aws

开发人员指南

Amazon Comprehend

Amazon Comprehend: 开发人员指南

Table of Contents

Amazon Comprehend 是什么？

Amazon Comprehend 使用自然语言处理 (NLP) 来提取有关文档内容的见解。它可以通过识别文档中的实体、关键短语、语言、情绪和其他常见元素生成见解。使用 Amazon Comprehend 在了解文档结构的基础上创建新产品。例如，使用 Amazon Comprehend，您可以在社交网络提要中搜索提及产品的内容，或者在整个文档存储库中扫描关键短语。

您可以使用 Amazon Comprehend 控制台或 Amazon Comprehend API 访问 Amazon Comprehend 文档分析功能。您可以对小型工作负载运行实时分析，也可以为大型文档集启动异步分析作业。您可以使用 Amazon Comprehend 提供的预训练模型，也可以训练自己的自定义模型进行分类和实体识别。

Amazon Comprehend 可能会存储您的内容以持续改进其分析模型的质量。要了解更多信息，请参阅 Amazon Comprehend 常见问题解答。

所有 Amazon Comprehend 特征都接受 UTF-8 文本文档作为输入。此外，自定义分类和自定义实体识别还接受图像文件、PDF 文件和 Word 文件作为输入。

Amazon Comprehend 可以根据具体特征检查和分析各种语言的文档。有关更多信息，请参阅Amazon Comprehend 支持的语言。Amazon Comprehend 的 主要语言 功能可以检查文档并为更广泛的语言选择确定主要语言。

主题

Amazon Comprehend 见解

Amazon Comprehend 使用预先训练的模型来检查和分析一个或一组文档，以收集有关它的见解。该模型在大量文本上持续训练，因此您无需提供训练数据。

Amazon Comprehend 分析了以下类型的见解：

- 实体：对文档中包含的人物、地点、项目和位置的引用。
- 关键短语：出现在文档中的短语。例如，关于篮球比赛的文档可能会返回球队的名称、场地名称和最终比分。
- 个人身份信息 (PII)：可以识别个人身份的个人数据，例如地址、银行账号或电话号码。
- 语言：文档的主要语言。
- 情绪：文档的主导情绪，可以是积极的、中性的、消极的，也可以是混合的。
- 目标情绪：与文档中特定实体相关的情绪。每个实体的情绪可以是积极、中性、消极或混合的。
- 语法：文档中每个单词的词性。

有关更多信息，请参阅洞察。

Amazon Comprehend 自定义

您可以根据自己的特定要求自定义 Amazon Comprehend，而无需具备构建基于机器学习的 NLP 解决方案所需的技能。使用自动机器学习或 AutoML，Amazon Comprehend 自定义使用您已有的数据为您构建自定义 NLP 模型。

自定义分类：创建自定义分类模型（分类器），将您的文档整理到您自己的类别中。

自定义实体识别：创建自定义实体识别模型（识别器），该模型可以根据您的特定术语和名词短语分析文本。

有关更多信息，请参阅Amazon Comprehend 自定义。

飞轮

随着时间的推移，使用飞轮可以简化训练和管理自定义模型版本的过程。飞轮有助于编排与训练和评估新的模型版本相关的任务。飞轮支持纯文本自定义模型，用于自定义分类和自定义实体识别。有关更多信息，请参阅飞轮。

文档集群（主题建模）

您也可以使用 Amazon Comprehend 检查文档语料库，以便根据文档中的相似关键字对它们进行整理。文档集群（主题建模）可用于将大型文档语料库组织成基于词频相似的主题或集群。有关更多信息，请参阅主题建模。

示例

以下示例展示了如何在应用程序中使用 Amazon Comprehend 操作。

Example 1：查找有关某个主题的文档

使用 Amazon Comprehend 主题建模查找有关特定主题的文档。扫描一组文档以确定所讨论的主题，并找到与每个主题相关的文档。您可以指定 Amazon Comprehend 应从文档集中返回的主题数量。

Example 2：了解客户对产品的感受

如果您的公司发布了目录，请让 Amazon Comprehend 告诉您顾客对您的产品的看法。将每位客户评论发送给 `DetectSentiment` 运营部门，它将告诉您客户对产品的评价是积极、消极、中性还是混合的。

Example 3：发现客户最关心的问题

使用 Amazon Comprehend 主题建模来发现客户在论坛和留言板上谈论的主题，然后使用实体检测来确定他们与该主题关联的人物、地点和事物。使用情绪分析来确定客户对某个话题的看法。

优势

使用 Amazon Comprehend 的优点包括：

- 将强大的自然语言处理功能集成到您的应用程序中：Amazon Comprehend 通过简单的 API 提供强大且准确的自然语言处理功能，消除了在应用程序中构建文本分析功能的复杂性。您不需要文本分析专业知识就能充分利用 Amazon Comprehend 生成的见解。
- 基于深度学习的自然语言处理：Amazon Comprehend 使用深度学习技术来准确分析文本。我们的模型不断使用跨多个领域的新数据进行训练，以提高准确性。
- 可扩展的自然语言处理：Amazon Comprehend 使您能够分析数百万份文档，从而发现其中包含的见解。
- 与其他 AWS 服务集成 — Amazon Comprehend 旨在与其他 AWS 服务无缝协作，例如亚马逊 S3、和。 AWS KMS AWS Lambda将您的文档存储在亚马逊 S3 中，或者使用 Firehose 分析实时数据。Support for AWS Identity and Access Management (IAM) 可以轻松安全地控制对 Amazon Comprehend 运营的访问权限。借助 IAM，您可以创建和管理用户和组，以便向开发人员和最终用户授予适当的访问权限。

- 输出结果和卷数据的加密：Amazon S3 已经允许您加密输入文档，而 Amazon Comprehend 将这一功能进一步扩展。通过使用您自己的 KMS 密钥，您可以加密作业的输出结果以及附加到处理分析作业的计算实例的存储卷上的数据。结果是显著增强了安全性。
- 低成本：Amazon Comprehend 不收取最低费用或预付款。您需要为分析的文档和训练的自定义模型付费。

Amazon Comprehend 定价

使用 Amazon Comprehend 时，您仅需为实际使用的资源付费。如果您是 AWS 新客户，还可以免费试用 Amazon Comprehend。有关更多信息，请参阅 AWS 免费使用套餐。

运行实时或异步分析作业需要支付费用。您需要为训练自定义模型付费，并为自定义模型管理付费。对于使用自定义模型的实时请求，从启动终端节点开始计费，直到删除终端节点为止。使用飞轮不收取任何额外费用。但是，当您运行飞轮迭代时，训练新模型版本和存储模型数据会产生标准费用。

有关费率和其他详细信息，请参阅 Amazon Comprehend 定价。

您是否是首次接触 Amazon Comprehend 的用户？

如果您是首次接触 Amazon Comprehend 的用户，我们建议您按顺序阅读以下章节：

1. 工作方式：本节介绍 Amazon Comprehend 的概念。
2. 设置：在本节中，您将创建一个帐户并设置 AWS CLI。
3. Amazon Comprehend 入门：在本节中，您将运行 Amazon Comprehend 分析作业。
4. 教程：使用 Amazon Comprehend 分析来自客户评论的见解：在本节中，您将进行情绪和实体分析并将结果可视化。
5. Amazon Comprehend API 参考：Amazon Comprehend 操作的参考文档。

AWS 提供了以下资源供您了解 Amazon Comprehend 服务：

- AWS 机器学习博客包含有关 Amazon Comprehend 的有用文章。
- Amazon Comprehend 资源提供有关 Amazon Comprehend 的实用视频和教程。

工作方式

Amazon Comprehend 使用预先训练的模型来收集有关一个或一组文档的见解。该模型在大量文本上持续训练，因此您无需提供训练数据。

您可以使用 Amazon Comprehend 构建自己的自定义模型，用于自定义分类和自定义实体识别。您可以使用 飞轮 来帮助管理自定义模型。

Amazon Comprehend 使用内置模型提供主题建模。主题建模检查文档语料库，并根据文档中的相似关键字组织文档。

Amazon Comprehend 提供同步和异步文档处理模式。使用同步模式处理一个文档或一批最多 25 个文档。使用异步作业来处理大量文档。

Amazon Comprehend 与 AWS Key Management Service (AWS KMS) 配合使用，为您的数据提供增强的加密。有关更多信息，请参阅 Amazon Comprehend 中的 KMS 加密。

重要概念

- 洞察
- Amazon Comprehend 自定义
- 主题建模
- 文档处理模式

洞察

Amazon Comprehend 可以分析一个或一组文档，以收集有关它的见解。Amazon Comprehend 对文档的一些见解包括：

- 实体：Amazon Comprehend 会返回文档中标识的实体（例如人员、地点和地点）的列表。
- 事件：Amazon Comprehend 可检测特定类型的事件和相关详细信息。
- 关键短语：Amazon Comprehend 提取文档中出现的关键短语。例如，关于篮球比赛的文档可能会返回球队的名称、场地名称和最终比分。
- 个人身份信息（PII）：Amazon Comprehend 分析文档以检测可识别个人身份的个人数据，例如地址、银行账号或电话号码。
- 主要语言：Amazon Comprehend 可识别文档中的主要语言。Amazon Comprehend 可识别 100 种语言。

- 情绪：Amazon Comprehend 可确定文档的主导情绪。情绪可以被评为积极、中性、消极或混合。
- 目标情绪：Amazon Comprehend 可确定文档中提及的特定实体的情绪。每次提及的情绪可以是积极、中性、消极或混合。
- 语法分析：Amazon Comprehend 可解析文档中的每个单词并确定该单词的词性。例如，在“西雅图它今天在下雨”这句话中，“它”被标识为代词，“下雨”被标识为动词，“西雅图”被标识为专有名词。

实体

实体是对现实世界对象（例如人物、地点和商业物品）的唯一名称的文本引用，也是对日期和数量等度量的精确引用。

例如，在“约翰在 2012 年搬到知更鸟巷 1313 号”的文本中，“约翰”可能被识别为 PERSON，“知更鸟巷 1313 号”可能被识别为 LOCATION，“2012”可能被识别为 DATE。

每个实体还有一个分数，用于表示 Amazon Comprehend 对正确检测到实体类型的置信度。您可以筛选出分数较低的实体，以降低使用错误检测的风险。

下表列出了实体类型。

Type	描述
COMMERCIAL_ITEM	品牌产品
DATE	完整的日期（例如，2017 年 11 月 25 日）、日（星期二）、月（5 月）或时间（上午 8:30）
EVENT	事件，例如节日、音乐会、选举等。
LOCATION	特定位置，例如国家、城市、湖泊、建筑物等。
组织	大型组织，例如政府、公司、宗教、运动队等。
OTHER	不属于任何其他实体类别的实体
个人	个人、群体、昵称、虚构人物
数量	量化的金额，例如货币、百分比、数字、字节等。
TITLE	任何创作或创作作品的正式名称，例如电影、书籍、歌曲等。

检测实体操作可以使用 Amazon Comprehend 支持的任何一种主要语言来执行。这只包括预定义（非自定义）实体检测。所有文件都必须使用同一种语言。

您可以使用以下任何 API 操作来检测文档或一组文档中的实体。

- DetectEntities
- BatchDetectEntities
- StartEntitiesDetectionJob

这些操作会返回一个 API 实体对象列表，文档中的每个实体对应一个。BatchDetectEntities 操作会返回一个 Entity 对象列表，批次中的每个文档对应一个列表。StartEntitiesDetectionJob 操作启动一个异步作业，该作业生成一个文件，其中包含作业中每个文档的 Entity 对象列表。

