 来移除图像。这在创建没有图像分割清单文件的图像分割模型时非常有用，这样就不必使用 Amazon Lookout for Vision 控制台重新绘制图像掩码。

主题

- 从数据集中移除图像（控制台）
- 从数据集中移除图像 (SDK)

从数据集中移除图像（控制台）

通过以下过程，使用 Amazon Lookout for Vision 控制台从数据集中移除图像。

从数据集中移除图像（控制台）

1. 打开项目的数据集图库。
2. 记下要移除的每张图像的名称。
3. 删除现有数据集。
4. 请执行下列操作之一：
   - 如果您使用清单文件创建的数据集，请执行以下操作：
     a. 在文本编辑器中，打开用于创建数据集的清单文件。
     b. 移除您在步骤 2 中记下的每张图像对应的 JSON 行。您可以通过检查 `source-ref` 字段，确定图像对应的 JSON 行。
     c. 保存清单文件。
     d. 使用更新后的清单文件，创建新数据集。
   - 如果您使用从 Amazon S3 桶导入的图像创建的数据集，请执行以下操作：
     a. 从 Amazon S3 桶中删除您在步骤 2 中记下的图像。
     b. 使用 Amazon S3 桶中的剩余图像，创建新数据集。如果您按文件夹名称对图像进行分类，则无需在下一步中对图像进行分类。
     c. 请执行下列操作之一：
        - 如果要创建图像分类模型，请对每张未标注的图像进行分类。
        - 如果要创建图像分割模型，请对每张未标注的图像进行分类和分割。
   - 如果您使用从本地计算机导入的图像创建的数据集，请执行以下操作：
     a. 在您的计算机上，创建一个包含要使用的图像的文件夹。请勿在其中包含要从数据集中移除的图像。有关更多信息，请参阅 使用存储在本地计算机上的图像创建数据集。
     b. 使用您在步骤 4.a 中创建的文件夹中的图像，创建数据集。
     c. 请执行下列操作之一：
        - 如果要创建图像分类模型，请对每张未标注的图像进行分类。
        - 如果要创建图像分割模型，请对每张未标注的图像进行分类和分割。
5. 训练模型。

从数据集中移除图像 (SDK)

您还可以使用 AWS SDK 从数据集中移除图像。

从数据集中移除图像 (SDK)

1. 打开项目的数据集图库。
2. 记下要移除的每张图像的名称。
3. 使用 ListDatasetEntries 操作，导出数据集对应的 JSON 行。
4. 使用导出的 JSON 行创建清单文件。
5. 在文本编辑器中，打开清单文件。
6. 移除您在步骤 2 中记下的每张图像对应的 JSON 行。您可以通过检查 `source-ref` 字段，确定图像对应的 JSON 行。
7. 保存清单文件。
8. 删除现有数据集。
9. 使用更新后的清单文件，创建新数据集。
10. 训练模型。

删除数据集

您可以使用控制台或 `DeleteDataset` 操作，从项目中删除数据集。数据集引用的图像不会被删除。如果从具有训练数据集和测试数据集的项目中删除测试数据集，则该项目将恢复为单数据集项目，剩余的数据集将在训练期间拆分，从而创建训练数据集和测试数据集。如果删除训练数据集，则直到创建新的训练数据集后，然后才能训练项目中的模型。

删除数据集（控制台）

执行以下过程中的步骤，可以删除数据集。如果删除项目中的所有数据集，则会显示创建数据集页面。

删除数据集（控制台）

1. 打开 Amazon Lookout for Vision 控制台，网址为 https://console.aws.amazon.com/lookoutvision/。
2. 选择开始使用。
3. 在左侧导航窗格中，选择项目。

4. 在项目页面上，选择包含要删除的数据集的项目。
5. 在左侧导航窗格中，选择数据集。
6. 选择操作，然后选择要删除的数据集。
7. 在删除对话框中，输入 delete 以确认要删除该数据集。
8. 选择删除训练数据集或删除测试数据集，将数据集删除。

删除数据集 (SDK)

使用 `DeleteDataset` 操作可删除数据集。

删除数据集 (SDK)

1. 安装并配置 AWS CLI 和 AWS SDK（如果尚未如此）。有关更多信息，请参阅 步骤 4：设置 AWS CLI 和 AWS 软件开发工具包。
2. 使用以下示例代码删除模型。

   CLI

   更改以下项的值

   - `project-name` 更改为包含要删除的模型的项目的名称。
   - `dataset-type` 更改为 `train` 或 `test`，具体取决于您要删除哪一种数据集。如果是单数据集项目，请指定 `train` 以删除该数据集。

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

   Python

   此代码取自 AWS 文档 SDK 示例 GitHub 存储库。请在此处查看完整示例。

```
    @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
```

Java V2

此代码取自 AWS 文档 SDK 示例 GitHub 存储库。请在此处查看完整示例。

```
/**
 * Deletes the train or test dataset in an Amazon Lookout for Vision project.
 *
 * @param lfvClient   An Amazon Lookout for Vision client.
 * @param projectName The name of the project in which you want to delete a
 *                    dataset.
 * @param datasetType The type of the dataset that you want to delete (train or
 *                    test).
 * @return Nothing.
 */
public static void deleteDataset(LookoutVisionClient lfvClient, String
 projectName, String datasetType)
                throws LookoutVisionException {

        logger.log(Level.INFO, "Deleting {0} dataset for project {1}",
                        new Object[] { datasetType, projectName });

        DeleteDatasetRequest deleteDatasetRequest =
 DeleteDatasetRequest.builder()
```

```
                        .projectName(projectName)
                        .datasetType(datasetType)
                        .build();

        lfvClient.deleteDataset(deleteDatasetRequest);

        logger.log(Level.INFO, "Deleted {0} dataset for project {1}",
                        new Object[] { datasetType, projectName });

}
```

从项目中导出数据集 (SDK)

您可以使用 AWS SDK，将数据集从 Amazon Lookout for Vision 项目导出到 Amazon S3 桶位置。

通过导出数据集，您可以执行许多人物，如使用源项目的数据集副本，创建一个 Lookout for Vision 项目。您还可以创建用于特定模型版本的数据集快照。

在此过程中，Python 代码会将项目的训练数据集（清单和数据集图像）导出到您指定的目标 Amazon S3 位置。如果项目中存在测试数据集清单和数据集图像，则该代码也会将它们导出。目标可以与源项目位于相同的 Amazon S3 桶中，也可以位于不同的 Amazon S3 桶中。该代码使用 ListDatasetEntries 来获取数据集清单文件。Amazon S3 操作会将数据集图像和更新后的清单文件复制到目标 Amazon S3 位置。

此过程展示了如何导出项目的数据集。它还展示了如何使用导出的数据集创建新项目。

从项目中导出数据集 (SDK)

1. 安装并配置 AWS CLI 和 AWS SDK（如果尚未如此）。有关更多信息，请参阅 步骤 4：设置 AWS CLI 和 AWS 软件开发工具包。
2. 确定用于数据集导出的目标 Amazon S3 路径。请确保目的地位于 Amazon Lookout for Vision 支持的 AWS区域。要创建新的 Amazon S3 桶，请参阅创建桶。
3. 确保用户有权访问用于数据集导出的目标 Amazon S3 路径，以及源项目数据集中图像文件的 S3 位置。您可以使用以下策略，该策略假设图像文件可以位于任何位置。将 *bucket/path* 替换为用于数据集导出的目标桶和路径。

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

```
        {
            "Sid": "PutExports",
            "Effect": "Allow",
            "Action": [
                "S3:PutObjectTagging",
                "S3:PutObject"
            ],
            "Resource": "arn:aws:s3:::bucket/path/*"
        },
        {
            "Sid": "GetSourceRefs",
            "Effect": "Allow",
            "Action": [
                "s3:GetObject",
                "s3:GetObjectTagging",
                "s3:GetObjectVersion"
            ],
            "Resource": "*"
        }
    ]
}
```

要提供访问权限，请为您的用户、组或角色添加权限：

- AWS IAM Identity Center 中的用户和组：

  创建权限集合。按照《AWS IAM Identity Center 用户指南》中 创建权限集 的说明进行操作。

- 通过身份提供商在 IAM 中托管的用户：

  创建适用于身份联合验证的角色。按照《IAM 用户指南》中为第三方身份提供商创建角色（联合身份验证）的说明进行操作。

- IAM 用户：
  - 创建您的用户可以担任的角色。按照《IAM 用户指南》中为 IAM 用户创建角色的说明进行操作。
  - （不推荐使用）将策略直接附加到用户或将用户添加到用户组。按照IAM 用户指南 中向用户添加权限（控制台）中的说明进行操作。

4. 将以下代码保存到名为 `dataset_export.py` 的文件中。

```
"""
```

```
Purpose

Shows how to export the datasets (manifest files and images)
from an Amazon Lookout for Vision project to a new Amazon
S3 location.
"""

import argparse
import json
import logging

import boto3
from botocore.exceptions import ClientError

logger = logging.getLogger(__name__)


def copy_file(s3_resource, source_file, destination_file):
    """
    Copies a file from a source Amazon S3 folder to a destination
    Amazon S3 folder.
    The destination can be in a different S3 bucket.
    :param s3: An Amazon S3 Boto3 resource.
    :param source_file: The Amazon S3 path to the source file.
    :param destination_file: The destination Amazon S3 path for
    the copy operation.
    """

    source_bucket, source_key = source_file.replace("s3://", "").split("/", 1)
    destination_bucket, destination_key = destination_file.replace("s3://",
 "").split(
        "/", 1
    )

    try:
        bucket = s3_resource.Bucket(destination_bucket)
        dest_object = bucket.Object(destination_key)
        dest_object.copy_from(CopySource={"Bucket": source_bucket, "Key":
 source_key})
        dest_object.wait_until_exists()
        logger.info("Copied %s to %s", source_file, destination_file)
    except ClientError as error:
        if error.response["Error"]["Code"] == "404":
            error_message = (
```

```
                f"Failed to copy {source_file} to "
                f"{destination_file}. : {error.response['Error']['Message']}"
            )
            logger.warning(error_message)
            error.response["Error"]["Message"] = error_message
        raise


def upload_manifest_file(s3_resource, manifest_file, destination):
    """
    Uploads a manifest file to a destination Amazon S3 folder.
    :param s3: An Amazon S3 Boto3 resource.
    :param manifest_file: The manifest file that you want to upload.
    :destination: The Amazon S3 folder location to upload the manifest
    file to.
    """

    destination_bucket, destination_key = destination.replace("s3://",
 "").split("/", 1)

    bucket = s3_resource.Bucket(destination_bucket)

    put_data = open(manifest_file, "rb")
    obj = bucket.Object(destination_key + manifest_file)

    try:
        obj.put(Body=put_data)
        obj.wait_until_exists()
        logger.info("Put manifest file '%s' to bucket '%s'.", obj.key,
 obj.bucket_name)
    except ClientError:
        logger.exception(
            "Couldn't put manifest file '%s' to bucket '%s'.", obj.key,
 obj.bucket_name
        )
        raise
    finally:
        if getattr(put_data, "close", None):
            put_data.close()


def get_dataset_types(lookoutvision_client, project):
    """
    Determines the types of the datasets (train or test) in an
```

```
    Amazon Lookout for Vision project.
    :param lookoutvision_client: A Lookout for Vision Boto3 client.
    :param project: The Lookout for Vision project that you want to check.
    :return: The dataset types in the project.
    """

    try:
        response = lookoutvision_client.describe_project(ProjectName=project)

        datasets = []

        for dataset in response["ProjectDescription"]["Datasets"]:
            if dataset["Status"] in ("CREATE_COMPLETE", "UPDATE_COMPLETE"):
                datasets.append(dataset["DatasetType"])
        return datasets

    except lookoutvision_client.exceptions.ResourceNotFoundException:
        logger.exception("Project %s not found.", project)
        raise


def process_json_line(s3_resource, entry, dataset_type, destination):
    """
    Creates a JSON line for a new manifest file, copies image and mask to
    destination.
    :param s3_resource: An Amazon S3 Boto3 resource.
    :param entry: A JSON line from the manifest file.
    :param dataset_type: The type (train or test) of the dataset that
    you want to create the manifest file for.
    :param destination: The destination Amazon S3 folder for the manifest
    file and dataset images.
    :return: A JSON line with details for the destination location.
    """
    entry_json = json.loads(entry)

    print(f"source: {entry_json['source-ref']}")

    # Use existing folder paths to ensure console added image names don't clash.
    bucket, key = entry_json["source-ref"].replace("s3://", "").split("/", 1)
    logger.info("Source location: %s/%s", bucket, key)

    destination_image_location = destination + dataset_type + "/images/" + key

    copy_file(s3_resource, entry_json["source-ref"], destination_image_location)
```

```
    # Update JSON for writing.
    entry_json["source-ref"] = destination_image_location

    if "anomaly-mask-ref" in entry_json:
        source_anomaly_ref = entry_json["anomaly-mask-ref"]
        mask_bucket, mask_key = source_anomaly_ref.replace("s3://", "").split("/",
 1)

        destination_mask_location = destination + dataset_type + "/masks/" +
 mask_key
        entry_json["anomaly-mask-ref"] = destination_mask_location

        copy_file(s3_resource, source_anomaly_ref, entry_json["anomaly-mask-ref"])

    return entry_json


def write_manifest_file(
    lookoutvision_client, s3_resource, project, dataset_type, destination
):
    """
    Creates a manifest file for a dataset. Copies the manifest file and
    dataset images (and masks, if present) to the specified Amazon S3 destination.
    :param lookoutvision_client: A Lookout for Vision Boto3 client.
    :param project: The Lookout for Vision project that you want to use.
    :param dataset_type: The type (train or test) of the dataset that
    you want to create the manifest file for.
    :param destination: The destination Amazon S3 folder for the manifest file
    and dataset images.
    """

    try:
        # Create a reusable Paginator
        paginator = lookoutvision_client.get_paginator("list_dataset_entries")

        # Create a PageIterator from the Paginator
        page_iterator = paginator.paginate(
            ProjectName=project,
            DatasetType=dataset_type,
            PaginationConfig={"PageSize": 100},
        )

        output_manifest_file = dataset_type + ".manifest"
```

```
        # Create manifest file then upload to Amazon S3 with images.
        with open(output_manifest_file, "w", encoding="utf-8") as manifest_file:
            for page in page_iterator:
                for entry in page["DatasetEntries"]:
                    try:
                        entry_json = process_json_line(
                            s3_resource, entry, dataset_type, destination
                        )

                        manifest_file.write(json.dumps(entry_json) + "\n")

                    except ClientError as error:
                        if error.response["Error"]["Code"] == "404":
                            print(error.response["Error"]["Message"])
                            print(f"Excluded JSON line: {entry}")
                        else:
                            raise
        upload_manifest_file(
            s3_resource, output_manifest_file, destination + "datasets/"
        )

    except ClientError:
        logger.exception("Problem getting dataset_entries")
        raise


def export_datasets(lookoutvision_client, s3_resource, project, destination):
    """
    Exports the datasets from an Amazon Lookout for Vision project to a specified
    Amazon S3 destination.
    :param project: The Lookout for Vision project that you want to use.
    :param destination: The destination Amazon S3 folder for the exported datasets.
    """
    # Add trailing backslash, if missing.
    destination = destination if destination[-1] == "/" else destination + "/"

    print(f"Exporting project {project} datasets to {destination}.")

    # Get each dataset and export to destination.

    dataset_types = get_dataset_types(lookoutvision_client, project)
    for dataset in dataset_types:
        logger.info("Copying %s dataset to %s.", dataset, destination)
```

```
        write_manifest_file(
            lookoutvision_client, s3_resource, project, dataset, destination
        )

    print("Exported dataset locations")
    for dataset in dataset_types:
        print(f"   {dataset}: {destination}datasets/{dataset}.manifest")

    print("Done.")


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

    parser.add_argument("project", help="The project that contains the dataset.")
    parser.add_argument("destination", help="The destination Amazon S3 folder.")


def main():
    """
    Exports the datasets from an Amazon Lookout for Vision project to a
    destination Amazon S3 location.
    """
    logging.basicConfig(level=logging.INFO, format="%(levelname)s: %(message)s")
    parser = argparse.ArgumentParser(usage=argparse.SUPPRESS)
    add_arguments(parser)

    args = parser.parse_args()

    try:
        session = boto3.Session(profile_name="lookoutvision-access")
        lookoutvision_client = session.client("lookoutvision")
        s3_resource = session.resource("s3")

        export_datasets(
            lookoutvision_client, s3_resource, args.project, args.destination
        )
    except ClientError as err:
        logger.exception(err)
        print(f"Failed: {format(err)}")
```