以下示例是 DetectEntities 操作的响应。

```
{
    "Entities": [
        {
            "Text": "today",
            "Score": 0.97,
            "Type": "DATE",
            "BeginOffset": 14,
            "EndOffset": 19
        },
        {
            "Text": "Seattle",
            "Score": 0.95,
            "Type": "LOCATION",
            "BeginOffset": 23,
            "EndOffset": 30
        }
    ],
    "LanguageCode": "en"
}
```

事件

使用事件检测来分析文本文档中特定类型的事件及其相关实体。Amazon Comprehend 支持使用异步分析作业对大型文档集合进行事件检测。有关事件的更多信息，包括事件分析作业示例，请参阅宣布推出 Amazon Comprehend 事件

实体

Amazon Comprehend 从输入文本中提取与检测到的事件相关的实体列表。实体可以是现实世界中的对象，例如人物、地点或位置；实体也可以是一个概念，例如测量、日期或数量。实体的每次出现都通过提及来标识，提及是在输入文本中对该实体的文本引用。对于每个唯一实体，所有提及都被分组到一个列表中。此列表提供了输入文本中实体出现的每个位置的详细信息。Amazon Comprehend 仅检测与支持的事件类型相关的实体。

与支持的事件类型关联的每个实体都会返回以下相关详细信息：

- 提及：输入文本中每次出现相同实体的详细信息。
  - BeginOffset: 输入文本中的字符偏移量，用于显示提及的开始位置（第一个字符位于位置 0）。
  - EndOffset: 输入文本中的字符偏移量，用于显示提及的结束位置。
  - 分数：Amazon Comprehend 对实体类型准确性的置信度。
  - GroupScore：Amazon Comprehend 对提及的内容与对同一实体的其他提及正确分组的信心程度。
  - 文本：实体的文本。
  - 类型：实体的类型。有关支持的所有实体类型，请参阅 实体类型。

事件

Amazon Comprehend 会返回其在输入文本中检测到的事件（支持的事件类型）列表。每个事件都会返回以下相关详细信息：

- 类型：事件的类型。有关支持的所有事件类型，请参阅 事件类型。
- 参数：与检测到的事件相关的参数列表。参数由与检测到的事件相关的实体组成。参数的角色描述了这种关系，例如谁在何时何地做了什么。
  - EntityIndex：一个索引值，用于从 Amazon Comprehend 为本次分析返回的实体列表中标识实体。
  - 角色：参数类型，用于描述该参数的实体与事件的关系。有关支持的所有参数类型，请参阅 参数类型。
  - 分数：Amazon Comprehend 对角色检测准确性的置信度。
- 触发器：检测到的事件的触发器列表。触发器是指示事件发生的单词或短语。
  - BeginOffset: 输入文本中的字符偏移量，显示触发器的起始位置（第一个字符位于位置 0）。
  - EndOffset: 输入文本中的字符偏移量，用于显示触发器的终止位置。

- 分数：Amazon Comprehend 对检测准确性的置信度。
- 文本：触发器的文本。
- GroupScore：Amazon Comprehend 对同一事件的触发器与其他触发器正确分组的信心程度。
- 类型：该触发器指示的事件类型。

检测事件结果格式

事件检测作业完成后，Amazon Comprehend 会将分析结果写入启动作业时指定的 Amazon S3 输出位置。

对于每个检测到的事件，输出按以下格式提供详细信息：

```
{
   "Entities": [
     {
       "Mentions": [
         {
           "BeginOffset": number,
           "EndOffset": number,
           "Score": number,
           "GroupScore": number,
           "Text": "string",
           "Type": "string"
         }, ...
       ]
     }, ...
   ],
   "Events": [
     {
       "Type": "string",
       "Arguments": [
         {
           "EntityIndex": number,
           "Role": "string",
           "Score": number
         }, ...
       ],
       "Triggers": [
         {
           "BeginOffset": number,
           "EndOffset": number,
```

```
          "Score": number,
          "Text": "string",
          "GroupScore": number,
          "Type": "string"
        }, ...
      ]
    }, ...
  ]
}
```

实体、事件和参数支持的类型

实体类型

Type	描述
DATE	对日期或时间的任何提及，无论是具体的还是笼统的。
设施	建筑物、机场、高速公路、桥梁和其他永久性人造结构和房地产改造。
LOCATION	物理位置，例如街道、城市、州、国家、水域或地理坐标。
货币价值	以美元或其他货币表示的价值。该值可以是具体的，也可以是近似值。
组织	由既定组织结构定义的公司和其他群体。
个人	个人或虚构人物的名字或昵称。
人物头衔	描述一个人的任何头衔，通常是雇佣类别（例如首席执行官）或尊称类别（例如先生）。
数量	数字或数值以及计量单位。
股票代码	股票代码，例如AMZN、国际证券识别码 (ISIN)、统一证券识别程序委员会 (CUSIP) 或证券交易所每日官方名单 (SEDOL)。

事件类型

Type	描述
破产	涉及无法偿还未偿债务的个人或公司的法律程序。
雇用	当雇员被雇用、解雇、退休或以其他方式改变就业状态时发生。
企业收购	当一家公司获得另一家公司的大部分或全部股份或实物资产的所有权，从而获得对该公司的控制权时。
一般投资	当个人或公司购买有望产生未来收入或升值的资产时发生。
公司合并	当两家或多家公司联合创建新的法人实体时发生。
首次公开募股	首次公开募股 (IPO)，以发行新股的方式向公众发售私营公司股票。
供股	向现有股东提供的按其现有持股比例购买额外股票的权利，称为认购权证。
存量发行	公司股东发行的证券。
储架发行	美国证券交易委员会（SEC）的一项条款，允许发行人注册新发行的证券并在一段时间内出售部分发行的证券，而无需重新注册证券或受到处罚。也称为上架登记。
公开收购	购买公司部分或全部股东股份的要约。
股票分割	当公司董事会通过向现有股东发行更多股票来增加已发行股票数量时。此事件也适用于反向股票分割。

参数类型

破产的参数类型

参数类型	描述
申报者	申请破产的个人或公司。
DATE	破产的日期或时间。
发生地	破产发生地（或最近）的地点或设施。

雇佣的参数类型

Type	描述
雇员	公司雇用的人员。
雇员头衔	雇员的头衔。
雇主	雇用人员的个人或公司。
START_DATE	雇佣的开始日期或时间。
结束日期	雇佣的结束日期或时间。

企业收购、一般投资的参数类型

Type	描述
金额	与交易关联的货币价值。
被投资者	与投资相关的个人或公司。
投资者	投资于资产的个人或公司。
DATE	收购或投资的日期或时间。
发生地	收购或投资发生的地点（或最接近的地点）。

公司合并的参数类型

Type	描述
DATE	合并的日期或时间。
新公司	合并产生的新法人实体。
参与者	参与合并的公司。

首次公开募股、供股、存量发行、储架发行、公开收购的参数类型

Type	描述
到期日期	发行的到期日期或时间。
投资者	投资于资产的个人或公司。
被要约人	接受发行的个人或公司。
发行金额	与发行关联的货币价值。
发行日期	发行的日期或时间。
要约人	发起发行的个人或公司。
发行总价值	与发行关联的货币总价值。
记录日期	发行的记录日期或时间。
销售代理	为出售该产品提供便利的个人或公司。
股票价格	与股票关联的货币价值。
股票数量	与本次发行相关的股票数量。
承销商	与本次发行承销相关的公司。

股票分割的参数类型

Type	描述
公司	发行股票分割的公司。
DATE	股票分割的日期或时间。
分割比率	股票分割前新增的已发行股票数量与当前已发行股票数量的比率。

关键短语

关键短语是一个包含描述特定事物的名词短语的字符串。它通常由名词和区分它的修饰语组成。例如，“天” 是一个名词；“美好的一天”是一个名词短语，包括数词（“一”）和一个形容词（“美好的”）。每个关键短语都包含一个分数，该分数表明 Amazon Comprehend 对该字符串是名词短语的置信度。您可以使用该分数来确定检测是否对您的应用程序具有足够高的置信度。

检测关键短语操作可以使用 Amazon Comprehend 支持的任何一种主要语言来执行。所有文件都必须使用同一种语言。

您可以使用以下任何操作来检测文档或一组文档中的关键短语。

- DetectKeyPhrases
- BatchDetectKeyPhrases
- StartKeyPhrasesDetectionJob

这些操作会返回一个KeyPhrase对象列表，文档中每个关键短语对应一个对象。BatchDetectKeyPhrases 操作会返回一个 KeyPhrase 对象列表，批次中的每个文档对应一个对象。StartKeyPhrasesDetectionJob 操作启动一个异步作业，该作业生成一个文件，其中包含作业中每个文档的 KeyPhrase 对象列表。

以下示例是 DetectKeyPhrases 操作的响应。

```
{
    "LanguageCode": "en",
    "KeyPhrases": [
        {
            "Text": "today",
```

```
            "Score": 0.89,
            "BeginOffset": 14,
            "EndOffset": 19
        },
        {
            "Text": "Seattle",
            "Score": 0.91,
            "BeginOffset": 23,
            "EndOffset": 30
        }
    ]
}
```

主要语言

您可以使用 Amazon Comprehend 检查文本以确定主要语言。Amazon Comprehend 使用 RFC 5646 中的标识符识别语言，如果有 2 个字母的 ISO 639-1 标识符，必要时带有区域子标签，则它会使用该标识符。否则，它将使用 ISO 639-2 3 个字母的代码。

有关 RFC 5646 的更多信息，请参阅 IETF 工具网站上用于识别语言的标签。

回复中包含一个分数，该分数表明 Amazon Comprehend 对某一特定语言是文档中主要语言的置信度。每个分数都独立于其他分数。分数并不表示某种语言占文档的特定百分比。

如果长文档（例如一本书）包含多种语言，则可以将长文档分成较小的部分，然后对各个部分进行 `DetectDominantLanguage` 操作。然后，您可以汇总结果以确定较长文档中每种语言的百分比。

Amazon Comprehend 语言检测具有以下限制：

- 它不支持语音语言检测。例如，它不会检测“arigato”为日语或“nihao”为中文。
- 它可能很难区分近似的语言对，例如印尼语和马来语；或者波斯尼亚语、克罗地亚语和塞尔维亚语。
- 为获得最佳结果，请提供至少 20 个字符的输入文本。

Amazon Comprehend 检测到以下语言。

代码	Language
af	南非荷兰语
am	阿姆哈拉语

代码	Language
ar	阿拉伯语
as	阿萨姆语
az	阿塞拜疆语
ba	巴什基尔语
be	白俄罗斯语
bn	孟加拉语
bs	波斯尼亚语
bg	保加利亚语
ca	加泰罗尼亚语
ceb	宿雾语
cs	捷克语
cv	楚瓦什语
cy	威尔士语
da	丹麦语
de	德语
el	希腊语
en	English
eo	世界语
et	爱沙尼亚语
eu	巴斯克语

代码	Language
fa	波斯语
fi	芬兰语
fr	French
gd	苏格兰盖尔语
ga	爱尔兰语
gl	加利西亚语
gu	古吉拉特语
ht	海地语
he	希伯来语
ha	豪萨语
hi	印地语
hr	克罗地亚语
hu	匈牙利语
hy	亚美尼亚语
ilo	伊洛卡诺语
id	印度尼西亚语
is	冰岛语
it	意大利语
jv	爪哇语
ja	日语