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

5. 运行该代码。提供以下命令行参数：

   - 项目：包含要导出的数据集的源项目的名称。
   - 目标：数据集的目标 Amazon S3 路径。

   例如，`python dataset_export.py myproject s3://bucket/path/`

6. 记下代码显示的清单文件位置。您在步骤 8 中需要用到它们。
7. 按照创建您的项目中的说明，使用导出的数据集创建新的 Lookout for Vision 项目。
8. 请执行下列操作之一：

   - 按照使用清单文件创建数据集（控制台）中的说明，使用 Lookout for Vision 控制台为新项目创建数据集。您不需要执行步骤 1-6。

     对于目标 12，请执行以下操作：

     a. 如果源项目有测试数据集，请选择单独的训练数据集和测试数据集，否则选择单数据集。

     b. 对于 .manifest 文件位置，请输入您在步骤 6 中记下的适当清单文件（训练或测试）的位置。

   - 使用 CreateDataset 操作，利用使用清单文件创建数据集（SDK）中的代码为您的新项目创建数据集。对于 `manifest_file` 参数，请使用您在步骤 6 中记下的清单文件位置。如果源项目有测试数据集，请再次使用该代码创建测试数据集。

9. 如果您已做好准备，请按照训练您的模型中的说明训练模型。

查看您的模型

一个项目可以有多个模型版本。您可以使用控制台查看项目中的模型。您还可以使用 `ListModels` 操作。

> **Note**
>
> 模型列表具有最终一致性。如果创建模型，您可能需要稍等片刻，然后模型列表才会达到最新状态。

查看您的模型（控制台）

执行以下过程中的步骤，可以在控制台中查看项目的模型。

查看您的模型（控制台）

1. 打开 Amazon Lookout for Vision 控制台，网址为 https://console.aws.amazon.com/lookoutvision/。
2. 选择开始使用。
3. 在左侧导航窗格中，选择项目。
4. 在项目页面上，选择包含您要查看的模型的项目。
5. 在左侧导航窗格中，选择模型，然后查看模型的详细信息。

查看您的模型 (SDK)

要查看模型的版本，您需要使用 `ListModels` 操作。要获取与特定模型版本有关的信息，请使用 `DescribeModel` 操作。下面的示例会列出项目中的所有模型版本，然后显示各个模型版本的性能和输出配置信息。

查看您的模型 (SDK)

1. 安装并配置 AWS CLI 和 AWS SDK（如果尚未如此）。有关更多信息，请参阅 步骤 4：设置 AWS CLI 和 AWS 软件开发工具包。
2. 使用以下示例代码，列出您的模型并获取与模型有关的信息。

   CLI

   使用 `list-models` 命令列出项目中的模型。

   更改以下值：

   - `project-name` 更改为包含要查看的模型的项目的名称。

```
aws lookoutvision list-models --project-name project name \
  --profile lookoutvision-access
```

使用 describe-model 命令获取与模型有关的信息。更改以下值：

- project-name 更改为包含要查看的模型的项目的名称。
- model-version 更改为要描述的模型版本。

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

Python

此代码取自 AWS 文档 SDK 示例 GitHub 存储库。请在此处查看完整示例。

```
    @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
```

Java V2

此代码取自 AWS 文档 SDK 示例 GitHub 存储库。请在此处查看完整示例。

```
/**
 * Lists the models in an Amazon Lookout for Vision project.
 *
 * @param lfvClient   An Amazon Lookout for Vision client.
 * @param projectName The name of the project that contains the models that
 *                    you want to list.
 * @return List <Metadata> A list of models in the project.
 */
public static List<ModelMetadata> listModels(LookoutVisionClient lfvClient,
 String projectName)
                throws LookoutVisionException {

        ListModelsRequest listModelsRequest = ListModelsRequest.builder()
                        .projectName(projectName)
                        .build();

        // Get a list of models in the supplied project.
        ListModelsResponse response = lfvClient.listModels(listModelsRequest);

        for (ModelMetadata model : response.models()) {
                logger.log(Level.INFO, "Model ARN: {0}\nVersion: {1}\nStatus:
 {2}\nMessage: {3}", new Object[] {
                                model.modelArn(),
                                model.modelVersion(),
                                model.statusMessage(),
                                model.statusAsString() });
        }

        return response.models();

}
```

删除模型

您可以使用控制台或使用 DeleteModel 操作，删除模型的版本。您无法删除正在运行或正在训练的模型版本。

如果模型版本正在运行，请先使用 StopModel 操作停止该模型版本。有关更多信息，请参阅 停止您的 Amazon Lookout for Vision 模型。如果模型正在训练，请等到训练完成后再删除模型。

删除模型可能需要几秒钟。要确定模型是否已删除，请调用 ListProjects，并检查 Models 数组中是否有该模型的版本 (ModelVersion)。

删除模型（控制台）

执行以下步骤，可以从控制台中删除模型。

删除模型（控制台）

1. 打开 Amazon Lookout for Vision 控制台，网址为 https://console.aws.amazon.com/lookoutvision/。
2. 选择开始使用。
3. 在左侧导航窗格中，选择项目。
4. 在项目页面上，选择包含您要删除的模型的项目。
5. 在左侧导航窗格中，选择 Models (模型)。
6. 在模型视图中，选择要删除的模型对应的单选按钮。
7. 在页面的顶部，选择 Delete (删除)。
8. 在删除对话框中，输入 delete 以确认要删除该模型。
9. 选择删除模型，将模型删除。

删除模型 (SDK)

通过以下过程，可使用 DeleteModel 操作删除模型。

删除模型 (SDK)

1. 安装并配置 AWS CLI 和 AWS SDK（如果尚未如此）。有关更多信息，请参阅 步骤 4：设置 AWS CLI 和 AWS 软件开发工具包。
2. 使用以下示例代码删除模型。

   CLI

   更改以下值：

   - project-name 更改为包含要删除的模型的项目的名称。

- model-version 更改为要删除的模型版本。

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

Python

此代码取自 AWS 文档 SDK 示例 GitHub 存储库。请在此处查看完整示例。

```
    @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
```

Java V2

此代码取自 AWS 文档 SDK 示例 GitHub 存储库。请在此处查看完整示例。

```
/**
* Deletes an Amazon Lookout for Vision model.
*
* @param lfvClient    An Amazon Lookout for Vision client. Returns after the
 model is deleted.
* @param projectName  The name of the project that contains the model that you
 want to delete.
* @param modelVersion The version of the model that you want to delete.
* @return void
*/
public static void deleteModel(LookoutVisionClient lfvClient,
                String projectName,
                String modelVersion) throws LookoutVisionException,
 InterruptedException {

        DeleteModelRequest deleteModelRequest = DeleteModelRequest.builder()
                        .projectName(projectName)
                        .modelVersion(modelVersion)
                        .build();

        lfvClient.deleteModel(deleteModelRequest);

        boolean deleted = false;

        do {

                ListModelsRequest listModelsRequest =
 ListModelsRequest.builder()
                                .projectName(projectName)
                                .build();
```

```
                // Get a list of models in the supplied project.
                ListModelsResponse response =
lfvClient.listModels(listModelsRequest);

                ModelMetadata modelMetadata = response.models().stream()
                                .filter(model ->
model.modelVersion().equals(modelVersion)).findFirst()
                                .orElse(null);

                if (modelMetadata == null) {
                        deleted = true;
                        logger.log(Level.INFO, "Deleted: Model version {0} of
project {1}.",
                                        new Object[] { modelVersion,
projectName });

                } else {
                        logger.log(Level.INFO, "Not yet deleted: Model version
{0} of project {1}.",
                                        new Object[] { modelVersion,
projectName });
                        TimeUnit.SECONDS.sleep(60);
                }

        } while (!deleted);

}
```

标记模型

通过使用标签，您可以识别、整理、搜索和筛选自己的 Amazon Lookout for Vision 模型。每个标签都是由用户定义的键和值组成的标签。例如，为了帮助确定模型的账单，您可使用 `Cost center` 键来标记模型，并添加适当的成本中心编号作为值。有关更多信息，请参阅标记亚马逊云科技资源。

使用标签可以执行以下操作：

- 使用成本分配标签跟踪模型的账单。有关更多信息，请参阅使用成本分配标签。
- 使用 Identity and Access Management (IAM) 控制对模型的访问。有关更多信息，请参阅使用资源标签控制对亚马逊云科技资源的访问。

- 自动管理模型。例如，可以运行自动启动或停止脚本，这些脚本可在非工作时间内关闭开发模型以降低成本。有关更多信息，请参阅 运行经过训练的 Amazon Lookout for Vision 模型。

您可以使用 Amazon Lookout for Vision 控制台或 AWS SDK 来标记模型。

主题

- 标记模型（控制台）
- 标记模型 (SDK)

标记模型（控制台）

您可以使用 Amazon Lookout for Vision 控制台为模型添加标签、查看附加到模型的标签并移除标签。

添加或移除标签（控制台）

此过程说明了如何向现有模型添加标签，或从现有模型中移除标签。您也可以在训练新模型时向其添加标签。有关更多信息，请参阅 训练您的模型。

向现有模型添加标签或从现有模型中移除标签（控制台）

1. 打开 Amazon Lookout for Vision 控制台，网址为 https://console.aws.amazon.com/lookoutvision/。
2. 选择开始使用。
3. 在导航窗格中，选择 Projects (项目)。
4. 在项目资源页面上，选择包含要标记的模型的项目。
5. 在导航窗格中，于之前选择的项目下，选择模型。
6. 在模型部分，选择要为其添加标签的模型。
7. 在模型的详细信息页面上，选择标签选项卡。
8. 在 Tags (标签) 部分中，选择 Manage tags (管理标签)。
9. 在管理标签页面上，选择添加新标签。
10. 输入键和值。
    a. 对于键，请为键输入一个名称。
    b. 对于 Value (值)，输入一个值。

11. 要添加更多标签，请重复步骤 9 和步骤 10。
12. （可选）要移除标签，请选择要移除的标签旁的移除。如果要移除先前保存的标签，则会在保存所做的更改时将其移除。
13. 选择保存更改以保存您的更改。

查看模型标签（控制台）

您可以使用 Amazon Lookout for Vision 控制台，查看附加到模型的标签。

要查看附加到项目内所有模型 的标签，必须使用 Amazon SDK。有关更多信息，请参阅 列出模型标签 (SDK)。

查看附加到模型的标签

1. 打开 Amazon Lookout for Vision 控制台，网址为 https://console.aws.amazon.com/lookoutvision/。
2. 选择开始使用。
3. 在导航窗格中，选择 Projects (项目)。
4. 在项目资源页面上，选择包含要查看其标签的模型的项目。
5. 在导航窗格中，于之前选择的项目下，选择模型。
6. 在模型部分，选择要查看其标签的模型。
7. 在模型的详细信息页面上，选择标签选项卡。标签部分中便会显示标签。

标记模型 (SDK)

您可以使用 AWS SDK 执行以下操作：

- 向新模型添加标签
- 向现有模型添加标签
- 列出附加到模型的标签
- 从模型中移除标签

此部分中包含 AWS CLI 示例。如果尚未安装 AWS CLI，请参阅步骤 4：设置 AWS CLI 和 AWS 软件开发工具包。

向新模型添加标签 (SDK)