代码	Language
kn	卡纳达语
ka	格鲁吉亚语
kk	哈萨克语
km	中部高棉语
ky	吉尔吉斯语
ko	韩语
ku	库尔德语
lo	老挝语
la	拉丁语
lv	拉脱维亚语
lt	立陶宛语
lb	卢森堡语
ml	马拉雅拉姆语
mt	马耳他语
mr	马拉地语
mk	马其顿语
mg	马达加斯加语
mn	蒙古语
ms	马来语
my	缅甸语

代码	Language
ne	尼泊尔语
new	尼瓦尔语
nl	荷兰语
no	挪威语
or	奥里亚语
om	奥罗莫语
pa	旁遮普语
pl	波兰语
pt	葡萄牙语
ps	普什图语
qu	盖丘亚语
ro	罗马尼亚语
ru	俄语
sa	梵语
si	僧伽罗语
sk	斯洛伐克语
sl	斯洛文尼亚语
sd	信德语
so	索马里语
es	西班牙语

代码	Language
sq	阿尔巴尼亚语
sr	塞尔维亚语
su	巽他语
sw	斯瓦西里语
sv	瑞典语
ta	泰米尔语
tt	鞑靼语
te	泰卢固语
tg	塔吉克语
tl	塔加洛语
th	泰语
tk	土库曼语
tr	土耳其语
ug	维吾尔族语
uk	乌克兰语
ur	乌尔都语
uz	乌兹别克斯坦语
vi	越南语
yi	意第绪语
yo	约鲁巴语

代码	Language
zh	中文（简体）
zh-TW	中文（繁体）

您可以使用以下任意操作来检测一个文档或一组文档中的主要语言。

- DetectDominantLanguage
- BatchDetectDominantLanguage
- StartDominantLanguageDetectionJob

该DetectDominantLanguage操作返回一个DominantLanguage对象。BatchDetectDominantLanguage 操作会返回一个 DominantLanguage 对象列表，批次中的每个文档对应一个对象。StartDominantLanguageDetectionJob 操作启动一个异步作业，该作业生成一个包含 DominantLanguage 对象列表的文件，每个对象对应作业中的每个文档。

以下示例是 DetectDominantLanguage 操作的响应。

```
{
    "Languages": [
        {
            "LanguageCode": "en",
            "Score": 0.9793661236763
        }
    ]
}
```

情绪

使用 Amazon Comprehend 来确定 UTF-8 编码的文本文档中内容的情绪。例如，您可以使用情绪分析来确定博客帖子中评论的情绪，以确定您的读者是否喜欢该帖子。

您可以确定对 Amazon Comprehend 支持的任何主要语言的文档的情绪。一项作业中的所有文件必须使用同一种语言。

情绪确定返回以下值：

- 积极：文本表达了总体上的积极情绪。

- 消极：文本表达了总体上的消极情绪。
- 混合：文本既表达了积极情绪，也表达了消极情绪。
- 中性：文本不表达积极或消极的情绪。

您可以使用以下任何 API 操作来检测文档或一组文档中的情绪。

- DetectSentiment
- BatchDetectSentiment
- StartSentimentDetectionJob

这些操作返回文本中最可能的情绪以及每种情绪的分数。分数表示正确检测到情绪的可能性。例如，在下面的示例中，文本有 Positive 情绪的可能性为95%。文本中带有 Negative 情绪的可能性不到1%。您可以使用 SentimentScore 来确定检测的准确性是否满足您的应用程序的需求。

该DetectSentiment操作返回一个包含检测到的情绪的对象和一个SentimentScore对象。BatchDetectSentiment 操作会返回一个情绪和 SentimentScore 对象列表，批次中的每个文档对应一个。StartSentimentDetectionJob 操作启动一个异步作业，该作业生成一个包含情绪和 SentimentScore 对象列表的文件，作业中的每个文档对应一个。

以下示例是 DetectSentiment 操作的响应。

```
{
"SentimentScore": {
        "Mixed": 0.030585512690246105,
        "Positive": 0.94992071056365967,
        "Neutral": 0.0141543131828308,
        "Negative": 0.00893945890665054
    },
    "Sentiment": "POSITIVE",
    "LanguageCode": "en"
}
```

目标情绪

目标情绪可让您详细了解与输入文档中特定实体（例如品牌或产品）关联的情绪。

目标情绪和情绪之间的区别在于输出数据的粒度级别。情绪分析确定每个输入文档的主导情绪，但不提供用于进一步分析的数据。目标情绪分析确定每个输入文档中特定实体的实体级情绪。您可以分析输出数据以确定获得正面或负面反馈的特定产品和服务。

例如，在一组餐厅评论中，一位顾客提供了以下评论：“炸玉米饼很好吃，工作人员也很友善。”对该评论的分析得出以下结果：

- 情绪分析将确定每条餐厅评论中的总体情绪是否为积极、消极、中性或混合的。在此示例中，总体情绪是积极的。
- 目标情绪分析可以确定顾客在评论中提到的餐厅实体和属性的情绪。在此示例中，客户对“炸玉米饼”和“员工”给予了积极评价。

目标情绪为每项分析作业提供以下输出：

- 文档中提及的实体的身份。
- 每个提及的实体的实体类型分类。
- 提及的每个实体的情绪和情绪分数。
- 对应于单个实体的提及组（共同引用组）。

您可以使用控制台或 API 来运行目标情绪分析。控制台和 API 支持针对目标情绪的实时分析和异步分析。

Amazon Comprehend 支持对英语文档进行目标情绪。

有关目标情绪的其他信息（包括教程），请参阅 AWS 机器学习博客中的使用 Amazon Comprehend 目标情绪在文本中提取精细情绪。

主题

实体类型

目标情绪可识别以下实体类型。如果实体不属于任何其他类别，则它会分配实体类型 OTHER。输出文件中提及的每个实体都包括实体类型，例如 `"Type": "PERSON"`。

实体类型定义

实体类型	定义
个人	例子包括个人、群体、昵称、虚构人物和动物名称。
LOCATION	地理位置，例如国家、城市、州、地址、地质构造、水体、自然地标和天文位置。
组织	例子包括政府、公司、运动队和宗教。
设施	建筑物、机场、高速公路、桥梁和其他永久性人造结构和房地产改造。
BRAND	特定商业项目或产品系列的组织、团体或生产商。
COMMERCIAL_ITEM	任何非通用可购买或可获得的物品，包括车辆和仅生产一件物品的大型产品。
MOVIE	电影或电视节目。实体可以是全名、昵称或副标题。
MUSIC	一首歌曲，全部或部分歌曲。此外，还有个人音乐创作的集合，例如专辑或选集。
BOOK	一本书，专业出版或自行出版。
SOFTWARE	正式发布的软件产品。
GAME	一种游戏，例如电子游戏、棋盘游戏、普通游戏或体育运动。
PERSONAL_TITLE	官方头衔和荣誉，例如校长、博士学位或博士.
EVENT	例子包括节日、音乐会、选举、战争、会议和宣传活动。
DATE	对日期或时间的任何引用，无论是具体的还是一般的，无论是绝对的还是相对的。
数量	所有测量值及其单位（货币、百分比、数字、字节等）。
ATTRIBUTE	实体的一种属性、特征或特性，例如产品的“质量”、手机的“价格”或 CPU 的“速度”。
OTHER	不属于任何其他类别的实体。

共同引用组

目标情绪标识每个输入文档中的共同引用组。共同引用组是文档中对应于一个现实世界实体的一组提及。

Example

在以下客户评论示例中，“spa”是实体，其实体类型是 FACILITY。该实体还有两次作为代词（“it”）被提及。

输出文件组织

目标情绪分析作业会创建一个 JSON 文本输出文件。该文件包含每个输入文档的一个 JSON 对象。每个 JSON 对象包含以下字段：

- 实体：在文档中找到的实体数组。
- 文件：输入文档的文件名。
- 行：如果输入文件每行一个文档，则实体包含文件中该文档的行号。

> **Note**
>
> 如果目标情绪在输入文本中未识别出任何实体，则它将返回一个空数组作为实体结果。

以下示例显示了包含三行输入的输入文件的实体。输入格式为每行一个文档，因此每行输入都是一个文档。

```
{ "Entities":[
```

```
    {entityA},
    {entityB},
    {entityC}
    ],
  "File": "TargetSentimentInputDocs.txt",
  "Line": 0
}
{ "Entities": [
    {entityD},
    {entityE}
  ],
  "File": "TargetSentimentInputDocs.txt",
  "Line": 1
}
{ "Entities": [
    {entityF},
    {entityG}
    ],
  "File": "TargetSentimentInputDocs.txt",
  "Line": 2
}
```

实体数组中的实体包括文档中检测到的实体提及的逻辑分组（称为共同引用组）。每个实体的总体结构如下：

```
{"DescriptiveMentionIndex": [0],
  "Mentions": [
     {mentionD},
     {mentionE}
    ]
}
```

实体包含以下字段：

- 提及：文档中提及该实体的数组。该数组代表一个共同引用组。有关示例，请参阅 the section called “共同引用组”。提及数组中提及的顺序是它们在文档中的位置（偏移量）顺序。每次提及都包括该提及的情绪分数和小组分数。小组分数表示这些提及属于同一实体的置信度。
- DescriptiveMentionIndex— “提及” 数组中的一个或多个索引，该索引为实体组提供最佳名称。例如，一个实体可能有三个提及，其文本值为“ABC 酒店”、“ABC 酒店”和“it”。最好的名字是 “ABC Hotel”，其 DescriptiveMentionIndex 值为 [0,1]。

每次提及都包含以下字段

- BeginOffset— 提及开始处的文档文本中的偏移量。
- EndOffset— 文档文本中提及结束处的偏移量。
- GroupScore— 确信该组中提到的所有实体都与同一个实体有关。
- 文本：文档中用于标识实体的文本。
- 类型1：实体的类型。Amazon Comprehend 支持多种实体类型。
- 分数：对实体相关性的置信度进行建模。值范围为 0 到 1，其中 1 表示置信度最高。
- MentionSentiment— 包含提及的情绪和情绪分数。
- 情绪：提及的情绪。值包括：积极、中性、消极和混合。
- SentimentScore— 为每种可能的情绪提供模型信心。值范围为 0 到 1，其中 1 表示置信度最高。

情绪值具有以下含义：

- 积极：提及的实体表达积极的情绪。
- 消极：提及的实体表示消极的情绪。
- 混合：提及的实体既表达了积极情绪，也表达了消极情绪。
- 中性：提及的实体不表达积极或消极的情绪。

在以下示例中，一个实体在输入文档中只有一个提及，因此 DescriptiveMentionIndex为零（提及数组中的第一个提及）。被识别的实体是一个名为“I”的人，情绪分数是中性的。

```
{"Entities":[
  {
    "DescriptiveMentionIndex": [0],
    "Mentions": [
      {
       "BeginOffset": 0,
       "EndOffset": 1,
       "Score": 0.999997,
       "GroupScore": 1,
       "Text": "I",
       "Type": "PERSON",
       "MentionSentiment": {
         "Sentiment": "NEUTRAL",
         "SentimentScore": {
           "Mixed": 0,
```

```
            "Negative": 0,
            "Neutral": 1,
            "Positive": 0
          }
        }
      }
    ]
   }
 ],
 "File": "Input.txt",
 "Line": 0
}
```