您可以在使用 CreateModel 操作创建模型时向其添加标签。请在 Tags 数组输入参数中指定一个或多个标签。

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

有关创建和训练模型的信息，请参阅训练模型（SDK）。

向现有模型添加标签 (SDK)

要向现有模型添加一个或多个标签，请使用 TagResource 操作。指定模型的 Amazon 资源名称 (ARN) (ResourceArn) 和要添加的标签 (Tags)。

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

有关 Java 代码示例，请参阅 TagModel。

列出模型标签 (SDK)

要列出附加到模型的标签，请使用 ListTagsForResource 操作并指定模型的 Amazon 资源名称 (ARN)，即 (ResourceArn)。该操作的响应是附加到指定模型的标签键和值的映射图。

```
aws lookoutvision list-tags-for-resource --resource-arn resource-arn \
  --profile lookoutvision-access
```

要查看项目中哪些模型具有特定标签，可调用 ListModels 获取模型列表。然后，针对 ListModels 响应中的每个模型分别调用 ListTagsForResource。检查 ListTagsForResource 的响应，查看是否存在所需的标签。

有关 Java 代码示例，请参阅 ListModelTags。有关用于在所有项目中搜索标签值的 Python 代码示例，请参阅 find_tag.py。

从模型中移除标签 (SDK)

要从模型中移除一个或多个标签，请使用 UntagResource 操作。指定模型的 Amazon 资源名称 (ARN) (ResourceArn) 和要移除的标签键 (Tag-Keys)。

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

有关 Java 代码示例，请参阅 UntagModel。

查看您的试用检测任务

您可以使用控制台查看自己的试用检测。您无法使用 AWS SDK 查看试用检测任务。

> **Note**
>
> 试用检测列表具有最终一致性。如果创建试用检测，您可能需要稍等片刻，然后试用检测列表才会达到最新状态。

查看您的试用检测任务（控制台）

使用以下过程可查看您的试用检测。

查看您的试用检测任务

1. 打开 Amazon Lookout for Vision 控制台，网址为 https://console.aws.amazon.com/lookoutvision/。
2. 选择开始使用。
3. 在左侧导航窗格中，选择试用检测。
4. 在试用检测页面上，选择一项试用检测任务，以查看其详细信息。

示例代码和数据集

以下是您可以用于 Amazon Lookout for Vision 的代码示例和数据集。

主题

- 示例代码
- 示例数据集

示例代码

以下 Amazon Lookout for Vision 的代码示例可供使用。

示例	描述
GitHub	用于训练和托管 Amazon Lookout for Vision 模型的 Python 代码示例。
Amazon Lookout for Vision 实验室	一个 Python 笔记本，您可以用它来创建具有电路板示例图像的模型。
Python 示例代码	Amazon Lookout for Vision 文档中使用的 Python 示例。
Java 示例代码	Amazon Lookout for Vision 文档中使用的 Java 示例。

示例数据集

以下是您可以用于 Amazon Lookout for Vision 的示例数据集。

主题

- 图像分割数据集
- 图像分类数据集

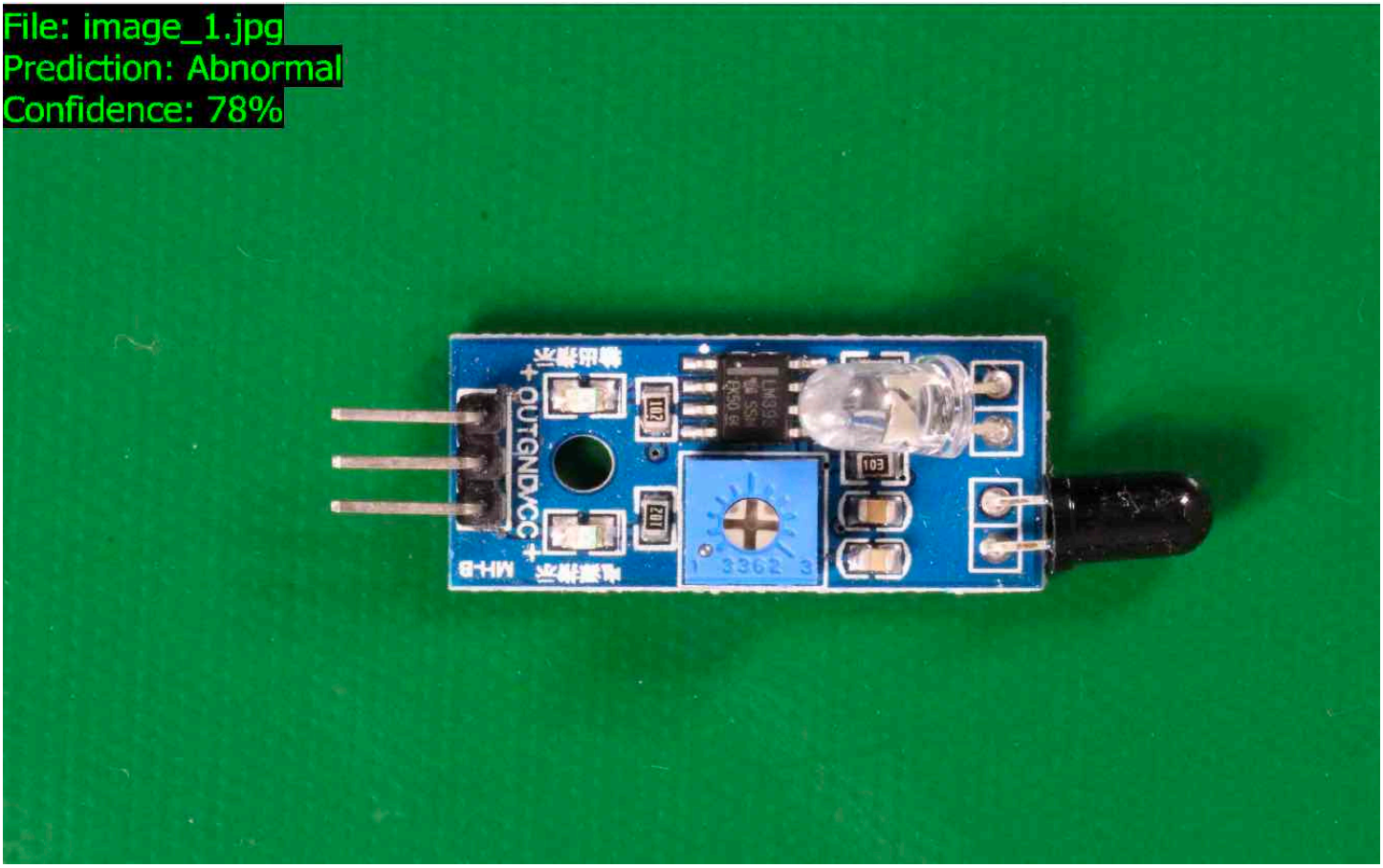
图像分割数据集

开始使用 Amazon Lookout for Vision 提供了一个破损饼干数据集，您可以用它来创建图像分割模型。

有关用于创建图像分割模型的另一个数据集，请参阅不使用 GPU 而在边缘利用 Amazon Lookout for Vision 识别异常的位置。

图像分类数据集

Amazon Lookout for Vision 提供了一些电路板的示例图像，您可用它们来创建图像分类模型。



您可以从 https://github.com/aws-samples/amazon-lookout-for-vision GitHub 存储库复制这些图像。这些图像位于 `circuitboard` 文件夹中。

`circuitboard` 文件夹包含以下文件夹。

- `train`：您可以在训练数据集使用的图像。
- `test`：您可以在测试数据集使用的图像。

- `extra_images`：您可以用来执行试用检测的图像，或者可用来通过 DetectAnomalies 操作试用所训练模型的图像。

`train` 和 `test` 文件夹各有一个名为 `normal` 的子文件夹（包含正常的图像）和一个名为 `anomaly` 的子文件夹（包含存在异常的图像）。

> **Note**
>
> 随后，当您使用控制台创建数据集时，Amazon Lookout for Vision 可以使用文件夹名称（`normal` 和 `anomaly`）自动标注图像。有关更多信息，请参阅 the section called “Amazon S3 存储桶”。

准备数据集图像

1. 将 https://github.com/aws-samples/amazon-lookout-for-vision 存储库克隆到您的计算机。有关更多信息，请参阅克隆存储库。
2. 创建 Amazon S3 存储桶。有关更多信息，请参阅如何创建 S3 桶？。
3. 在命令提示符下输入以下命令，以便将数据集图像从您的计算机复制到 Amazon S3 桶。

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

上传图像后，您便可以创建模型。通过从您之前将电路板图像上传到的 Amazon S3 位置添加图像，您可以自动对图像进行分类。请记住，您需要按照每次成功训练模型以及模型的运行（托管）时间量付费。

创建分类模型

1. 完成创建项目（控制台）。
2. 完成使用存储在 Amazon S3 桶中的图像创建数据集。
   - 在步骤 6 中，选择单独的训练数据集和测试数据集选项卡。
   - 在步骤 8a 中，输入您在准备数据集图像中上传的训练图像的 S3 URI。例如 `s3://your-bucket/circuitboard/train`。在步骤 8b 中，输入测试数据集的 S3 URI。例如，`s3://your-bucket/circuitboard/test`。
   - 请务必完成步骤 9。

3. 完成训练模型（控制台）。
4. 完成启动您的模型（控制台）。
5. 完成检测图像中的异常。您可以使用 `test_images` 文件夹中的图像。
6. 当您使用完模型后，请完成停止您的模型（控制台）。

Amazon Lookout for Vision 中的安全性

AWS 十分重视云安全性。为了满足对安全性最敏感的组织的需求，我们打造了具有超高安全性的数据中心和网络架构。作为 AWS 客户，您也将从这些数据中心和网络架构受益。

安全性是 AWS 和您的共同责任。责任共担模型将其描述为云的安全性和云中的安全性：

- 云的安全性 – AWS 负责保护在 AWS 云中运行 AWS 服务的基础设施。AWS 还向您提供可安全使用的服务。第三方审核员定期测试和验证我们的安全性的有效性，作为 AWS Compliance Programs 的一部分。要了解适用于 Amazon Lookout for Vision 的合规性计划，请参阅合规性计划范围内的亚马逊云科技服务。
- 云中的安全性 - 您的责任由您使用的 AWS 服务决定。您还需要对其它因素负责，包括您的数据的敏感性、您公司的要求以及适用的法律法规。

此文档将帮助您了解如何在使用 Lookout for Vision 时应用责任共担模式。以下主题说明如何配置 Lookout for Vision 以实现您的安全性和合规性目标。您还将了解如何使用其他亚马逊云科技服务来帮助您监控和保护自己的 Lookout for Vision 资源。

主题

- Amazon Lookout for Vision 中的数据保护
- 适用于 Amazon Lookout for Vision 的身份和访问管理
- Amazon Lookout for Vision 的合规性验证
- Amazon Lookout for Vision 中的韧性
- Amazon Lookout for Vision 中的基础设施安全性

Amazon Lookout for Vision 中的数据保护

AWS责任共担模式会应用于 Amazon Lookout for Vision 中的数据保护。如该模式中所述，AWS 负责保护运行所有 AWS Cloud 的全球基础设施。您负责维护对托管在此基础设施上的内容的控制。您还负责您所使用的 AWS 服务 的安全配置和管理任务。有关数据隐私的更多信息，请参阅数据隐私常见问题。有关欧洲数据保护的信息，请参阅 AWS 安全性博客 上的博客文章 AWS Shared Responsibility Model and GDPR。

出于数据保护目的，我们建议您保护 AWS 账户 凭证并使用 AWS IAM Identity Center 或 AWS Identity and Access Management (IAM) 设置单个用户。这样，每个用户只获得履行其工作职责所需的权限。我们还建议您通过以下方式保护数据：

- 对每个账户使用多重身份验证 (MFA)。
- 使用 SSL/TLS 与 AWS 资源进行通信。我们要求使用 TLS 1.2，建议使用 TLS 1.3。
- 使用 AWS CloudTrail 设置 API 和用户活动日志记录。
- 使用 AWS 加密解决方案以及 AWS 服务 中的所有默认安全控制。
- 使用高级托管安全服务（例如 Amazon Macie），它有助于发现和保护存储在 Amazon S3 中的敏感数据。
- 如果您在通过命令行界面或 API 访问 AWS 时需要经过 FIPS 140-2 验证的加密模块，请使用 FIPS 端点。有关可用的 FIPS 端点的更多信息，请参阅《美国联邦信息处理标准 (FIPS) 第 140-2 版》。

我们强烈建议您切勿将机密信息或敏感信息（如您客户的电子邮件地址）放入标签或自由格式文本字段（如名称字段）。这包括通过控制台、API、AWS CLI 或 AWS SDK 使用 Lookout for Vision 或其他 AWS 服务的情况。在用于名称的标签或自由格式文本字段中输入的任何数据都可能会用于计费或诊断日志。如果您向外部服务器提供网址，我们强烈建议您不要在网址中包含凭证信息来验证对该服务器的请求。

数据加密

以下信息说明了 Amazon Lookout for Vision 在哪些情况下使用数据加密来保护您的数据。

静态加密

映像

为了训练您的模型，Amazon Lookout for Vision 会复制您的源训练图像和测试图像。所复制的图像会在 Amazon Simple Storage Service (S3) 中，通过服务器端加密，使用 AWS 拥有的密钥或您提供的密钥进行静态加密。这些密钥使用 Amazon Key Management Service (SSE-KMS) 来存储。源图像不受影响。有关更多信息，请参阅训练您的模型。

Amazon Lookout for Vision 模型

默认情况下，经过训练的模型和清单文件会在 Amazon S3 中使用服务器端加密进行加密，加密使用的是 Amazon Key Management Service 中存储的 KMS 密钥 (SSE-KMS)。Lookout for Vision 使用 AWS 拥有的密钥。有关更多信息，请参阅使用服务器端加密保护数据。训练结果会写入到 `CreateModel` 的 `output_bucket` 输入参数中指定的桶中。对于这些训练结果，将使用为桶 (`output_bucket`) 配置的加密设置进行加密。