使用控制台进行实时分析

您可以使用 Amazon Comprehend 控制台实时运行 the section called “目标情绪”。使用示例文本或将您自己的文本粘贴到输入文本框中，然后选择分析。

在见解面板中，控制台显示目标情绪分析的三个视图：

- 分析的文本：显示分析的文本并给每个实体加下划线。下划线的颜色表示分析分配给实体的情绪值（积极、中性、消极或混合）。控制台将颜色映射显示在分析文本框的右上角。如果将光标悬停在实体上，控制台会显示一个弹出式面板，其中包含该实体的分析值（实体类型、情绪分数）。
- 结果：显示一个表格，其中包含文本中标识的每个实体提及的一行。对于每个实体，该表显示了实体和实体分数。该行还包括主要情绪和每个情绪值的分数。如果多次提及同一个实体（称为 the section called “共同引用组”），则表格会将这些提及显示为一组与主实体关联的可折叠行。

  如果将鼠标悬停在结果表中的实体行上，则控制台会在分析文本面板中突出显示提及的实体。
- 应用程序集成：显示 API 请求的参数值以及 API 响应中返回的 JSON 对象的结构。有关 JSON 对象中字段的描述，请参阅 the section called “输出文件组织”。

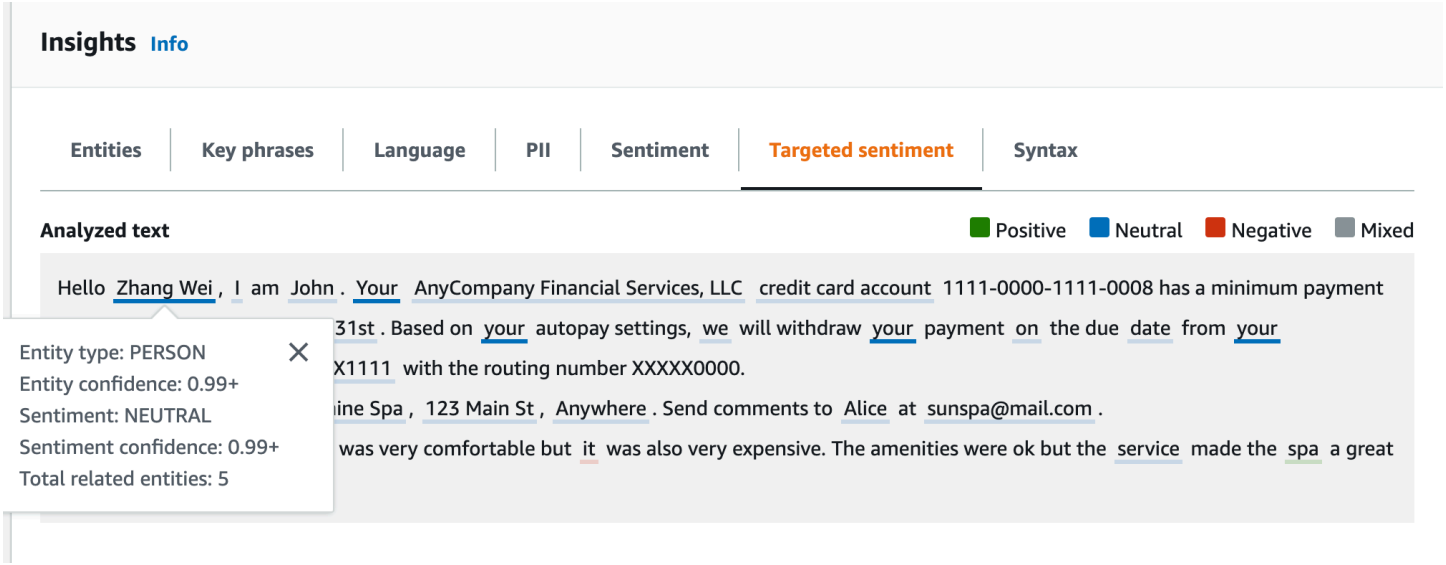
控制台实时分析示例

此示例使用以下文本作为输入，这是控制台提供的默认输入文本。

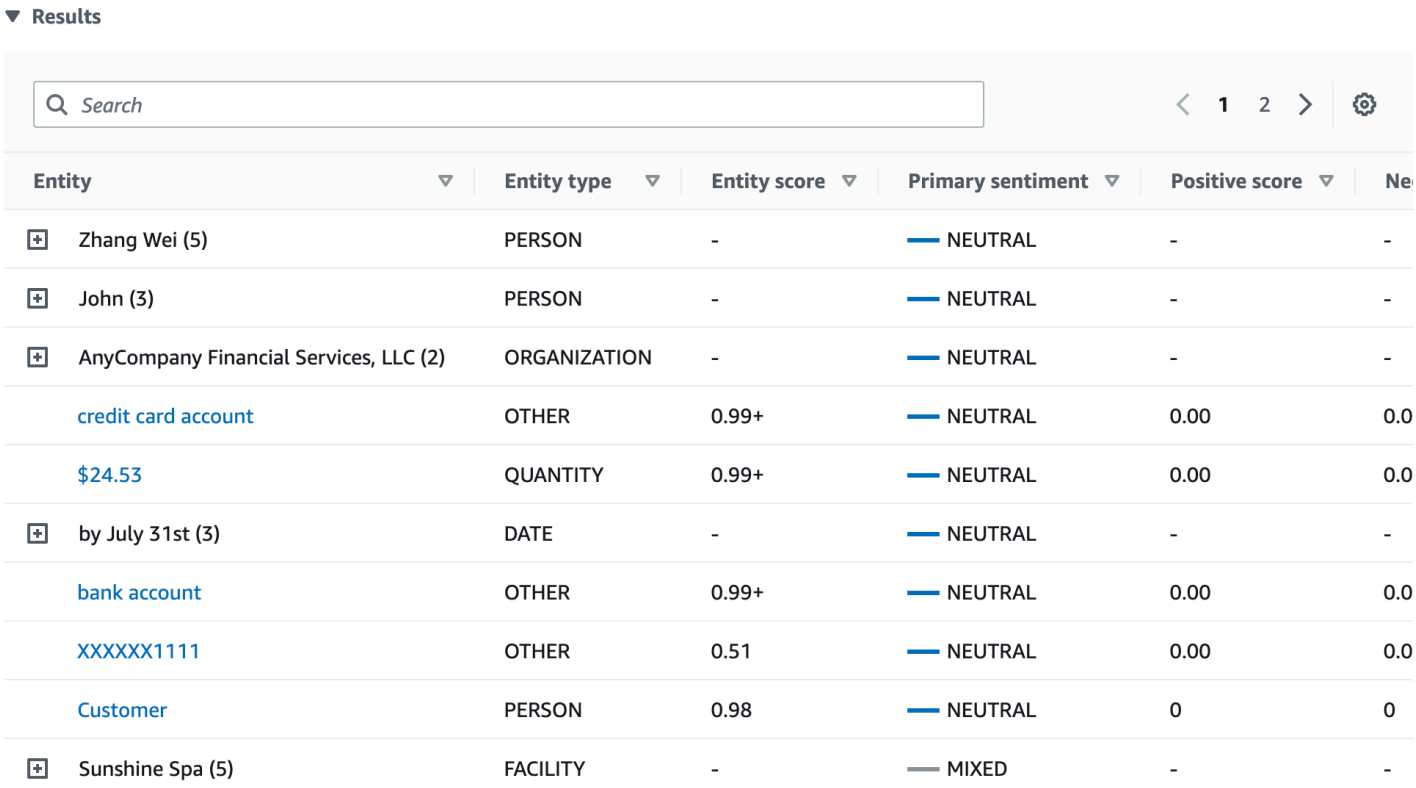
```
Hello Zhang Wei, I am John. Your AnyCompany Financial Services, LLC credit card account
 1111-0000-1111-0008 has a minimum payment
  of $24.53 that is due by July 31st. Based on your autopay settings, we will withdraw
 your payment on the due date from your
  bank account number XXXXXX1111 with the routing number XXXXX0000.
```

```
 Customer feedback for Sunshine Spa, 123 Main St, Anywhere. Send comments to Alice at
sunspa@mail.com.
 I enjoyed visiting the spa. It was very comfortable but it was also very expensive.
The amenities were ok but the service made
 the spa a great experience.
```

分析文本面板显示了此示例的以下输出。将鼠标悬停在文本 Zhang Wei 上方可查看该实体的弹出面板。



结果表提供了有关每个实体的更多详细信息，包括实体得分、主要情绪和每种情绪的分数。

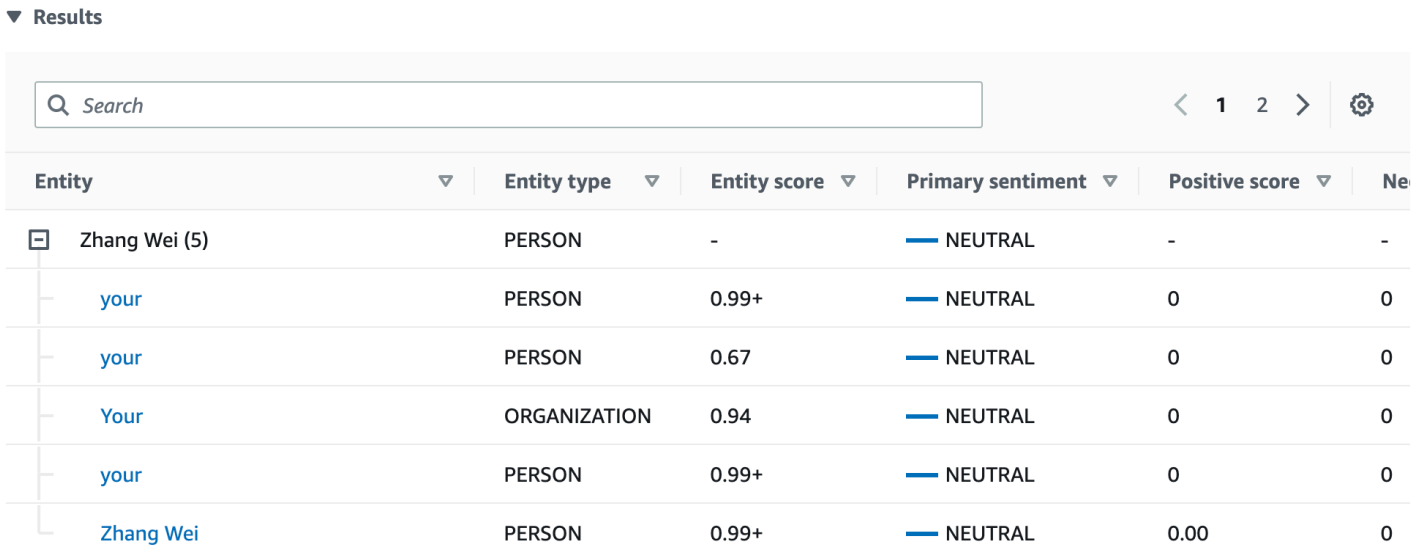


Results

Search 1 2

Entity	Entity type	Entity score	Primary sentiment	Positive score	Ne
Zhang Wei (5)	PERSON	-	NEUTRAL	-	-
John (3)	PERSON	-	NEUTRAL	-	-
AnyCompany Financial Services, LLC (2)	ORGANIZATION	-	NEUTRAL	-	-
credit card account	OTHER	0.99+	NEUTRAL	0.00	0.0
$24.53	QUANTITY	0.99+	NEUTRAL	0.00	0.0
by July 31st (3)	DATE	-	NEUTRAL	-	-
bank account	OTHER	0.99+	NEUTRAL	0.00	0.0
XXXXXX1111	OTHER	0.51	NEUTRAL	0.00	0.0
Customer	PERSON	0.98	NEUTRAL	0	0
Sunshine Spa (5)	FACILITY	-	MIXED	-	-