Amazon Lookout for Vision 控制台桶

Amazon Lookout for Vision 控制台会创建一个 Amazon S3 桶（控制台桶），您可用它来管理自己的项目。控制台桶使用默认 Amazon S3 加密进行加密。有关更多信息，请参阅适用于 S3 存储桶的 Amazon Simple Storage Service 默认加密。如果您使用自己的 KMS 密钥，请在创建控制台存储桶后对其进行配置。有关更多信息，请参阅使用服务器端加密保护数据。Amazon Lookout for Vision 阻止对控制台桶的公共访问。

传输中加密

Amazon Lookout for Vision API 端点仅支持基于 HTTPS 的安全连接。所有通信都使用传输层安全性协议 (TLS) 进行加密。

密钥管理

您可以使用 Amazon Key Management Service (KMS)，对存储在 Amazon S3 桶中的输入图像管理它们的加密。有关更多信息，请参阅步骤 5：（可选）使用自己的 Amazon Key Management Service 密钥。

默认情况下，您的图像使用亚马逊云科技拥有和管理的密钥进行加密。您也可以选择使用自己的 Amazon Key Management Service (KMS) 密钥。有关更多信息，请参阅 Amazon Key Management Service 概念。

互联网络流量隐私保护

适用于 Amazon Lookout for Vision 的 Amazon Virtual Private Cloud (Amazon VPC) 端点是 VPC 内的逻辑实体，仅允许连接到 Amazon Lookout for Vision。Amazon VPC 会将请求发送到 Amazon Lookout for Vision，并将响应发送回 VPC。有关更多信息，请参阅《Amazon VPC 用户指南》的 VPC 端点。有关 Amazon VPC 端点与 Amazon Lookout for Vision 一起使用的信息，请参阅使用接口端点 (AWS PrivateLink) 访问 Amazon Lookout for Vision。

适用于 Amazon Lookout for Vision 的身份和访问管理

AWS Identity and Access Management (IAM) AWS 服务 可帮助管理员安全地控制对 AWS 资源的访问权限。IAM 管理员控制着谁可以通过身份验证（登录）并获得授权（拥有权限），从而使用 Lookout for Vision 资源。您可以使用 IAM AWS 服务 ，无需支付额外费用。

主题

- 受众
- 使用身份进行身份验证
- 使用策略管理访问
- Amazon Lookout for Vision 如何与 IAM 协同工作
- Amazon Lookout for Vision 基于身份的策略示例
- 适用于 Amazon Lookout for Vision 的 AWS 托管式策略
- Amazon Lookout for Vision 身份和访问权限问题排查

受众

你的使用方式 AWS Identity and Access Management (IAM) 会有所不同，具体取决于你在 Lookout for Vision 中所做的工作。

服务用户：如果您使用 Lookout for Vision 服务完成自己的工作，则您的管理员会为您提供所需的凭证和权限。当您要使用更多 Lookout for Vision 功能来完成工作时，可能会需要额外的权限。了解如何管理访问权限有助于您向管理员请求适合的权限。如果您无法访问 Lookout for Vision 中的功能，请参阅 Amazon Lookout for Vision 身份和访问权限问题排查。

服务管理员：如果您在公司负责管理 Lookout for Vision 资源，则可能具有对 Lookout for Vision 的完全访问权限。您负责确定自己的服务用户应访问哪些 Lookout for Vision 功能和资源。然后，您必须向 IAM 管理员提交请求以更改服务用户的权限。请查看该页面上的信息以了解 IAM 的基本概念。要了解有关您的公司如何能够将 IAM 与 Lookout for Vision 搭配使用的更多信息，请参阅 Amazon Lookout for Vision 如何与 IAM 协同工作。

IAM 管理员：如果您是 IAM 管理员，可能会希望详细了解如何编写策略，以便管理对 Lookout for Vision 的访问权限。要查看您可在 IAM 中使用的 Lookout for Vision 基于身份的策略示例，请参阅 Amazon Lookout for Vision 基于身份的策略示例。

使用身份进行身份验证

身份验证是您 AWS 使用身份凭证登录的方式。您必须以 IAM 用户身份或通过担 AWS 账户根用户任 IAM 角色进行身份验证（登录 AWS）。

您可以使用通过身份源提供的凭据以 AWS 联合身份登录。 AWS IAM Identity Center （IAM Identity Center）用户、贵公司的单点登录身份验证以及您的 Google 或 Facebook 凭据就是联合身份的示例。当您以联合身份登录时，您的管理员以前使用 IAM 角色设置了身份联合验证。当你使用联合访问 AWS 时，你就是在间接扮演一个角色。

根据您的用户类型，您可以登录 AWS Management Console 或 AWS 访问门户。有关登录的更多信息 AWS，请参阅《AWS 登录 用户指南》中的如何登录到您 AWS 账户的。

如果您 AWS 以编程方式访问，则会 AWS 提供软件开发套件 (SDK) 和命令行接口 (CLI)，以便使用您的凭据对请求进行加密签名。如果您不使用 AWS 工具，则必须自己签署请求。有关使用推荐的方法自行签署请求的更多信息，请参阅 IAM 用户指南中的签署 AWS API 请求。

无论使用何种身份验证方法，您可能需要提供其他安全信息。例如， AWS 建议您使用多重身份验证 (MFA) 来提高账户的安全性。要了解更多信息，请参阅《AWS IAM Identity Center 用户指南》中的 多重身份验证和《IAM 用户指南》中的 在 AWS中使用多重身份验证（MFA）。

AWS 账户 root 用户

创建时 AWS 账户，首先要有一个登录身份，该身份可以完全访问账户中的所有资源 AWS 服务 和资源。此身份被称为 AWS 账户 root 用户，使用您创建账户时使用的电子邮件地址和密码登录即可访问该身份。强烈建议您不要使用根用户执行日常任务。保护好根用户凭证，并使用这些凭证来执行仅根用户可以执行的任务。有关要求您以根用户身份登录的任务的完整列表，请参阅《IAM 用户指南》中的需要根用户凭证的任务。

联合身份

作为最佳实践，要求人类用户（包括需要管理员访问权限的用户）使用与身份提供商的联合身份验证 AWS 服务 通过临时证书进行访问。

联合身份是指您的企业用户目录、Web 身份提供商、Identity C enter 目录中的用户，或者任何使用 AWS 服务 通过身份源提供的凭据进行访问的用户。 AWS Directory Service当联合身份访问时 AWS 账户，他们将扮演角色，角色提供临时证书。

要集中管理访问权限，建议您使用 AWS IAM Identity Center。您可以在 IAM Identity Center 中创建用户和群组，也可以连接并同步到您自己的身份源中的一组用户和群组，以便在您的所有 AWS 账户 和应用程序中使用。有关 IAM Identity Center 的信息，请参阅《AWS IAM Identity Center 用户指南》中的什么是 IAM Identity Center？

IAM 用户和群组

I AM 用户是您 AWS 账户 内部对个人或应用程序具有特定权限的身份。在可能的情况下，我们建议使用临时凭证，而不是创建具有长期凭证（如密码和访问密钥）的 IAM 用户。但是，如果您有一些特定的使用场景需要长期凭证以及 IAM 用户，建议您轮换访问密钥。有关更多信息，请参阅《IAM 用户指南》中的 对于需要长期凭证的使用场景定期轮换访问密钥。

IAM 组是一个指定一组 IAM 用户的身份。您不能使用组的身份登录。您可以使用组来一次性为多个用户指定权限。如果有大量用户，使用组可以更轻松地管理用户权限。例如，您可能具有一个名为 IAMAdmins 的组，并为该组授予权限以管理 IAM 资源。

用户与角色不同。用户唯一地与某个人员或应用程序关联，而角色旨在让需要它的任何人代入。用户具有永久的长期凭证，而角色提供临时凭证。要了解更多信息，请参阅《IAM 用户指南》中的何时创建 IAM 用户（而不是角色）。

IAM 角色

I AM 角色是您内部具有特定权限 AWS 账户 的身份。它类似于 IAM 用户，但与特定人员不关联。您可以 AWS Management Console 通过切换角色在中临时担任 IAM 角色。您可以通过调用 AWS CLI 或 AWS API 操作或使用自定义 URL 来代入角色。有关使用角色的方法的更多信息，请参阅《IAM 用户指南》中的使用 IAM 角色。

具有临时凭证的 IAM 角色在以下情况下很有用：

- 联合用户访问 – 要向联合身份分配权限，请创建角色并为角色定义权限。当联合身份进行身份验证时，该身份将与角色相关联并被授予由此角色定义的权限。有关联合身份验证的角色的信息，请参阅《IAM 用户指南》中的为第三方身份提供商创建角色。如果您使用 IAM Identity Center，则需要配置权限集。为控制您的身份在进行身份验证后可以访问的内容，IAM Identity Center 将权限集与 IAM 中的角色相关联。有关权限集的信息，请参阅《AWS IAM Identity Center 用户指南》中的 权限集。
- 临时 IAM 用户权限 – IAM 用户可代入 IAM 用户或角色，以暂时获得针对特定任务的不同权限。
- 跨账户存取 – 您可以使用 IAM 角色以允许不同账户中的某个人（可信主体）访问您的账户中的资源。角色是授予跨账户访问权限的主要方式。但是，对于某些资源 AWS 服务，您可以将策略直接附加到资源（而不是使用角色作为代理）。要了解角色和基于资源的跨账户访问策略之间的区别，请参阅 IA M 用户指南中的跨账户资源访问。
- 跨服务访问 — 有些 AWS 服务 使用其他 AWS 服务服务中的功能。例如，当您在某个服务中进行调用时，该服务通常会在 Amazon EC2 中运行应用程序或在 Simple Storage Service (Amazon S3) 中存储对象。服务可能会使用发出调用的主体的权限、使用服务角色或使用服务相关角色来执行此操作。
  - 转发访问会话 (FAS) — 当您使用 IAM 用户或角色在中执行操作时 AWS，您被视为委托人。使用某些服务时，您可能会执行一个操作，然后此操作在其他服务中启动另一个操作。FAS 使用调用委托人的权限以及 AWS 服务 向下游服务发出请求的请求。 AWS 服务只有当服务收到需要与其他 AWS 服务 或资源交互才能完成的请求时，才会发出 FAS 请求。在这种情况下，您必须具有执行这两个操作的权限。有关发出 FAS 请求时的策略详情，请参阅转发访问会话。

- 服务角色 - 服务角色是服务代表您在您的账户中执行操作而分派的 IAM 角色。IAM 管理员可以在 IAM 中创建、修改和删除服务角色。有关更多信息，请参阅《IAM 用户指南》中的创建向 AWS 服务委派权限的角色。
- 服务相关角色-服务相关角色是一种与服务相关联的服务角色。 AWS 服务服务可以代入代表您执行操作的角色。服务相关角色出现在您的中 AWS 账户 ，并且归服务所有。IAM 管理员可以查看但不能编辑服务相关角色的权限。

- 在 A@@ mazon EC2 上运行的应用程序 — 您可以使用 IAM 角色管理在 EC2 实例上运行并发出 AWS CLI 或 AWS API 请求的应用程序的临时证书。这优先于在 EC2 实例中存储访问密钥。要向 EC2 实例分配 AWS 角色并使其可供其所有应用程序使用，您需要创建附加到该实例的实例配置文件。实例配置文件包含角色，并使 EC2 实例上运行的程序能够获得临时凭证。有关更多信息，请参阅《IAM 用户指南》中的使用 IAM 角色为 Amazon EC2 实例上运行的应用程序授予权限。

要了解是使用 IAM 角色还是 IAM 用户，请参阅IAM 用户指南中的何时创建 IAM 角色（而不是用户）。

使用策略管理访问

您可以 AWS 通过创建策略并将其附加到 AWS 身份或资源来控制中的访问权限。策略是其中的一个对象 AWS ，当与身份或资源关联时，它会定义其权限。 AWS 在委托人（用户、root 用户或角色会话）发出请求时评估这些策略。策略中的权限确定是允许还是拒绝请求。大多数策略都以 JSON 文档的 AWS 形式存储在中。有关 JSON 策略文档的结构和内容的更多信息，请参阅 IAM 用户指南中的 JSON 策略概览。

管理员可以使用 AWS JSON 策略来指定谁有权访问什么。也就是说，哪个主体可以对什么资源执行操作，以及在什么条件下执行。

默认情况下，用户和角色没有权限。要授予用户对所需资源执行操作的权限，IAM 管理员可以创建 IAM 策略。管理员随后可以向角色添加 IAM policy，用户可以代入角色。

IAM 策略定义操作的权限，无关乎您使用哪种方法执行操作。例如，假设您有一个允许 `iam:GetRole` 操作的策略。拥有该策略的用户可以从 AWS Management Console AWS CLI、或 AWS API 获取角色信息。

基于身份的策略

基于身份的策略是可附加到身份（如 IAM 用户、用户组或角色）的 JSON 权限策略文档。这些策略控制用户和角色可在何种条件下对哪些资源执行哪些操作。要了解如何创建基于身份的策略，请参阅《IAM 用户指南》中的创建 IAM policy。

基于身份的策略可以进一步归类为内联策略或托管策略。内联策略直接嵌入单个用户、组或角色中。托管策略是独立的策略，您可以将其附加到中的多个用户、群组和角色 AWS 账户。托管策略包括 AWS 托管策略和客户托管策略。要了解如何在托管式策略和内联策略之间进行选择，请参阅 IAM 用户指南中的在托管式策略与内联策略之间进行选择。

基于资源的策略

基于资源的策略是附加到资源的 JSON 策略文档。基于资源的策略的示例包括 IAM 角色信任策略和 Amazon S3 存储桶策略。在支持基于资源的策略的服务中，服务管理员可以使用它们来控制对特定资源的访问。对于在其中附加策略的资源，策略定义指定主体可以对该资源执行哪些操作以及在什么条件下执行。您必须在基于资源的策略中指定主体。委托人可以包括账户、用户、角色、联合用户或 AWS 服务。