在本示例中，目标情绪分析可以识别出，在输入文本中每次提及你都是对人物实体张伟的引用。控制台将这些提及显示为一组与主实体关联的可折叠行。



Results

Search 1 2

Entity	Entity type	Entity score	Primary sentiment	Positive score	Ne
Zhang Wei (5)	PERSON	-	NEUTRAL	-	-
your	PERSON	0.99+	NEUTRAL	0	0
your	PERSON	0.67	NEUTRAL	0	0
Your	ORGANIZATION	0.94	NEUTRAL	0	0
your	PERSON	0.99+	NEUTRAL	0	0
Zhang Wei	PERSON	0.99+	NEUTRAL	0.00	0

应用程序集成面板显示 DetectTargetedSentiment API 生成的 JSON 对象。有关完整示例，请参阅下一节。

目标情绪输出示例

以下示例显示了目标情绪分析作业的输出文件。输入文件由三个简单的文档组成：

```
The burger was very flavorful and the burger bun was excellent. However, customer
 service was slow.
My burger was good, and it was warm. The burger had plenty of toppings.
The burger was cooked perfectly but it was cold. The service was OK.
```

对该输入文件进行目标情绪分析会产生以下输出。

```
 {"Entities":[
   {
     "DescriptiveMentionIndex": [
       0
     ],
     "Mentions": [
       {
         "BeginOffset": 4,
         "EndOffset": 10,
         "Score": 0.999991,
         "GroupScore": 1,
         "Text": "burger",
         "Type": "OTHER",
         "MentionSentiment": {
           "Sentiment": "POSITIVE",
           "SentimentScore": {
             "Mixed": 0,
             "Negative": 0,
             "Neutral": 0,
             "Positive": 1
           }
         }
       }
     ]
   },
   {
     "DescriptiveMentionIndex": [
       0
     ],
     "Mentions": [
       {
         "BeginOffset": 38,
```

```
          "EndOffset": 44,
          "Score": 1,
          "GroupScore": 1,
          "Text": "burger",
          "Type": "OTHER",
          "MentionSentiment": {
            "Sentiment": "NEUTRAL",
            "SentimentScore": {
              "Mixed": 0.000005,
              "Negative": 0.000005,
              "Neutral": 0.999591,
              "Positive": 0.000398
            }
          }
        }
      ]
    },
    {
      "DescriptiveMentionIndex": [
        0
      ],
      "Mentions": [
        {
          "BeginOffset": 45,
          "EndOffset": 48,
          "Score": 0.961575,
          "GroupScore": 1,
          "Text": "bun",
          "Type": "OTHER",
          "MentionSentiment": {
            "Sentiment": "POSITIVE",
            "SentimentScore": {
              "Mixed": 0.000327,
              "Negative": 0.000286,
              "Neutral": 0.050269,
              "Positive": 0.949118
            }
          }
        }
      ]
    },
    {
      "DescriptiveMentionIndex": [
        0
```

```
      ],
      "Mentions": [
        {
          "BeginOffset": 73,
          "EndOffset": 89,
          "Score": 0.999988,
          "GroupScore": 1,
          "Text": "customer service",
          "Type": "ATTRIBUTE",
          "MentionSentiment": {
            "Sentiment": "NEGATIVE",
            "SentimentScore": {
              "Mixed": 0.000001,
              "Negative": 0.999976,
              "Neutral": 0.000017,
              "Positive": 0.000006
            }
          }
        }
      ]
    }
  ],
  "File": "TargetSentimentInputDocs.txt",
  "Line": 0
}
{
  "Entities": [
    {
      "DescriptiveMentionIndex": [
        0
      ],
      "Mentions": [
        {
          "BeginOffset": 0,
          "EndOffset": 2,
          "Score": 0.99995,
          "GroupScore": 1,
          "Text": "My",
          "Type": "PERSON",
          "MentionSentiment": {
            "Sentiment": "NEUTRAL",
            "SentimentScore": {
              "Mixed": 0,
              "Negative": 0,
```

```
            "Neutral": 1,
            "Positive": 0
          }
        }
      }
    ]
  },
  {
    "DescriptiveMentionIndex": [
      0,
      2
    ],
    "Mentions": [
      {
        "BeginOffset": 3,
        "EndOffset": 9,
        "Score": 0.999999,
        "GroupScore": 1,
        "Text": "burger",
        "Type": "OTHER",
        "MentionSentiment": {
          "Sentiment": "POSITIVE",
          "SentimentScore": {
            "Mixed": 0.000002,
            "Negative": 0.000001,
            "Neutral": 0.000003,
            "Positive": 0.999994
          }
        }
      },
      {
        "BeginOffset": 24,
        "EndOffset": 26,
        "Score": 0.999756,
        "GroupScore": 0.999314,
        "Text": "it",
        "Type": "OTHER",
        "MentionSentiment": {
          "Sentiment": "POSITIVE",
          "SentimentScore": {
            "Mixed": 0,
            "Negative": 0.000003,
            "Neutral": 0.000006,
            "Positive": 0.999991
```

```
            }
          }
        },
        {
          "BeginOffset": 41,
          "EndOffset": 47,
          "Score": 1,
          "GroupScore": 0.531342,
          "Text": "burger",
          "Type": "OTHER",
          "MentionSentiment": {
            "Sentiment": "POSITIVE",
            "SentimentScore": {
              "Mixed": 0.000215,
              "Negative": 0.000094,
              "Neutral": 0.00008,
              "Positive": 0.999611
            }
          }
        }
      ]
    },
    {
      "DescriptiveMentionIndex": [
        0
      ],
      "Mentions": [
        {
          "BeginOffset": 52,
          "EndOffset": 58,
          "Score": 0.965462,
          "GroupScore": 1,
          "Text": "plenty",
          "Type": "QUANTITY",
          "MentionSentiment": {
            "Sentiment": "NEUTRAL",
            "SentimentScore": {
              "Mixed": 0,
              "Negative": 0,
              "Neutral": 1,
              "Positive": 0
            }
          }
        }
```

```
      ]
    },
    {
      "DescriptiveMentionIndex": [
        0
      ],
      "Mentions": [
        {
          "BeginOffset": 62,
          "EndOffset": 70,
          "Score": 0.998353,
          "GroupScore": 1,
          "Text": "toppings",
          "Type": "OTHER",
          "MentionSentiment": {
            "Sentiment": "NEUTRAL",
            "SentimentScore": {
              "Mixed": 0,
              "Negative": 0,
              "Neutral": 0.999964,
              "Positive": 0.000036
            }
          }
        }
      ]
    }
  ],
  "File": "TargetSentimentInputDocs.txt",
  "Line": 1
}
{
  "Entities": [
    {
      "DescriptiveMentionIndex": [
        0
      ],
      "Mentions": [
        {
          "BeginOffset": 4,
          "EndOffset": 10,
          "Score": 1,
          "GroupScore": 1,
          "Text": "burger",
          "Type": "OTHER",
```

```
          "MentionSentiment": {
            "Sentiment": "POSITIVE",
            "SentimentScore": {
              "Mixed": 0.001515,
              "Negative": 0.000822,
              "Neutral": 0.000243,
              "Positive": 0.99742
            }
          }
        },
        {
          "BeginOffset": 36,
          "EndOffset": 38,
          "Score": 0.999843,
          "GroupScore": 0.999661,
          "Text": "it",
          "Type": "OTHER",
          "MentionSentiment": {
            "Sentiment": "NEGATIVE",
            "SentimentScore": {
              "Mixed": 0,
              "Negative": 0.999996,
              "Neutral": 0.000004,
              "Positive": 0
            }
          }
        }
      ]
    },
    {
      "DescriptiveMentionIndex": [
        0
      ],
      "Mentions": [
        {
          "BeginOffset": 53,
          "EndOffset": 60,
          "Score": 1,
          "GroupScore": 1,
          "Text": "service",
          "Type": "ATTRIBUTE",
          "MentionSentiment": {
            "Sentiment": "NEUTRAL",
            "SentimentScore": {
```

```
              "Mixed": 0.000033,
              "Negative": 0.000089,
              "Neutral": 0.993325,
              "Positive": 0.006553
            }
          }
        }
      ]
    }
  ],
  "File": "TargetSentimentInputDocs.txt",
  "Line": 2
}
  }
```

语法分析

使用语法分析来解析文档中的单词，并返回文档中每个单词的词性部分或句法函数。您可以识别文档中的名词、动词、形容词等。使用这些信息可以更深入地了解文档的内容，并了解文档中单词的关系。

例如，您可以在文档中查找名词，然后查找与这些名词相关的动词。在像“我的祖母移动了沙发”这样的句子中，您可以看到名词“祖母”和“沙发”，以及动词“移动”。您可以使用这些信息来构建应用程序，用于分析您感兴趣的单词组合的文本。

为了开始分析，Amazon Comprehend 会解析源文本，找到文本中的各个单词。解析文本后，会为每个单词分配其在源文本中占用的词性部分。

Amazon Comprehend 可以识别以下词性。

令牌	词性
ADJ	形容词 通常修饰名词的单词。
ADP	介词 介词或后置短语的开头。
ADV	副词

令牌	词性
	通常修饰动词的单词。它们还可以修饰形容词和其他副词。
AUX	助词 动词短语的动词附带的功能词。
CCONJ	并列连词 并列连词将句子中的单词、短语或从句连接起来，而不使它们从属于另一个句子。
CONJ	连词 连词连接句子中的单词、短语或从句。
DET	冠词 指定特定名词短语的冠词和其他单词。
INTJ	感叹词 用作感叹号或感叹号一部分的单词。
NOUN	名词 指定人、地点、事物、动物或想法的词语。
NUM	数字 表示数字的单词，通常是限定词、形容词或代词。

令牌	词性
O	其他 无法分配词性类别的单词。
PART	助词 与另一个单词或短语关联的功能词，用于传递含义。
PRON	代词 代替名词或名词短语的单词。
PROPN	专有名词 名词，是特定个人、地点或物体的名字。
PUNCT	标点符号 用于分隔文本的非字母字符。
SCONJ	从属连词 将从属子句与句子连接起来的连词。从属连词的一个例子是“因为”。
SYM	符号 类似单词的实体，例如美元符号 ($) 或数学符号。
VERB	谓词 表示事件和行动的词语。

有关词性的更多信息，请参阅通用依赖项网站上的通用POS标签。

这些操作返回用于标识单词以及该单词在文本中表示的词性的令牌。每个令牌代表源文本中的一个单词。它提供了单词在来源中的位置、单词在文本中采用的词性、Amazon Comprehend 对正确识别词性的置信度，以及从源文本中解析出来的单词。

以下是语法令牌列表的结构。为文档中的每个单词生成一个语法令牌。

```
{
   "SyntaxTokens": [
      {
         "BeginOffset": number,
         "EndOffset": number,
         "PartOfSpeech": {
            "Score": number,
            "Tag": "string"
         },
         "Text": "string",
         "TokenId": number
      }
   ]
}
```

每个令牌都提供以下信息：

- `BeginOffset` 和 `EndOffset`：提供单词在输入文本中的位置。
- `PartOfSpeech`：提供两条信息，`Tag` 用于标识词性，`Score` 表示 Amazon Comprehend Syntax 对正确识别词性的置信度。
- `Text`：提供已识别的单词。
- `TokenId`：提供令牌的标识符。标识符是令牌在令牌列表中的位置。

Amazon Comprehend 自定义

您可以根据自己的特定要求自定义 Amazon Comprehend，而无需具备构建基于机器学习的 NLP 解决方案所需的技能。使用自动机器学习 (AutoML)，Comprehend 自定义使用您提供的训练数据代表您构建自定义 NLP 模型。

输入文档处理：Amazon Comprehend 支持一步完成自定义分类和自定义实体识别的文档处理。例如，您可以将纯文本文档和半结构化文档（例如 PDF 文档、Microsoft Word 文档和图像）混合输入到自定义分析作业中。有关更多信息，请参阅 文档处理。

自定义分类：创建自定义分类模型（分类器），将您的文档整理到您自己的类别中。对于每个分类标签，请提供一组最能代表该标签的文档，并对您的分类器进行训练。经过训练后，分类器可用于任意数量的未加标签的文档集。您可以使用控制台获得无代码体验或安装最新的 AWS SDK。有关更多信息，请参阅 自定义分类。

自定义实体识别：创建自定义实体识别模型（识别器），该模型可以根据您的特定术语和名词短语分析文本。您可以训练识别器提取诸如保单编号之类的术语或暗示客户升级的短语。要训练模型，您需要提供实体列表和包含这些实体的一组文档。模型一旦经过训练，就可以针对模型提交分析作业以提取其自定义实体。有关更多信息，请参阅 自定义实体识别。

主题建模

您可以使用 Amazon Comprehend 来检查一组文档的内容，以确定共同的主题。例如，您可以向 Amazon Comprehend 提供一组新闻文章，它会确定主题，例如体育、政治或娱乐。文档中的文本不需要注释。

Amazon Comprehend 使用基于潜在狄利克雷分配的学习模型来确定一组文档中的主题。它检查每个文档以确定单词的上下文和含义。在整个文档集中，通常属于相同上下文的一组单词构成了一个主题。

一个单词与文档中的主题相关联，取决于该主题在文档中的普遍程度以及该主题与该词的相关程度。根据特定文档中的主题分布，可以将同一个单词与不同文档中的不同主题相关联。

例如，一篇主要谈论体育的文章中的“葡萄糖”一词可以分配给“体育”这个主题，而一篇关于“医学”的文章中，同一个词可以分配给“医学”这个主题

每个与主题相关的单词都有权重，表示该单词在多大程度上有助于定义该主题。权重表示在整个文档集中，与主题中的其他单词相比，该单词在主题中出现的次数。

为了获得最准确的结果，您应该向 Amazon Comprehend 提供尽可能多的语料库。要获得最佳效果：

- 在每个主题建模作业中，您应至少使用 1000 个文档。
- 每份文档的长度应至少为 3 个句子。
- 如果文档主要由数字数据组成，则应将其从语料库中删除。

主题建模是一个异步过程。您可以使用操作将文件清单从亚马逊 S3 存储桶提交到 Amazon Comprehend。StartTopicsDetectionJob响应将发送到 Amazon S3 存储桶。您可以配置输入和输出存储桶。获取您使用该ListTopicsDetectionJobs操作提交的主题建模作业的列表，并查看有关使用

该DescribeTopicsDetectionJob操作的作业的信息。发送到 Amazon S3 存储桶的内容可能包含客户内容。有关删除敏感数据的更多信息，请参阅如何清空 S3 存储桶？或如何删除 S3 存储桶？。

文档必须是 UTF-8 格式的文本文件。您可以通过两种方式提交文档。下表显示了这些选项。

格式	描述
每个文件一个文档	每个文件包含一个输入文档。这最适合大型文档的集合。
每行一个文档	输入是单个文件。文件中的每一行都被视为一个文档。这最适合简短的文档，例如社交媒体帖子。 每行必须以换行符 (LF, \n)、回车符 (CR, \r) 或两者兼有 (CRLF, \r\n) 结尾。Unicode 行分隔符 (u+2028) 不能用来结束一行。

有关更多信息，请参阅 InputDataConfig 数据类型。

在 Amazon Comprehend 处理您的文档集合后，它会返回一个包含两个文件（`topic-terms.csv` 和 `doc-topics.csv`）的压缩存档。有关输出文件的更多信息，请参阅OutputDataConfig。

第一个输出文件 `topic-terms.csv` 是集合中的主题列表。对于每个主题，默认情况下，该列表按权重排列主题列出根据其的热门术语。例如，如果您向 Amazon Comprehend 提供一组报纸文章，它可能会返回以下内容来描述该集合中的前两个主题：

主题	租期	重量
000	团队	0.118533
000	游戏	0.106072
000	player	0.031625
000	赛季	0.023633
000	play	0.021118

主题	租期	重量
000	码	0.024454
000	教练	0.016012
000	游戏	0.016191
000	足球	0.015049
000	四分卫	0.014239
001	杯子	0.205236
001	食物	0.040686
001	分钟	0.036062
001	add	0.029697
001	汤匙	0.028789
001	油	0.021254
001	胡椒	0.022205
001	茶匙	0.020040
001	葡萄酒	0.016588
001	糖	0.015101

权重表示给定主题中单词的概率分布。由于 Amazon Comprehend 仅返回每个主题的前 10 个单词，因此权重之和不会等于 1.0。在极少数情况下，如果主题中的单词少于 10 个，则权重之和将为 1.0。

通过查看单词在所有主题中的出现情况，按其判别力对单词进行排序。通常，这与它们的权重相同，但在某些情况下，例如表格中的“play”和“yard”这两个词，这会导致排序与权重不同。

您可以指定要返回的主题数量。例如，如果您要求 Amazon Comprehend 返回 25 个主题，则它会返回该集合中最突出的 25 个主题。Amazon Comprehend 可以在一个集合中检测多达 100 个主题。根据您对该领域的了解选择主题数量。可能需要一些实验才能得出正确的数字。

第二个文件 doc-topics.csv 列出了与主题相关的文档以及与该主题相关的文档比例。如果您指定 ONE_DOC_PER_FILE，则由文件名标识文档。如果您指定 ONE_DOC_PER_LINE，则由文件名和文件中以 0 为索引的行号来标识文档。例如，对于以每个文件包含一个文档的方式提交的文档集合，Amazon Comprehend 可能会返回以下内容：

文档	主题	比例
示例文档 1	000	0.999330137
示例文档 2	000	0.998532187
示例文档 3	000	0.998384574
...		
示例文档 N	000	3.57E-04

Amazon Comprehend 利用了 MBM 的词性还原列表数据集中的信息，该数据集根据开放数据库许可证 (ODbL) v1.0 在此处提供。

文档处理模式

Amazon Comprehend 支持三种文档处理模式。您选择的模式取决于您需要处理的文档数量以及查看结果所需的立即程度：

- 单文档同步：您使用单个文档调用 Amazon Comprehend，并收到同步响应，该响应会立即发送到您的应用程序（或控制台）。
- 多文档同步：您使用最多 25 个文档的集合调用 Amazon Comprehend API，然后收到同步响应。
- 异步批处理：对于大量文档集合，请将文档放入 Amazon S3 存储桶中，然后启动异步作业（使用控制台或 API 操作）来分析文档。Amazon Comprehend 将分析结果存储在您在请求中指定的 S3 存储桶/文件夹中。

主题

- 单个文档处理
- 多文档同步处理
- 异步批处理

单个文档处理

单文档操作是将文档分析结果直接返回到应用程序的同步操作。当您创建一次只能处理一个文档的交互式应用程序时，请使用单文档同步操作。

有关同步 API 操作的更多信息，请参阅 使用内置模型进行实时分析（适用于控制台）和 使用 API 进行实时分析。

多文档同步处理

如果您要处理多个文档，则可以使用 Batch* API 操作将多个文档一次发送到 Amazon Comprehend。每次请求中您最多可以发送 25 个文档。Amazon Comprehend 会发回一份响应列表，每个响应对应请求中的每个文档。通过这些操作发出的请求是同步的。您的应用程序调用该操作，然后等待服务的响应。

使用 Batch* 操作与为请求中的每个文档调用单个文档 API 相同。使用这些 API 可以提高应用程序的性能。

每个 API 的输入都是一个 JSON 结构，其中包含要处理的文档。对于除 BatchDetectDominantLanguage 之外的所有操作，您都必须设置输入语言。只能为每个请求设置一种输入语言。例如，以下是 BatchDetectEntities 操作的输入。它包含两个文档，并且是英文的。

```
{
   "LanguageCode": "en",
   "TextList": [
      "I have been living in Seattle for almost 4 years",
      "It is raining today in Seattle"
   ]
}
```

Batch* 操作的响应包含两个列表，即 ResultList 和 ErrorList。ResultList 包含成功处理的每个文档的一条记录。请求中每个文档的结果与您在文档上运行单个文档操作时得到的结果相同。根据输入文件中文档的顺序为每个文档的结果分配一个索引。BatchDetectEntities 操作的响应是：

```
{
   "ResultList"  : [
      {
         "Index": 0,
         "Entities": [
```

```
            {
               "Text": "Seattle",
               "Score": 0.95,
               "Type": "LOCATION",
               "BeginOffset": 22,
               "EndOffset": 29
            },
            {
               "Text": "almost 4 years",
               "Score": 0.89,
               "Type": "QUANTITY",
               "BeginOffset": 34,
               "EndOffset": 48
            }
         ]
      },
      {
         "Index": 1,
         "Entities": [
            {
              "Text": "today",
              "Score": 0.87,
              "Type": "DATE",
              "BeginOffset": 14,
              "EndOffset": 19
            },
            {
               "Text": "Seattle",
               "Score": 0.96,
               "Type": "LOCATION",
               "BeginOffset": 23,
               "EndOffset": 30
            }
         ]
      }
   ],
   "ErrorList": []
}
```

当请求中出现错误时，响应中会包含一个 ErrorList，用于标识包含错误的文档。文档由其在输入列表中的索引标识。例如，BatchDetectLanguage 操作的以下输入包含无法处理的文档：

```
{
```

```
   "TextList": [
     "hello friend",
     "$$$$$$",
     "hola amigo"
   ]
}
```

来自 Amazon Comprehend 的响应包括一个错误列表，用于标识包含错误的文档：

```
{
    "ResultList": [
        {
          "Index": 0,
          "Languages":[
            {
              "LanguageCode":"en",
              "Score": 0.99
            }
          ]
        },
        {
          "Index": 2
          "Languages":[
            {
              "LanguageCode":"es",
              "Score": 0.82
            }
          ]
        }
    ],
    "ErrorList": [
      {
        "Index": 1,
        "ErrorCode": "InternalServerException",
        "ErrorMessage": "Unexpected Server Error. Please try again."
      }
    ]
}
```

有关同步批处理 API 操作的更多信息，请参阅 实时批处理 API。

异步批处理

要分析大型文档和大型文档集合，请使用 Amazon Comprehend 异步操作。

要分析文档集合，通常执行以下步骤：

1. 将这些文档存储在 Amazon S3 存储桶中。
2. 启动一个或多个分析作业来分析文档。
3. 监控分析作业的进度。
4. 作业完成后，从 S3 存储桶中检索分析结果。