基于资源的策略是位于该服务中的内联策略。您不能在基于资源的策略中使用 IAM 中的 AWS 托管策略。

访问控制列表 (ACL)

访问控制列表（ACL）控制哪些主体（账户成员、用户或角色）有权访问资源。ACL 与基于资源的策略类似，尽管它们不使用 JSON 策略文档格式。

Amazon S3 和 Amazon VPC 就是支持 ACL 的服务示例。 AWS WAF要了解有关 ACL 的更多信息，请参阅《Amazon Simple Storage Service 开发人员指南》中的访问控制列表（ACL）概览。

其他策略类型

AWS 支持其他不太常见的策略类型。这些策略类型可以设置更常用的策略类型向您授予的最大权限。

- 权限边界：权限边界是一个高级特征，用于设置基于身份的策略可以为 IAM 实体（IAM 用户或角色）授予的最大权限。您可为实体设置权限边界。这些结果权限是实体基于身份的策略及其权限边界的交集。在 `Principal` 中指定用户或角色的基于资源的策略不受权限边界限制。任一项策略中的显式拒绝将覆盖允许。有关权限边界的更多信息，请参阅《IAM 用户指南》中的 IAM 实体的权限边界。
- 服务控制策略 (SCP)-SCP 是 JSON 策略，用于指定组织或组织单位 (OU) 的最大权限。 AWS Organizations AWS Organizations 是一项用于对您的企业拥有的多 AWS 账户 项进行分组和集中管理的服务。如果在组织内启用了所有功能，则可对任意或全部账户应用服务控制策略 (SCP)。SCP 限制成员账户中的实体（包括每个 AWS 账户根用户实体）的权限。有关 Organizations 和 SCP 的更多信息，请参阅《AWS Organizations 用户指南》中的 SCP 的工作原理。
- 会话策略 – 会话策略是当您以编程方式为角色或联合用户创建临时会话时作为参数传递的高级策略。结果会话的权限是用户或角色的基于身份的策略和会话策略的交集。权限也可以来自基于资源的

策略。任一项策略中的显式拒绝将覆盖允许。有关更多信息，请参阅《IAM 用户指南》中的会话策略。

多个策略类型

当多个类型的策略应用于一个请求时，生成的权限更加复杂和难以理解。要了解在涉及多种策略类型时如何 AWS 确定是否允许请求，请参阅 IAM 用户指南中的策略评估逻辑。

Amazon Lookout for Vision 如何与 IAM 协同工作

在使用 IAM 管理对 Lookout for Vision 的访问之前，您应该了解哪些 IAM 功能可与 Lookout for Vision 一起使用。

能够与 Amazon Lookout for Vision 一起使用的 IAM 功能

IAM 功能	Lookout for Vision 支持
基于身份的策略	是
基于资源的策略	否
策略操作	是
策略资源	是
策略条件键（特定于服务）	是
ACL	否
ABAC（策略中的标签）	部分
临时凭证	是
转发访问会话（FAS）	是
服务角色	否
服务相关角色	否

要全面了解 Lookout for Vision AWS 和其他服务如何与大多数 IAM 功能配合使用，AWS 请参阅 IAM 用户指南中与 IAM 配合使用的服务。

Lookout for Vision 基于身份的策略

支持基于身份的策略	是

基于身份的策略是可附加到身份（如 IAM 用户、用户组或角色）的 JSON 权限策略文档。这些策略控制用户和角色可在何种条件下对哪些资源执行哪些操作。要了解如何创建基于身份的策略，请参阅《IAM 用户指南》中的创建 IAM policy。

通过使用 IAM 基于身份的策略，您可以指定允许或拒绝的操作和资源以及允许或拒绝操作的条件。您无法在基于身份的策略中指定主体，因为它适用于其附加的用户或角色。要了解可在 JSON 策略中使用的所有元素，请参阅《IAM 用户指南》中的 IAM JSON 策略元素引用。

Lookout for Vision 基于身份的策略示例

要查看 Lookout for Vision 基于身份的策略示例，请参阅 Amazon Lookout for Vision 基于身份的策略示例。

Lookout for Vision 中基于资源的策略

支持基于资源的策略	否

基于资源的策略是附加到资源的 JSON 策略文档。基于资源的策略的示例包括 IAM 角色信任策略和 Amazon S3 存储桶策略。在支持基于资源的策略的服务中，服务管理员可以使用它们来控制对特定资源的访问。对于在其中附加策略的资源，策略定义指定主体可以对该资源执行哪些操作以及在什么条件下执行。您必须在基于资源的策略中指定主体。委托人可以包括账户、用户、角色、联合用户或 AWS 服务。

要启用跨账户存取，您可以将整个账户或其他账户中的 IAM 实体指定为基于资源的策略中的主体。将跨账户主体添加到基于资源的策略只是建立信任关系工作的一半而已。当委托人和资源处于不同位置时 AWS 账户，可信账户中的 IAM 管理员还必须向委托人实体（用户或角色）授予访问资源的权限。他们通过将基于身份的策略附加到实体以授予权限。但是，如果基于资源的策略向同一个账户中的主体授予访问权限，则不需要额外的基于身份的策略。有关更多信息，请参阅 IAM 用户指南中的跨账户访问 IAM 中的资源。

Lookout for Vision 的策略操作

支持策略操作	是

管理员可以使用 AWS JSON 策略来指定谁有权访问什么。也就是说，哪个主体可以对什么资源执行操作，以及在什么条件下执行。

JSON 策略的 `Action` 元素描述可用于在策略中允许或拒绝访问的操作。策略操作通常与关联的 AWS API 操作同名。有一些例外情况，例如没有匹配 API 操作的仅限权限 操作。还有一些操作需要在策略中执行多个操作。这些附加操作称为相关操作。

在策略中包含操作以授予执行关联操作的权限。

要查看 Lookout for Vision 操作的列表，请参阅服务授权参考 中的 Amazon Lookout for Vision 定义的操作。

Lookout for Vision 中的策略操作在操作名称之前使用以下前缀：

```
lookoutvision
```

要在单个语句中指定多项操作，请使用逗号将它们隔开。

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

要查看 Lookout for Vision 基于身份的策略示例，请参阅 Amazon Lookout for Vision 基于身份的策略示例。

Lookout for Vision 的策略资源

支持策略资源	是

管理员可以使用 AWS JSON 策略来指定谁有权访问什么。也就是说，哪个主体 可以对什么资源执行操作，以及在什么条件下执行。

Resource JSON 策略元素指定要向其应用操作的一个或多个对象。语句必须包含 Resource 或 NotResource 元素。作为最佳实践，请使用其 Amazon 资源名称 (ARN) 指定资源。对于支持特定资源类型（称为资源级权限）的操作，您可以执行此操作。

对于不支持资源级权限的操作（如列出操作），请使用通配符 (*) 指示语句应用于所有资源。

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

要查看 Lookout for Vision 资源类型及其 ARN 的列表，请参阅服务授权参考 中的 Amazon Lookout for Vision 定义的资源。要了解您可以对哪些操作指定每个资源的 ARN，请参阅 Amazon Lookout for Vision 定义的操作。

要查看 Lookout for Vision 基于身份的策略示例，请参阅 Amazon Lookout for Vision 基于身份的策略示例。

Lookout for Vision 的策略条件键

支持特定于服务的策略条件键	是

管理员可以使用 AWS JSON 策略来指定谁有权访问什么。也就是说，哪个主体可以对什么资源执行操作，以及在什么条件下执行。

在 Condition 元素（或 Condition 块）中，可以指定语句生效的条件。Condition 元素是可选的。您可以创建使用条件运算符（例如，等于或小于）的条件表达式，以使策略中的条件与请求中的值相匹配。

如果您在一个语句中指定多个 Condition 元素，或在单个 Condition 元素中指定多个键，则 AWS 使用逻辑 AND 运算评估它们。如果您为单个条件键指定多个值，则使用逻辑OR运算来 AWS 评估条件。在授予语句的权限之前必须满足所有的条件。

在指定条件时，您也可以使用占位符变量。例如，只有在使用 IAM 用户名标记 IAM 用户时，您才能为其授予访问资源的权限。有关更多信息，请参阅《IAM 用户指南》中的 IAM policy 元素：变量和标签。

AWS 支持全局条件密钥和特定于服务的条件密钥。要查看所有 AWS 全局条件键，请参阅 IAM 用户指南中的AWS 全局条件上下文密钥。

有关 Lookout for Vision 条件键的列表，请参阅服务授权参考 中的 Amazon Lookout for Vision 的条件键。要了解您可以对哪些操作和资源使用条件键，请参阅 Amazon Lookout for Vision 定义的操作。

要查看 Lookout for Vision 基于身份的策略示例，请参阅 Amazon Lookout for Vision 基于身份的策略示例。

Lookout for Vision 中的 ACL

支持 ACL	否

访问控制列表 (ACL) 控制哪些主体（账户成员、用户或角色）有权访问资源。ACL 与基于资源的策略类似，尽管它们不使用 JSON 策略文档格式。

用于 Lookout for Vision 的 ABAC

支持 ABAC（策略中的标签）	部分

基于属性的访问权限控制（ABAC）是一种授权策略，该策略基于属性来定义权限。在中 AWS，这些属性称为标签。您可以向 IAM 实体（用户或角色）和许多 AWS 资源附加标签。标记实体和资源是 ABAC 的第一步。然后设计 ABAC 策略，以在主体的标签与他们尝试访问的资源标签匹配时允许操作。

ABAC 在快速增长的环境中非常有用，并在策略管理变得繁琐的情况下可以提供帮助。

要基于标签控制访问，您需要使用 `aws:ResourceTag/key-name`、`aws:RequestTag/key-name` 或 `aws:TagKeys` 条件键在策略的条件元素中提供标签信息。

如果某个服务对于每种资源类型都支持所有这三个条件键，则对于该服务，该值为是。如果某个服务仅对于部分资源类型支持所有这三个条件键，则该值为部分。

有关 ABAC 的更多信息,请参阅《IAM 用户指南》中的什么是 ABAC？。要查看设置 ABAC 步骤的教程，请参阅《IAM 用户指南》中的使用基于属性的访问权限控制（ABAC）。

对 Lookout for Vision 使用临时凭证

支持临时凭证	是

当你使用临时证书登录时，有些 AWS 服务 不起作用。有关更多信息，包括哪些 AWS 服务 适用于临时证书，请参阅 IAM 用户指南中的AWS 服务 与 IAM 配合使用的信息。

如果您使用除用户名和密码之外的任何方法登录，则 AWS Management Console 使用的是临时证书。例如，当您 AWS 使用公司的单点登录 (SSO) 链接进行访问时，该过程会自动创建临时证书。当您以用户身份登录控制台，然后切换角色时，您还会自动创建临时凭证。有关切换角色的更多信息，请参阅《IAM 用户指南》中的 切换到角色（控制台）。

您可以使用 AWS CLI 或 AWS API 手动创建临时证书。然后，您可以使用这些临时证书进行访问 AWS。 AWS 建议您动态生成临时证书，而不是使用长期访问密钥。有关更多信息，请参阅 IAM 中的临时安全凭证。

Lookout for Vision 的转发访问会话

支持转发访问会话 (FAS)	是

当您使用 IAM 用户或角色在中执行操作时 AWS，您被视为委托人。使用某些服务时，您可能会执行一个操作，然后此操作在其他服务中启动另一个操作。FAS 使用调用委托人的权限以及 AWS 服务 向下游服务发出请求的请求。 AWS 服务只有当服务收到需要与其他 AWS 服务 或资源交互才能完成的请求时，才会发出 FAS 请求。在这种情况下，您必须具有执行这两个操作的权限。有关发出 FAS 请求时的策略详情，请参阅转发访问会话。

Lookout for Vision 的服务角色

支持服务角色	否

服务角色是由一项服务担任、代表您执行操作的 IAM 角色。IAM 管理员可以在 IAM 中创建、修改和删除服务角色。有关更多信息，请参阅《IAM 用户指南》中的创建向 AWS 服务委派权限的角色。

> **Warning**
>
> 更改服务角色的权限可能会破坏 Lookout for Vision 的功能。仅当 Lookout for Vision 提供相关指导的情况下，才应编辑服务角色。

Lookout for Vision 的服务相关角色

支持服务相关角色	否

服务相关角色是一种与服务相关联的 AWS 服务服务角色。服务可以代入代表您执行操作的角色。服务相关角色出现在您的中 AWS 账户 ，并且归服务所有。IAM 管理员可以查看但不能编辑服务相关角色的权限。

有关创建或管理服务相关角色的详细信息，请参阅能够与 IAM 搭配使用的AWS 服务。在表中查找服务相关角色列中包含 Yes 的表。选择是链接以查看该服务的服务相关角色文档。

Amazon Lookout for Vision 基于身份的策略示例

默认情况下，用户和角色没有创建或修改 Lookout for Vision 资源的权限。他们也无法使用 AWS Management Console、 AWS Command Line Interface (AWS CLI) 或 AWS API 执行任务。要授予用户对所需资源执行操作的权限，IAM 管理员可以创建 IAM policy。管理员随后可以向角色添加 IAM 策略，用户可以代入角色。

要了解如何使用这些示例 JSON 策略文档创建基于 IAM 身份的策略，请参阅 IAM 用户指南中的 创建 IAM 策略。

有关 Lookout for Vision 定义的操作和资源类型的详细信息，包括每种资源类型的 ARN 格式，请参阅服务授权参考 中的 Amazon Lookout for Vision 的操作、资源和条件键。

主题

- 策略最佳实践
- 访问单个 Amazon Lookout for Vision 项目
- 基于标签的策略示例

策略最佳实践

基于身份的策略决定着用户是否可以创建、访问或删除您账户中的 Lookout for Vision 资源。这些操作可能会使 AWS 账户产生成本。创建或编辑基于身份的策略时，请遵循以下指南和建议：

- 开始使用 AWS 托管策略并转向最低权限权限 — 要开始向用户和工作负载授予权限，请使用为许多常见用例授予权限的AWS 托管策略。它们在你的版本中可用 AWS 账户。我们建议您通过定义针对

您的用例的 AWS 客户托管策略来进一步减少权限。有关更多信息，请参阅《IAM 用户指南》中的 AWS 托管策略 或 工作职能的AWS 托管策略。

- 应用最低权限 – 在使用 IAM policy 设置权限时，请仅授予执行任务所需的权限。为此，您可以定义在特定条件下可以对特定资源执行的操作，也称为最低权限许可。有关使用 IAM 应用权限的更多信息，请参阅《IAM 用户指南》中的 IAM 中的策略和权限。
- 使用 IAM policy 中的条件进一步限制访问权限 – 您可以向策略添加条件来限制对操作和资源的访问。例如，您可以编写策略条件来指定必须使用 SSL 发送所有请求。如果服务操作是通过特定的方式使用的，则也可以使用条件来授予对服务操作的访问权限 AWS 服务，例如 AWS CloudFormation。有关更多信息，请参阅《IAM 用户指南》中的 IAM JSON 策略元素：条件 。
- 使用 IAM Access Analyzer 验证您的 IAM policy，以确保权限的安全性和功能性 – IAM Access Analyzer 会验证新策略和现有策略，以确保策略符合 IAM policy语言 (JSON) 和 IAM 最佳实践。IAM Access Analyzer 提供 100 多项策略检查和可操作的建议，以帮助您制定安全且功能性强的策略。有关更多信息，请参阅《IAM 用户指南》中的 IAM Acess Analyzer 策略验证。
- 需要多重身份验证 (MFA)-如果 AWS 账户您的场景需要 IAM 用户或根用户，请启用 MFA 以提高安全性。若要在调用 API 操作时需要 MFA，请将 MFA 条件添加到您的策略中。有关更多信息，请参阅《IAM 用户指南》中的 配置受 MFA 保护的 API 访问。

有关 IAM 中的最佳实操的更多信息，请参阅《IAM 用户指南》中的 IAM 中的安全最佳实操。

访问单个 Amazon Lookout for Vision 项目

在本示例中，您想向 AWS 账户中的用户授予访问您的 Amazon Lookout for Vision 项目的权限。

```
{
    "Sid": "SpecificProjectOnly",
    "Effect": "Allow",
    "Action": [
        "lookoutvision:DetectAnomalies"
    ],
    "Resource": "arn:aws:lookoutvision:us-east-1:123456789012:model/myproject/*"
}
```

基于标签的策略示例

基于标签的策略是 JSON 策略文档，其中指定主体可对所标记的资源执行哪些操作。

使用标签访问资源

此示例策略将为您亚马逊云科技账户中的一个用户或角色授予权限，使之能够对任何用键 stage 和值 production 标记的模型使用 DetectAnomalies 操作。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "LookoutVision:DetectAnomalies"
            ],
            "Resource": "*",
            "Condition": {
                "StringEquals": {
                    "aws:ResourceTag/stage": "production"
                }
            }
        }
   ]
}
```

使用标签拒绝访问特定的 Amazon Lookout for Vision 操作

此示例策略将拒绝为您亚马逊云科技账户中的一个用户或角色授权，使之无法对任何用键 stage 和值 production 标记的模型调用 DeleteModel 或 StopModel 操作。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Deny",
            "Action": [
                "LookoutVision:DeleteModel",
                "LookoutVision:StopModel"
            ],
            "Resource": "*",
            "Condition": {
                "StringEquals": {
                    "aws:ResourceTag/stage": "production"
                }
            }
```

```
        }
    ]
}
```

适用于 Amazon Lookout for Vision 的 AWS 托管式策略

AWS 托管式策略是由 AWS 创建和管理的独立策略。AWS 托管式策略旨在为许多常见用例提供权限，以便您可以开始为用户、组和角色分配权限。

请记住，AWS 托管式策略可能不会为您的特定使用场景授予最低权限，因为它们可供所有 AWS 客户使用。我们建议通过定义特定于您的使用场景的客户管理型策略来进一步减少权限。

您无法更改 AWS 托管式策略中定义的权限。如果 AWS 更新在 AWS 托管式策略中定义的权限，则更新会影响该策略所附加到的所有主体身份（用户、组和角色）。当新的 AWS 服务启动或新的 API 操作可用于现有服务时，AWS 最有可能更新 AWS 托管式策略。

有关更多信息，请参阅《IAM 用户指南》中的 AWS 托管式策略。

亚马逊云科技托管式策略：AmazonLookoutVisionReadOnlyAccess

使用 `AmazonLookoutVisionReadOnlyAccess` 政策可允许用户通过以下 Amazon Lookout for Vision 操作（SDK 操作），对 Amazon Lookout for Vision（及其依赖项）进行只读访问。例如，您可以使用 `DescribeModel` 获取有关现有模型的信息。

- DescribeDataset
- DescribeModel
- DescribeModelPackagingJob
- DescribeProject
- ListDatasetEntries
- ListModelPackagingJobs
- ListModels
- ListProjects
- ListTagsForResource

要调用只读操作，用户不需要 Amazon S3 桶权限。但是，操作响应中可能包含对 Amazon S3 桶的引用。例如，在 `ListDatasetEntries` 的响应中，`source-ref` 条目便是引用 Amazon S3 桶中的图像。如果用户需要访问所引用的桶，请添加 Amazon S3 桶权限。例如，用户可能想要下载 `source-ref` 字段所引用的图像。有关更多信息，请参阅 授予 Amazon S3 桶权限。

您可以将 `AmazonLookoutVisionReadOnlyAccess` 策略附加得到 IAM 身份。

权限详细信息

此策略包含以下权限。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "LookoutVisionReadOnlyAccess",
            "Effect": "Allow",
            "Action": [
                "lookoutvision:DescribeDataset",
                "lookoutvision:DescribeModel",
                "lookoutvision:DescribeProject",
                "lookoutvision:DescribeModelPackagingJob",
                "lookoutvision:ListDatasetEntries",
                "lookoutvision:ListModels",
                "lookoutvision:ListProjects",
                "lookoutvision:ListTagsForResource",
                "lookoutvision:ListModelPackagingJobs"
            ],
            "Resource": "*"
        }
    ]
}
```

亚马逊云科技托管式策略：AmazonLookoutVisionFullAccess

使用 `AmazonLookoutVisionFullAccess` 政策可允许用户通过 Amazon Lookout for Vision 操作（SDK 操作），完全访问 Amazon Lookout for Vision（及其依赖项）。例如，您无需使用 Amazon Lookout for Vision 控制台即可训练模型。有关更多信息，请参阅操作。

要创建数据集 (CreateDataset) 或创建模型 (CreateModel)，用户必须对存储数据集图像、Amazon SageMaker Ground Truth 清单文件和训练输出的 Amazon S3 存储桶拥有完全访问权限。有关更多信息，请参阅 步骤 2：设置权限。

您也可以通过使用 AmazonLookoutVisionConsoleFullAccess 政策，为 Amazon Lookout for Vision SDK 操作赋予权限。

您可以将 AmazonLookoutVisionFullAccess 策略附加得到 IAM 身份。

权限详细信息

此策略包含以下权限。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "LookoutVisionFullAccess",
            "Effect": "Allow",
            "Action": [
                "lookoutvision:*"
            ],
            "Resource": "*"
        }
    ]
}
```

亚马逊云科技托管式策略：AmazonLookoutVisionConsoleFullAccess

使用 AmazonLookoutVisionFullAccess 政策可允许用户完全访问 Amazon Lookout for Vision 控制台、操作（SDK 操作），以及该服务具有的任何依赖项。有关更多信息，请参阅 开始使用 Amazon Lookout for Vision。

LookoutVisionConsoleFullAccess 政策中包含了对您的 Amazon Lookout for Vision 控制台桶的权限。有关控制台桶的信息，请参阅步骤 3：创建控制台桶。要将数据集、图像和 Amazon SageMaker Ground Truth 清单文件存储在其他 Amazon S3 桶中，用户需要额外的权限。有关更多信息，请参阅 the section called “设置 Amazon S3 桶权限”。

您可以将 AmazonLookoutVisionConsoleFullAccess 策略附加得到 IAM 身份。

权限组

此策略根据提供的权限集分为多个语句：

- LookoutVisionFullAccess：允许进行访问以执行所有 Lookout for Vision 操作。
- LookoutVisionConsoleS3BucketSearchAccess：允许列出调用者拥有的所有 Amazon S3 桶。Lookout for Vision 使用此操作来识别亚马逊云科技区域特定的 Lookout for Vision 控制台桶（如果调用者的账户中存在该桶）。
- LookoutVisionConsoleS3BucketFirstUseSetupAccessPermissions：允许创建和配置与 Lookout for Vision 控制台桶名称模式相匹配的 Amazon S3 桶。Lookout for Vision 在找不到区域特定的 Lookout for Vision 控制台桶时，将会使用这些操作来创建和配置一个这样的桶。
- LookoutVisionConsoleS3BucketAccess：允许对与 Lookout for Vision 控制台桶名称模式相匹配的 Amazon S3 桶执行依赖性 Amazon S3 操作。当从 Amazon S3 桶中创建数据集，以及当启动试用检测任务时，Lookout for Vision 使用 s3:ListBucket 来搜索图像对象。作为以下操作的组成部分，Lookout for Vision 会使用 s3:GetBucketLocation 和 s3:GetBucketVersioning 来验证桶 AWS 区域、所有者和配置：
  - 创建数据集
  - 训练模型
  - 启动试用检测任务
  - 执行试用检测反馈

  LookoutVisionConsoleS3ObjectAccess：允许在与 Lookout for Vision 控制台桶名称模式相匹配的桶中，读取和写入 Amazon S3 对象。Lookout for Vision 使用这些操作在控制台图库视图中显示图像，并上传用于数据集的新图像。此外，这些权限还允许 Lookout for Vision 在创建数据集、训练模型、启动试用检测任务和执行试用检测反馈时写出元数据。
- LookoutVisionConsoleDatasetLabelingToolsAccess：允许执行依赖性 Amazon SageMaker GroundTruth 标注操作。Lookout for Vision 使用这些操作来扫描 S3 桶中的图像，创建 GroundTruth 清单文件，并且用验证标签来注释试用检测任务结果。
- LookoutVisionConsoleDashboardAccess：允许读取 Amazon CloudWatch 指标。Lookout for Vision 使用这些操作来填充控制面板图形和检测到的异常的统计数据。
- LookoutVisionConsoleTagSelectorAccess：允许读取账户特定的标签键和标签值建议。Lookout for Vision 使用这些权限在管理标签控制台页面中为标签键和标签值提供建议。

- LookoutVisionConsoleKmsKeySelectorAccess：允许列出 AWS Key Management Service (KMS) 密钥和别名。Amazon Lookout for Vision 使用此权限，针对某些 Lookout for Vision 操作（支持使用客户管理的 KMS 密钥进行加密），在建议的标签选择中填充 KMS 密钥。

```
{
    "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": "*"
        }
    ]
}
```

亚马逊云科技托管式策略：AmazonLookoutVisionConsoleReadOnlyAccess

使用 AmazonLookoutVisionConsoleReadOnlyAccess 政策可允许用户以只读方式访问 Amazon Lookout for Vision 控制台、操作（SDK 操作），以及该服务具有的任何依赖项。

AmazonLookoutVisionConsoleReadOnlyAccess 政策中包含了对 Amazon Lookout for Vision 控制台桶的 Amazon S3 权限。如果数据集图像和 Amazon SageMaker Ground Truth 清单文件位于其他 Amazon S3 桶中，则用户需要额外的权限。有关更多信息，请参阅 the section called “设置 Amazon S3 桶权限”。