有关使用异步 API 操作的更多信息，请参阅 使用控制台运行分析作业（控制台）和 使用 API 执行异步分析作业。

Amazon Comprehend 支持的语言

Amazon Comprehend 的各种特征支持多种语言。支持的语言和支持它们的特征可以在下表中查看。

主题

支持的语言

Amazon Comprehend（检测主要语言功能除外）支持以下语言来实现一项或多项特征。

代码	Language
de	德语
en	English
es	西班牙语
it	意大利语
pt	葡萄牙语
fr	French
ja	日语
ko	韩语
hi	印地语
ar	阿拉伯语
zh	中文（简体）
zh-TW	中文（繁体）

> **Note**
>
> Amazon Comprehend 使用 RFC 5646 中的标识符识别语言，如果有 2 个字母的 ISO 639-1 标识符，并带有区域子标签（如有需要），则它会使用该标识符。否则，它将使用 ISO 639-2 3 个字母的代码。
>
> 有关 RFC 5646 的更多信息，请参阅 IETF 工具网站上用于识别语言的标签。

Amazon Comprehend 特征支持的语言

功能	支持的语言
主要语言	请参阅 主要语言。
实体	所有支持的语言。
关键短语	所有支持的语言。
检测 PII 实体	英语和西班牙语。
标注 PII 实体	英语和西班牙语。
情绪	所有支持的语言。
目标情绪	英语。
语法分析	德语 (de)、英语 (en)、西班牙语 (es)、法语 (fr)、意大利语 (it) 和葡萄牙语 (pt)。
主题建模	不取决于所使用的语言。不支持基于字符的语言，例如中文、日语和韩语。
自定义分类	纯文本模型支持以下语言：德语 (de)、英语 (en)、西班牙语 (es)、法语 (fr)、意大利语 (it) 和葡萄牙语 (pt)。 原生文档模型仅支持英文文档。

功能	支持的语言
自定义实体识别	德语 (de)、英语 (en)、西班牙语 (es)、法语 (fr)、意大利语 (it) 和葡萄牙语 (pt)。 PDF 和 Word 的自定义实体识别仅支持英文文档。

设置

首次使用 Amazon Comprehend 前，请完成以下任务。

任务设置

- 注册获取 AWS 账户
- 创建具有管理访问权限的用户
- 设置 AWS Command Line Interface (AWS CLI)
- 授予编程式访问权限

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。

设置 AWS Command Line Interface (AWS CLI)

您无需使用 AWS CLI 即可执行入门练习中的步骤。但是，本指南中的某些其他练习会需要它。如果您愿意，可以跳过此步骤并转至Amazon Comprehend 入门，然后 AWS CLI 再进行设置。

要安装和配置 AWS CLI

1. 安装 AWS CLI。有关说明，请参阅《AWS Command Line Interface 用户指南》中的以下主题：

   安装或更新最新版本的 AWS Command Line Interface

2. 配置 AWS CLI。有关说明，请参阅《AWS Command Line Interface 用户指南》中的以下主题：

   配置 AWS Command Line Interface

授予编程式访问权限

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 要使用”。 • 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。

哪个用户需要编程式访问权限？	目的	方式
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 • 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。

Amazon Comprehend 入门

以下练习使用 Amazon Comprehend 控制台创建和运行异步实体检测作业。本练习假定您熟悉 Amazon Simple Storage Service (Amazon S3)。有关更简单的示例，请参阅 使用内置模型进行实时分析。

创建实体检测作业

1. 登录 AWS Management Console 并打开 Amazon Comprehend 控制台，网址：https://console.aws.amazon.com/comprehend/
2. 从左侧菜单中选择分析作业，然后选择创建作业。
3. 在作业设置下，为作业命名。该名称在区域和账户中必须是唯一的。
4. 对于分析类型，选择实体。
5. 在 语言中，选择输入文档的语言。
6. 在输入数据下的数据来源中，选择示例文档。控制台将 S3 位置设置为包含公共示例的文件夹。
7. 在输出数据下的 S3 位置中，粘贴输出文件在 Amazon S3 中的 URL 或文件夹位置。
8. 在访问权限部分下，选择创建 IAM 角色。控制台创建一个新的 IAM 角色，该角色具有 Amazon Comprehend 访问输入和输出存储桶的适当权限。
9. 填完表单后，选择创建作业以创建并启动主题检测作业。

   新作业出现在作业列表中，状态字段会显示该作业的状态。该字段可以用于 `IN_PROGRESS` 正在处理的作业、 `COMPLETED` 已成功完成的作业以及 `FAILED` 存在错误的作业。
10. 选择作业以打开作业详细信息面板。
11. 在输出下的输出数据位置中，选择用于打开 Amazon S3 控制台的链接。
12. 在 Amazon S3 控制台中，选择下载并保存 `output.tar.gz` 文件。
13. 解压文件并将其保存为 Json 文件。
14. 有关实体类型和每个检测到的实体的字段的描述，请参阅 the section called “实体”。

使用 Amazon Comprehend 控制台进行分析

您可以使用 Amazon Comprehend 控制台实时分析文档或运行异步分析作业。

使用内置模型的实时分析，您可以识别实体、提取关键短语、检测主要语言、检测 PII、确定情绪、分析目标情绪和分析语法。

您可以使用内置模型运行分析作业，以查找实体、事件、短语、主要语言、情绪、目标情绪和个人身份信息 (PII) 等见解。您还可以运行主题建模作业。

该控制台还支持使用自定义模型进行实时和异步分析。有关更多信息，请参阅 自定义分类 和 自定义实体识别。

主题

使用内置模型进行实时分析

您可以使用 Amazon Comprehend 控制台对 UTF-8 编码的文本文档进行实时分析。该文档可以是英语或 Amazon Comprehend 支持的其他语言之一。结果显示在控制台中，以便您可以查看分析。

要开始分析文档，请登录 AWS Management Console 并打开 Amazon Comprehend 控制台。

您可以用自己的文本替换示例文本，然后选择分析来分析您的文本。在分析的文本下方，结果窗格显示有关文本的更多信息。

使用内置模型进行实时分析

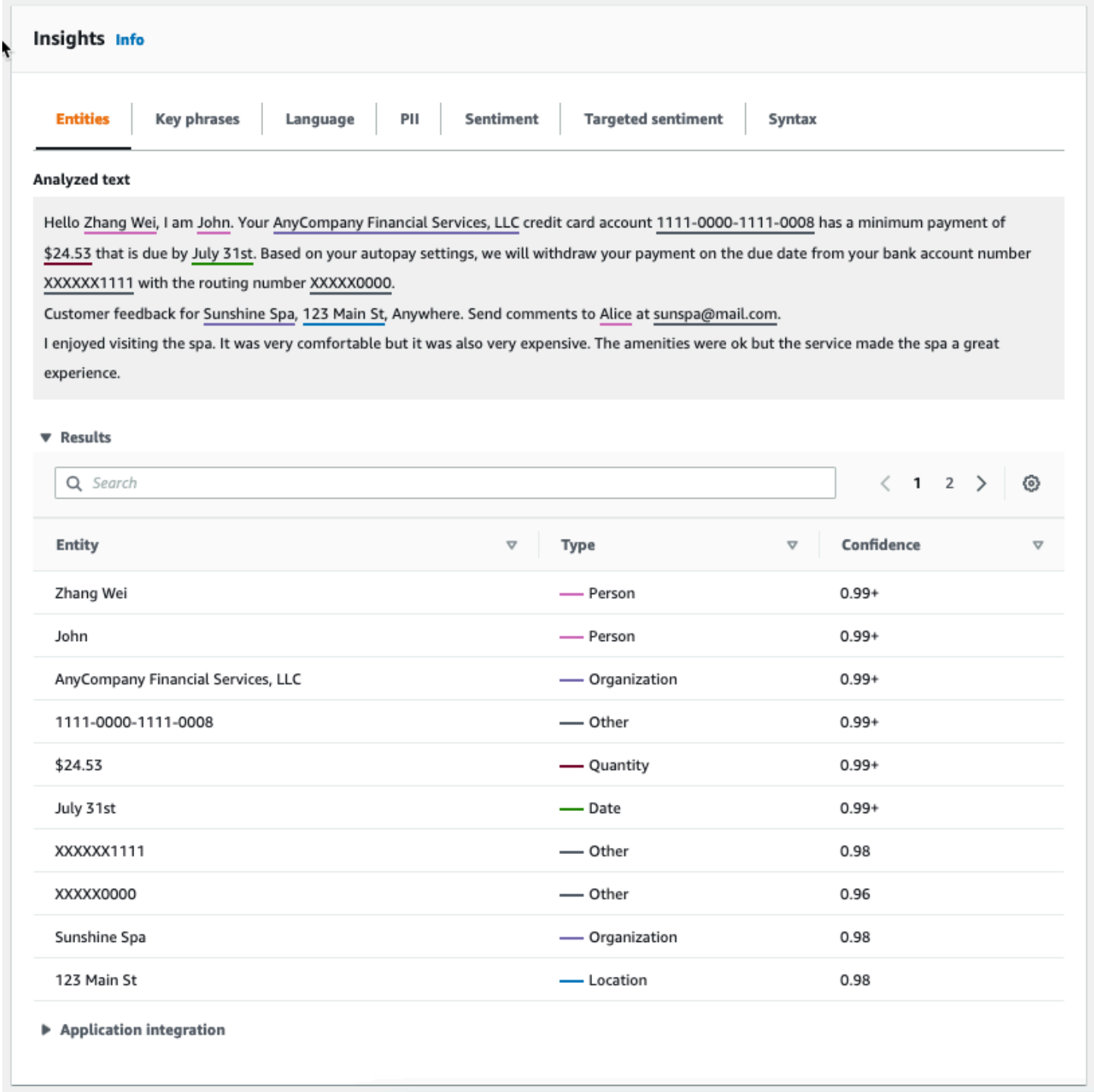
1. 登录 AWS Management Console 并打开亚马逊 Comprehend 控制台，网址为 https://console.aws.amazon.com/comprehend/
2. 从左侧菜单中，选择实时分析。
3. 在输入类型下，为分析类型选择内置。
4. 输入您要分析的文本。
5. 选择分析。控制台在见解面板中显示文本分析结果。见解面板包含每种见解类型的选项卡。以下各节介绍了见解类型的结果。