您可以将 AmazonLookoutVisionConsoleReadOnlyAccess 策略附加得到 IAM 身份。

权限组

此策略根据提供的权限集分为多个语句：

- LookoutVisionReadOnlyAccess：允许进行访问以执行只读 Lookout for Vision 操作。
- LookoutVisionConsoleS3BucketSearchAccess：允许列出调用者拥有的所有 S3 桶。Lookout for Vision 使用此操作来识别亚马逊云科技区域特定的 Lookout for Vision 控制台桶（如果调用者的账户中有）。
- LookoutVisionConsoleS3ObjectReadAccess：允许读取 Lookout for Vision 控制台桶中的 Amazon S3 对象和 Amazon S3 对象版本。Lookout for Vision 使用这些操作来显示数据集、模型和试用检测中的图像。
- LookoutVisionConsoleDashboardAccess：允许读取 Amazon CloudWatch 指标。Lookout for Vision 使用这些操作来为控制面板图形和检测到的异常填充统计数据。

```
{
```

```
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "LookoutVisionReadOnlyAccess",
            "Effect": "Allow",
            "Action": [
                "lookoutvision:DescribeDataset",
                "lookoutvision:DescribeModel",
                "lookoutvision:DescribeProject",
                "lookoutvision:DescribeTrialDetection",
                "lookoutvision:DescribeModelPackagingJob",
                "lookoutvision:ListDatasetEntries",
                "lookoutvision:ListModels",
                "lookoutvision:ListProjects",
                "lookoutvision:ListTagsForResource",
                "lookoutvision:ListTrialDetections",
                "lookoutvision:ListModelPackagingJobs"
            ],
            "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 对 AWS 托管式策略的更新

查看自 Lookout for Vision 开始跟踪更改以来，适用于该服务的 AWS 托管式策略的更新详细信息。如需自动提醒此页面发生的更改，请订阅 Lookout for Vision 文档历史记录页面上的 RSS 源。

添加了模型打包操作	Amazon Lookout for Vision 向 AmazonLookoutVisionFullAccess 和 AmazonLookoutVisionConsoleFullAccess 策略中添加了以下模型打包操作： • DescribeModelPackagingJob • ListModelPackagingJobs • StartModelPackagingJob Amazon Lookout for Vision 向 AmazonLookoutVisionReadOnlyAccess 和 AmazonLookoutVisionConsoleReadOnlyAccess 策略中添加了以下模型打包操作： • DescribeModelPackagingJob • ListModelPackagingJobs	2021 年 12 月 7 日
添加了新策略	Amazon Lookout for Vision 添加了以下策略。 • AmazonLookoutVisionReadOnlyAccess • AmazonLookoutVisionFullAccess • AmazonLookoutVisionConsoleFullAccess • AmazonLookoutVisionConsoleReadOnlyAccess	2021 年 5 月 11 日
Lookout for Vision 开始跟踪更改	Amazon Lookout for Vision 开始跟踪其 AWS 托管式策略的更改。	2021 年 3 月 1 日

Amazon Lookout for Vision 身份和访问权限问题排查

使用以下信息可帮助您诊断和修复在使用 Lookout for Vision 和 IAM 时可能遇到的常见问题。

主题

- 我未被授权在 Lookout for Vision 中执行某项操作
- 我无权执行 iam：PassRole
- 我想允许我以外的人访问我的 Lookout for Vision 资源 AWS 账户

我未被授权在 Lookout for Vision 中执行某项操作

如果您收到错误提示，表明您无权执行某个操作，则您必须更新策略以允许执行该操作。

当 `mateojackson` IAM 用户尝试使用控制台查看有关虚构 *`my-example-widget`* 资源的详细信息，但不拥有虚构 `lookoutvision:`*`GetWidget`* 权限时，会发生以下示例错误。

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

在此情况下，必须更新 `mateojackson` 用户的策略，以允许使用 `lookoutvision:`*`GetWidget`* 操作访问 *`my-example-widget`* 资源。

如果您需要帮助，请联系您的 AWS 管理员。您的管理员是提供登录凭证的人。

我无权执行 iam：PassRole

如果您收到一个错误，表明您无权执行 `iam:PassRole` 操作，则必须更新策略以允许您将角色传递给 Lookout for Vision。

有些 AWS 服务 允许您将现有角色传递给该服务，而不是创建新的服务角色或服务相关角色。为此，您必须具有将角色传递到服务的权限。

当名为 `marymajor` 的 IAM 用户尝试使用控制台在 Lookout for Vision 中执行操作时，会发生以下示例错误。但是，服务必须具有服务角色所授予的权限才可执行此操作。Mary 不具有将角色传递到服务的权限。

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

在这种情况下，必须更新 Mary 的策略以允许她执行 `iam:PassRole` 操作。

如果您需要帮助，请联系您的 AWS 管理员。您的管理员是提供登录凭证的人。

我想允许我以外的人访问我的 Lookout for Vision 资源 AWS 账户

您可以创建一个角色，以便其他账户中的用户或您组织外的人员可以使用该角色来访问您的资源。您可以指定谁值得信赖，可以担任角色。对于支持基于资源的策略或访问控制列表（ACL）的服务，您可以使用这些策略向人员授予对您的资源的访问权。

要了解更多信息，请参阅以下内容：

- 要了解 Lookout for Vision 是否支持这些功能，请参阅 Amazon Lookout for Vision 如何与 IAM 协同工作。
- 要了解如何提供对您拥有的资源的访问权限 AWS 账户 ，请参阅 IAM 用户指南中的向您拥有 AWS 账户 的另一个 IAM 用户提供访问权限。
- 要了解如何向第三方提供对您的资源的访问权限 AWS 账户，请参阅 IAM 用户指南中的向第三方提供访问权限。 AWS 账户
- 要了解如何通过联合身份验证提供访问权限，请参阅《IAM 用户指南》中的为经过外部身份验证的用户（联合身份验证）提供访问权限。
- 要了解使用角色和基于资源的策略进行跨账户访问的区别，请参阅 IA M 用户指南中的跨账户资源访问。

Amazon Lookout for Vision 的合规性验证

作为 AWS 多项合规计划的一部分，第三方审计师评估亚马逊 Lookout for Vision 的安全与合规性。Amazon Lookout for Vision 符合通用数据保护条例（GDPR）。

要了解是否属于特定合规计划的范围，请参阅AWS 服务 “按合规计划划分的范围” ”，然后选择您感兴趣的合规计划。 AWS 服务 有关一般信息，请参阅AWS 合规计划AWS。

您可以使用下载第三方审计报告 AWS Artifact。有关更多信息，请参阅中的 “下载报告” 中的 “ AWS Artifact。

您在使用 AWS 服务 时的合规责任取决于您的数据的敏感性、贵公司的合规目标以及适用的法律和法规。 AWS 提供了以下资源来帮助实现合规性：

- 安全与合规性快速入门指南 — 这些部署指南讨论了架构注意事项，并提供了在这些基础上 AWS 部署以安全性和合规性为重点的基准环境的步骤。

- 在 A@@ mazon Web Services 上构建 HIPAA 安全与合规架构 — 本白皮书描述了各公司如何使用 AWS 来创建符合 HIPAA 资格的应用程序。

> **Note**
>
> 并非所有 AWS 服务 人都符合 HIPAA 资格。有关更多信息，请参阅符合 HIPAA 要求的服务参考。

- AWS 合规资源AWS — 此工作簿和指南集可能适用于您所在的行业和所在地区。
- AWS 客户合规指南 — 从合规角度了解责任共担模式。这些指南总结了保护的最佳实践， AWS 服务并将指南映射到跨多个框架（包括美国国家标准与技术研究院 (NIST)、支付卡行业安全标准委员会 (PCI) 和国际标准化组织 (ISO)）的安全控制。
- 使用AWS Config 开发人员指南中的规则评估资源 — 该 AWS Config 服务评估您的资源配置在多大程度上符合内部实践、行业准则和法规。
- AWS Security Hub— 这 AWS 服务 提供了您内部安全状态的全面视图 AWS。Security Hub 通过安全控件评估您的 AWS 资源并检查其是否符合安全行业标准和最佳实践。有关受支持服务及控件的列表，请参阅 Security Hub 控件参考。
- Amazon GuardDuty — 它通过监控您的 AWS 账户环境中是否存在可疑和恶意活动，来 AWS 服务检测您的工作负载、容器和数据面临的潜在威胁。 GuardDuty 通过满足某些合规性框架规定的入侵检测要求，可以帮助您满足各种合规性要求，例如 PCI DSS。
- AWS Audit Manager— 这 AWS 服务 可以帮助您持续审计 AWS 使用情况，从而简化风险管理以及对法规和行业标准的合规性。

Amazon Lookout for Vision 中的韧性

AWS全球基础设施围绕AWS区域和可用区构建。AWS区域提供多个在物理上独立且隔离的可用区，这些可用区通过延迟低、吞吐量高且冗余性高的网络连接在一起。利用可用区，您可以设计和操作在可用区之间无中断地自动实现故障转移的应用程序和数据库。与传统的单个或多个数据中心基础设施相比，可用区具有更高的可用性、容错性和可扩展性。

有关AWS区域和可用区的更多信息，请参阅AWS全球基础设施。

Amazon Lookout for Vision 中的基础设施安全性

作为一项托管式服务，Amazon Lookout for Vision 由 AWS 全球网络安全提供保护。有关 AWS 安全服务以及 AWS 如何保护基础架构的信息，请参阅 AWS 云安全。要按照基础设施安全最佳实践设计您的 AWS 环境，请参阅《安全性支柱 AWS Well-Architected Framework》中的 基础设施保护。

您可以使用 AWS 的已发布 API 调用，通过网络访问 Lookout for Vision。客户端必须支持以下内容：

- 传输层安全性协议（TLS） 我们要求使用 TLS 1.2，建议使用 TLS 1.3。
- 具有完全向前保密 (PFS) 的密码套件，例如 DHE（临时 Diffie-Hellman）或 ECDHE（临时椭圆曲线 Diffie-Hellman）。大多数现代系统（如 Java 7 及更高版本）都支持这些模式。

此外，必须使用访问密钥 ID 和与 IAM 委托人关联的秘密访问密钥来对请求进行签名。或者，您可以使用 AWS Security Token Service (AWS STS) 生成临时安全凭证来对请求进行签名。

监控 Amazon Lookout for Vision

就保持 Amazon Lookout for Vision 和其他亚马逊云科技解决方案的可靠性、可用性和性能而言，监控是其中一个重要组成部分。亚马逊云科技提供了以下监控工具，用于监测 Lookout for Vision，报告错误情况，并在适当时采取自动化措施：

- Amazon CloudWatch 可实时监控您的AWS资源以及您在AWS上运行的应用程序。您可以收集和跟踪指标，创建自定义的控制面板，以及设置警报以在指定的指标达到您指定的阈值时通知您或采取措施。例如，您可以具有 Amazon EC2 实例的 CloudWatch 跟踪 CPU 使用率或其他指标并且在需要时自动启动新实例。有关更多信息，请参阅 Amazon CloudWatch 用户指南。
- Amazon CloudWatch Logs 使您能够监控、存储和访问来自 Amazon EC2 实例、CloudTrail 和其他来源的日志文件。CloudWatch Logs 可以监控日志文件中的信息，并在达到特定阈值时通知您。您还可以在高持久性存储中检索您的日志数据。有关更多信息，请参阅 Amazon CloudWatch Logs 用户指南。
- Amazon EventBridge 可用来自动执行您的AWS服务并自动响应系统事件，如应用程序可用性问题或资源更改。AWS 服务中的事件将近乎实时传输到 EventBridge。您可以编写简单的规则来指示您关注的事件，并指示要在事件匹配规则时执行的自动化操作。有关更多信息，请参阅 Amazon EventBridge 用户指南。
- AWS CloudTrail 捕获由您的 AWS 账户或代表该账户发出的 API 调用和相关事件，并将日志文件传输到您指定的 Simple Storage Service（Amazon S3）存储桶。您可以标识哪些用户和账户调用了 AWS、从中发出调用的源 IP 地址以及调用的发生时间。有关更多信息，请参阅 AWS CloudTrail 用户指南。

使用 Amazon CloudWatch 监控 Lookout for Vision

您可以使用 CloudWatch 来监控 Lookout for Vision，它会收集原始数据，并将其处理成可读、近实时的指标。这些统计数据会保存 15 个月，从而使您能够访问历史信息，并能够更好地了解您的 Web 应用程序或服务的执行情况。此外，可以设置用于监测特定阈值的警报，并在达到相应阈值时发送通知或执行操作。有关更多信息，请参阅 Amazon CloudWatch 用户指南。

Lookout for Vision 服务会在 `AWS/LookoutVision` 命名空间中报告以下指标。

指标	描述
`DetectedAnomalyCount`	在项目中检测到的异常数量

指标	描述
	有效统计数据：Sum,Average 单位：计数
ProcessedImageCount	经历异常检测的图像总数 有效统计数据：Sum,Average 单位：计数
InvalidImageCount	无效的、无法返回结果的图像数量 有效统计数据：Sum,Average 单位：计数
SuccessfulRequestCount	成功的 API 调用数量 有效统计数据：Sum,Average 单位：计数
ErrorCount	API 错误数量 有效统计数据：Sum,Average 单位：计数
ThrottledCount	由于节流导致的 API 错误数量 有效统计数据：Sum,Average 单位：计数
Time	Lookout for Vision 计算异常检测所用的时间（以毫秒为单位） 有效统计数据：Data Samples,Average 单位：Average 统计数据以毫秒为单位，Data Samples 统计数据以计数为单位

指标	描述
MinInferenceUnits	StartModel 请求期间指定的最小推理单位数。 有效统计数据：Average 单位：计数
MaxInferenceUnits	StartModel 请求期间指定的最大推理单位数。 有效统计数据：Average 单位：计数
DesiredInferenceUnits	Lookout for Vision 扩大或缩小到的推理单位数量。 有效统计数据：Average 单位：计数
InServiceInferenceUnits	模型正在使用的推理单位数量。 有效统计数据：Average 建议您使用平均值统计数据，获取实例使用数量的 1 分钟平均值。 单位：计数

Lookout for Vision 指标支持以下维度。

维度	描述
ProjectName	您可以按项目拆分指标，以便查看哪些项目存在问题或需要更新。
ModelVersion	您可以按模型版本拆分指标，以便查看哪些模型存在问题或需要更新。

使用 AWS CloudTrail 记录 Lookout for Vision API 调用

Amazon Lookout for Vision 与 AWS CloudTrail 集成，后者是一项服务，可以提供 Lookout for Vision 中由用户、角色或AWS服务所采取的操作的记录。CloudTrail 会以事件形式捕获 Lookout for Vision 的所有 API 调用。所捕获的调用中包括来自 Lookout for Vision 控制台的调用，以及对 Lookout for Vision API 操作的代码调用。如果您创建跟踪记录，则可以使 CloudTrail 事件（包括 Lookout for Vision 的事件）持续传送到 Amazon S3 桶。如果您不配置跟踪，则仍可在 CloudTrail 控制台中的 Event history（事件历史记录）中查看最新事件。使用 CloudTrail 收集的信息，您可以确定向 Lookout for Vision 发出的请求、发出请求的 IP 地址、请求方、请求时间以及其他详细信息。

要了解有关 CloudTrail 的更多信息，请参阅《AWS CloudTrail 用户指南》。

CloudTrail 中的 Lookout for Vision 信息

在您创建 AWS 账户时，将在该账户上启用 CloudTrail。当 Lookout for Vision 中发生活动时，该活动将通过一个 CloudTrail 事件与其他 AWS 服务事件一同记录在事件历史记录中。您可以在 AWS 账户中查看、搜索和下载最新事件。有关更多信息，请参阅使用 CloudTrail 事件历史记录查看事件。

要持续记录 AWS 账户中的事件（包括 Lookout for Vision 的事件），请创建跟踪记录。通过跟踪，CloudTrail 可将日志文件传送至 Amazon S3 存储桶。默认情况下，在控制台中创建跟踪记录时，此跟踪记录应用于所有亚马逊云科技区域。此跟踪在 AWS 分区中记录所有区域中的事件，并将日志文件传送至您指定的 Amazon S3 存储桶。此外，您可以配置其他 AWS 服务，进一步分析在 CloudTrail 日志中收集的事件数据并采取行动。有关更多信息，请参阅下列内容：

- 创建跟踪概览
- CloudTrail 支持的服务和集成
- 为 CloudTrail 配置 Amazon SNS 通知
- 从多个区域接收 CloudTrail 日志文件和从多个账户接收 CloudTrail 日志文件

所有 Lookout for Vision 操作都由 CloudTrail 记录，并记录在 Lookout for Vision API 参考文档中。例如，对 `CreateProject`、`DetectAnomalies` 和 `StartModel` 操作的调用会在 CloudTrail 日志文件中生成条目。

如果您监控 Amazon Lookout for Vision API 调用，可能会看到对以下 API 的调用。

- lookoutvision:StartTriallDetection
- lookoutvision:ListTriallDetection

- lookoutvision:DescribeTrialDetection

Amazon Lookout for Vision 使用这些特殊调用来支持与试用检测相关的各种操作。有关更多信息，请参阅 通过试用检测任务验证您的模型。

每个事件或日志条目都包含有关生成请求的人员信息。身份信息可帮助您确定以下内容：

- 请求是使用根用户凭证还是 AWS Identity and Access Management 用户凭证发出的。
- 请求是使用角色还是联合身份用户的临时安全凭证发出的。
- 请求是否由其它 AWS 服务发出。

有关更多信息，请参阅 CloudTrail userIdentity 元素。

了解 Lookout for Vision 日志文件条目

跟踪是一种配置，可用于将事件作为日志文件传送到您指定的 Amazon S3 桶。CloudTrail 日志文件包含一个或多个日志条目。一个事件表示来自任何源的一个请求，包括有关所请求的操作、操作的日期和时间、请求参数等方面的信息。CloudTrail 日志文件不是公用 API 调用的有序堆栈跟踪，因此它们不会按任何特定顺序显示。

下面的示例显示了一个 CloudTrail 日志条目，该条目说明了 `CreateDataset` 操作。

```
{
  "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"
}
```

使用 AWS CloudFormation 创建 Amazon Lookout for Vision 资源

Amazon Lookout for Vision 与 AWS CloudFormation 集成，后者是一项服务，可帮助您对 AWS 资源进行建模和设置，这样您只需花较少的时间来创建和管理资源与基础设施。您可以创建一个模板来描述所需的所有 AWS 资源，然后 AWS CloudFormation 可为您预调配和配置这些资源。

您可以使用 AWS CloudFormation 来预调配和配置 Amazon Lookout for Vision 项目。

使用 AWS CloudFormation 时，可重复使用您的模板，以便一致且重复地设置 Lookout for Vision 项目。只需描述一次项目，然后就可以在多个亚马逊云科技账户和区域中反复预调配相同的项目。

Lookout for Vision 和 AWS CloudFormation 模板

要为 Lookout for Vision 和相关服务预调配并配置项目，您必须了解 AWS CloudFormation 模板。模板是 JSON 或 YAML 格式的文本文件。这些模板描述要在 AWS CloudFormation 堆栈中调配的资源。如果您不熟悉 JSON 或 YAML，可以在 AWS CloudFormation Designer 的帮助下开始使用 AWS CloudFormation 模板。有关更多信息，请参阅 AWS CloudFormation 用户指南中的什么是 AWS CloudFormation Designer？。

有关 Lookout for Vision 项目的引用信息，包括 JSON 和 YAML 模板示例，请参阅 LookoutVision 资源类型引用。

了解有关 AWS CloudFormation 的更多信息

要了解有关 AWS CloudFormation 的更多信息，请参阅以下资源：

- AWS CloudFormation
- AWS CloudFormation 用户指南
- AWS CloudFormation API 参考
- AWS CloudFormation 命令行界面用户指南

使用接口端点 (AWS PrivateLink) 访问 Amazon Lookout for Vision

您可以使用 AWS PrivateLink 在 VPC 和 Amazon Lookout for Vision 之间创建私有连接。您可以像 Lookout for Vision 就在您的 VPC 中一样访问它，无需使用互联网网关、NAT 设备、VPN 连接或 AWS Direct Connect 连接。您的 VPC 中的实例不需要公有 IP 地址即可访问 Lookout for Vision。

您可以通过创建由 AWS PrivateLink 提供支持的接口端点来建立此私有连接。我们将在您为接口终端节点启用的每个子网中创建一个终端节点网络接口。这些是请求者托管的网络接口，用作发往 Lookout for Vision 的流量的入口点。

有关更多信息，请参阅 AWS PrivateLink 指南中的通过 AWS PrivateLink 访问 AWS 服务。

Lookout for Vision VPC 端点的注意事项

在为 Lookout for Vision 设置接口端点之前，请检查 AWS PrivateLink 指南 中的注意事项。

Lookout for Vision 支持通过接口端点调用其所有 API 操作。

Lookout for Vision 不支持 VPC 端点策略。默认情况下，允许通过接口端点对 Lookout for Vision 进行完全访问。或者，您可以将安全组与端点网络接口相关联，以便控制通过接口端点流向 Lookout for Vision 的流量。

为 Lookout for Vision 创建接口 VPC 端点

您可以使用 Amazon VPC 控制台或 AWS Command Line Interface (AWS CLI)，为 Lookout for Vision 创建 VPC 端点。有关更多信息，请参阅 AWS PrivateLink 指南中的创建接口端点。

使用以下服务名称，为 Lookout for Vision 创建接口端点：

```
com.amazonaws.region.lookoutvision
```

如果为接口端点启用私有 DNS，则可使用 Lookout for Vision 的默认区域 DNS 名称向其发出 API 请求。例如，`lookoutvision.us-east-1.amazonaws.com`。

为 Lookout for Vision 创建 VPC 端点策略

端点策略是一种 IAM 资源，您可以将其附加到接口端点。默认端点策略允许通过接口端点完全访问 Lookout for Vision。要对从 VPC 访问 Lookout for Vision 的允许权限，请将自定义端点策略附加到接口端点。

端点策略指定以下信息：

- 可执行操作的主体（AWS 账户、IAM 用户和 IAM 角色）。
- 可执行的操作。
- 可对其执行操作的资源。

有关更多信息，请参阅 AWS PrivateLink 指南中的使用端点策略控制对服务的访问权限。

示例：适用于 Lookout for Vision 操作的 VPC 端点策略

以下是适用于 Lookout for Vision 的自定义端点策略示例。当您将此策略附加到接口端点时，它会规定有权访问该 VPC 接口端点的所有用户都可以为与 `myProject` 项目关联的 Lookout for Vision 模型 `myModel` 调用 `DetectAnomalies` API 操作。

```
{
   "Statement":[
      {
         "Principal":"*",
         "Effect":"Allow",
         "Action":[
            "lookoutvision:DetectAnomalies"
         ],
         "Resource": "arn:aws:lookoutvision:us-west-2:123456789012:model/myProject/
myModel"
      }
   ]
}
```

Amazon Lookout for Vision 中的配额

下表描述了 Amazon Lookout for Vision 中的现行配额。有关可更改的配额的信息，请参阅亚马逊云科技服务限额。

模型配额

以下配额适用于模型的测试、训练和功能。

资源	配额
支持的文件格式	PNG 和 JPEG 图像格式
Amazon S3 桶中图像文件的最小图像尺寸	64 x 64 像素
Amazon S3 桶中图像文件的最大图像尺寸	最大值为 4096 X 4096 像素。尺寸越小，上传速度越快。
项目中使用的图像文件的不同图像尺寸	数据集中的所有图像必须尺寸相同
Amazon S3 桶中图像的最大文件大小	8 MB
缺少标签	训练前必须将图像标注为正常或异常。训练期间会忽略未标注的图像。
训练数据集中标注为正常的图像的最小数量	具有单独训练数据集和测试数据集的项目为 10。具有单个数据集的项目为 20。
训练数据集中标注为异常的图像的最小数量	具有单独训练数据集和测试数据集的项目为 0。具有单个数据集的项目为 10。
分类训练数据集中的最大图像数量	16000
分类测试数据集中的最大图像数量	4,000
测试数据集中标注为正常的图像的最小数量	10
测试数据集中标注为异常的图像的最小数量	10

资源	配额
异常定位训练数据集中的最大图像数	8000
异常定位测试数据集中的最大图像数	800
试用检测数据集中的最大图像数	2000
最大数据集清单文件大小	1GB
模型中的最大训练数据集数量	1
最长训练时间	24 小时
最长测试时间	24 小时
项目中的最大异常标签数量	100
掩码图像上的最大异常标签数量	20
异常标签对应的最小图像数量。要进行计算，图像必须仅包含一种类型的异常标签。	单数据集项目为 20。具有单独训练数据集和测试数据集的项目为 10。

Amazon Lookout for Vision 文档历史记录

下表描述了 Amazon Lookout for Vision 开发者指南 每个发布版本中的重要修改。如需对此文档更新的通知，您可以订阅 RSS 源。

- 文档上次更新时间：2023 年 2 月 20 日

变更	说明	日期
添加了示例 Lambda 函数	通过示例展示如何使用 AWS Lambda 函数发现异常。有关更多信息，请参阅使用 Amazon Lambda 函数查找异常。	2023 年 2 月 20 日
更新了 AWS WAF 的 IAM 指南	更新了指南，使其符合 IAM 最佳实践。有关更多信息，请参阅 IAM 安全最佳实践。	2023 年 2 月 8 日
添加了数据集导出示例	添加了 Python 示例，用于展示如何使用 `ListDatasetEntries` 操作从 Amazon Lookout for Vision 项目中导出数据集。有关更多信息，请参阅从项目中导出数据集 (SDK)。	2022 年 12 月 2 日
更新了“开始使用”主题	更新了“开始使用”指南，用于展示如何使用示例数据集创建图像分割模型。有关更多信息，请参阅开始使用 Amazon Lookout for Vision。	2022 年 10 月 20 日
添加了问题排查主题	添加了模型训练问题排查主题。有关更多信息，请参阅模型训练问题排查。	2022 年 10 月 17 日

添加了有关使用 Amazon SageMaker Ground Truth 作业的主题	您无需自己标注图像，而是可以使用 Amazon SageMaker Ground Truth 作业，对用于图像分类模型和图像分割模型的图像进行标注。有关更多信息，请参阅使用 Amazon SageMaker Ground Truth 作业。	2022 年 8 月 19 日
Amazon Lookout for Vision 现在提供了异常定位功能。	您可以创建分割模型，用于发现图像上存在的不同异常类型（如划痕、凹痕或撕裂）的位置，查明异常标签和异常大小。有关更多信息，请参阅运行经过训练的 Amazon Lookout for Vision 模型。	2022 年 8 月 16 日
Amazon Lookout for Vision 现在可在边缘设备上提供 CPU 推理功能。	现在，可以部署 Amazon Lookout for Vision 模型，使之在运行 Linux 且只有 CPU 的 x86 计算平台上执行本地推理任务，无需 GPU 加速器。有关更多信息，请参阅 CPU 加速器。	2022 年 8 月 16 日
Amazon Lookout for Vision 现在可以自动扩缩推理单位。	为了帮助应对需求高峰，Amazon Lookout for Vision 现在可以自动扩缩供模型使用的推理单位数量。有关更多信息，请参阅运行经过训练的 Amazon Lookout for Vision 模型。	2022 年 8 月 16 日
添加了 Java 示例	Amazon Lookout for Vision 开发者指南中现在包含了 Java 示例。有关更多信息，请参阅开始使用 AWS SDK。	2022 年 5 月 2 日

边缘设备模型部署通用版	现在已全面支持将模型部署到由 AWS IoT Greengrass Version 2 管理的边缘设备。有关更多信息，请参阅在边缘设备上使用您的 Amazon Lookout for Vision 模型。	2022 年 3 月 14 日
更新了控制台桶信息	更新了有关控制台桶内容的信息，以及与创建控制台桶的替代方法相关的信息。有关更多信息，请参阅步骤 4：创建控制台桶。	2022 年 3 月 7 日
通过 CSV 文件创建清单文件	现在，您可以使用脚本读取 CSV 文件中的分类信息，从而简化清单文件的创建。有关更多信息，请参阅通过 CSV 文件创建清单文件。	2022 年 2 月 10 日
边缘设备模型部署的预览版	现在已发布预览版，支持将模型部署到由 AWS IoT Greengrass Version 2 管理的边缘设备。有关更多信息，请参阅在边缘设备上使用您的 Amazon Lookout for Vision 模型。	2021 年 12 月 7 日
添加了新的 Python 和 Java 2 示例	添加了 Python 和 Java 2 示例，用于演示如何使用 `DetectAnomalies` 来分析图像。有关更多信息，请参阅检测图像中的异常。	2021 年 9 月 7 日

添加了新的 AWS 托管式策略。	Amazon Lookout for Vision 增加了对 AWS 托管式策略的支持。有关更多信息，请参阅适用于 Amazon Lookout for Vision 的 AWS 托管式策略。	2021 年 5 月 11 日
更新了推理单位信息。	添加了描述推理单位及其收费方式的信息。有关更多信息，请参阅运行经过训练的 Amazon Lookout for Vision 模型。	2021 年 3 月 15 日
Amazon Lookout for Vision 通用版。	Amazon Lookout for Vision 现已正式发布。更新了 Python 代码示例，以便处理训练模型等异步任务。	2021 年 2 月 17 日
添加了标记和 AWS CloudFormation 支持。	现在，您可以标记 Amazon Lookout for Vision 模型并使用 AWS CloudFormation 创建项目。有关更多信息，请参阅标记模型和使用 Amazon CloudFormation 创建 Amazon Lookout for Vision 项目。	2021 年 1 月 31 日
新功能和指南	这是 Amazon Lookout for Vision 服务 Amazon Lookout for Vision 开发者指南 的初始版本。	2020 年 12 月 1 日

AWS 术语表

有关最新的 AWS 术语，请参阅《AWS 词汇表参考》中的 AWS 词汇表。

本文属于机器翻译版本。若本译文内容与英语原文存在差异，则一律以英文原文为准。