主题

- 实体
- 关键短语
- Language
- 个人身份信息（PII）
- 情绪
- 目标情绪
- 语法

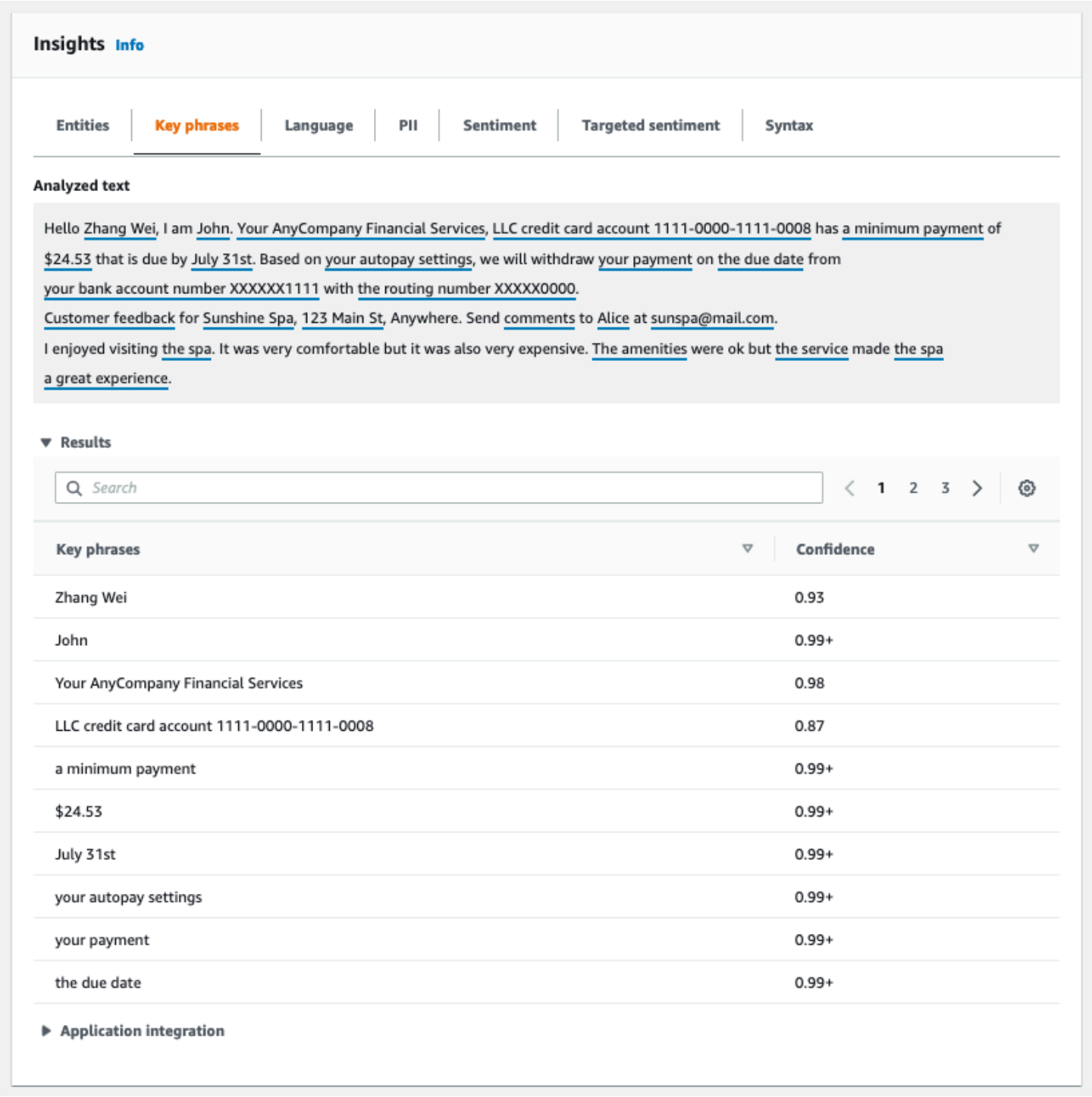
实体

实体选项卡列出了每个实体、其类别以及 Amazon Comprehend 在输入文本中检测到的置信度。结果采用颜色编码，以表示不同的实体类型，例如组织、地点、日期和个人。有关更多信息，请参阅 实体。



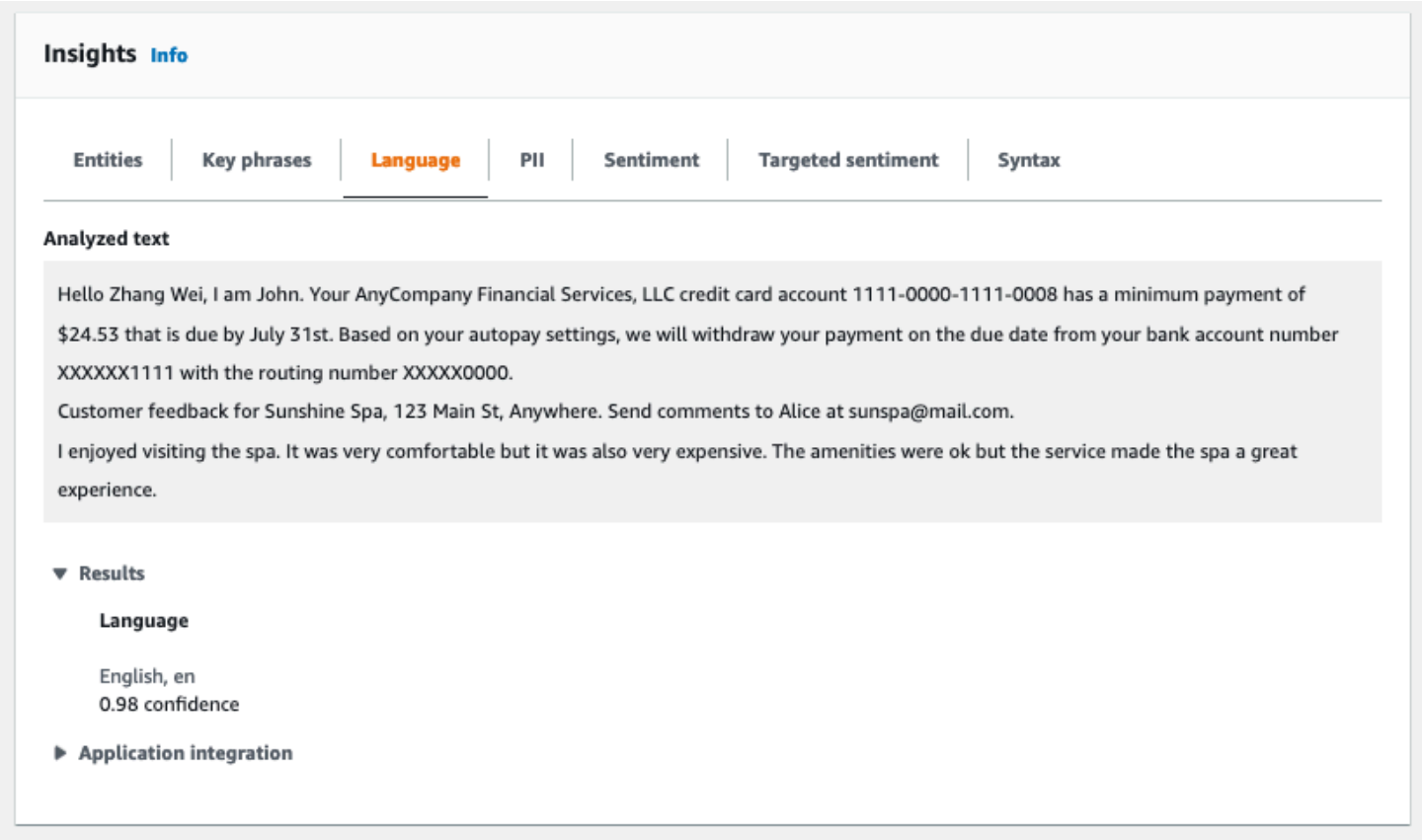
关键短语

关键短语选项卡列出了 Amazon Comprehend 在输入文本中检测到的关键名词短语以及相关的置信度。有关更多信息，请参阅 关键短语。



Language

语言选项卡显示文本的主导语言以及 Amazon Comprehend 对正确检测到主导语言的置信度。Amazon Comprehend 可识别 100 种语言。有关更多信息，请参阅 主要语言。



个人身份信息（PII）

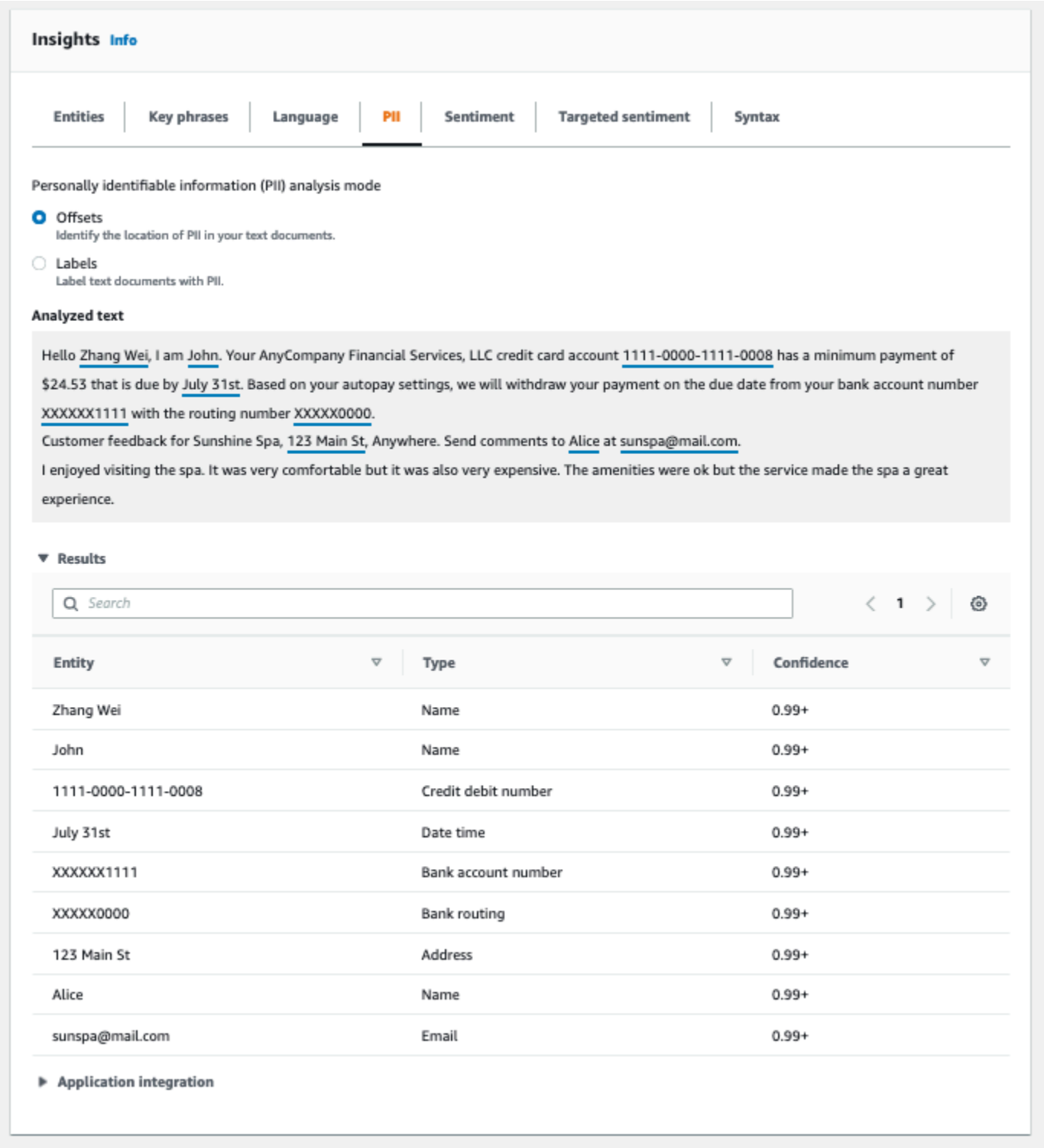
PII 选项卡列出了输入文本中包含个人身份信息（PII）的实体。PII 实体是对可用于识别个人身份的个人数据的文本引用，例如地址、银行账号或电话号码。有关更多信息，请参阅 检测 PII 实体。

PII 选项卡提供两种分析模式：

- 偏移量
- 标签

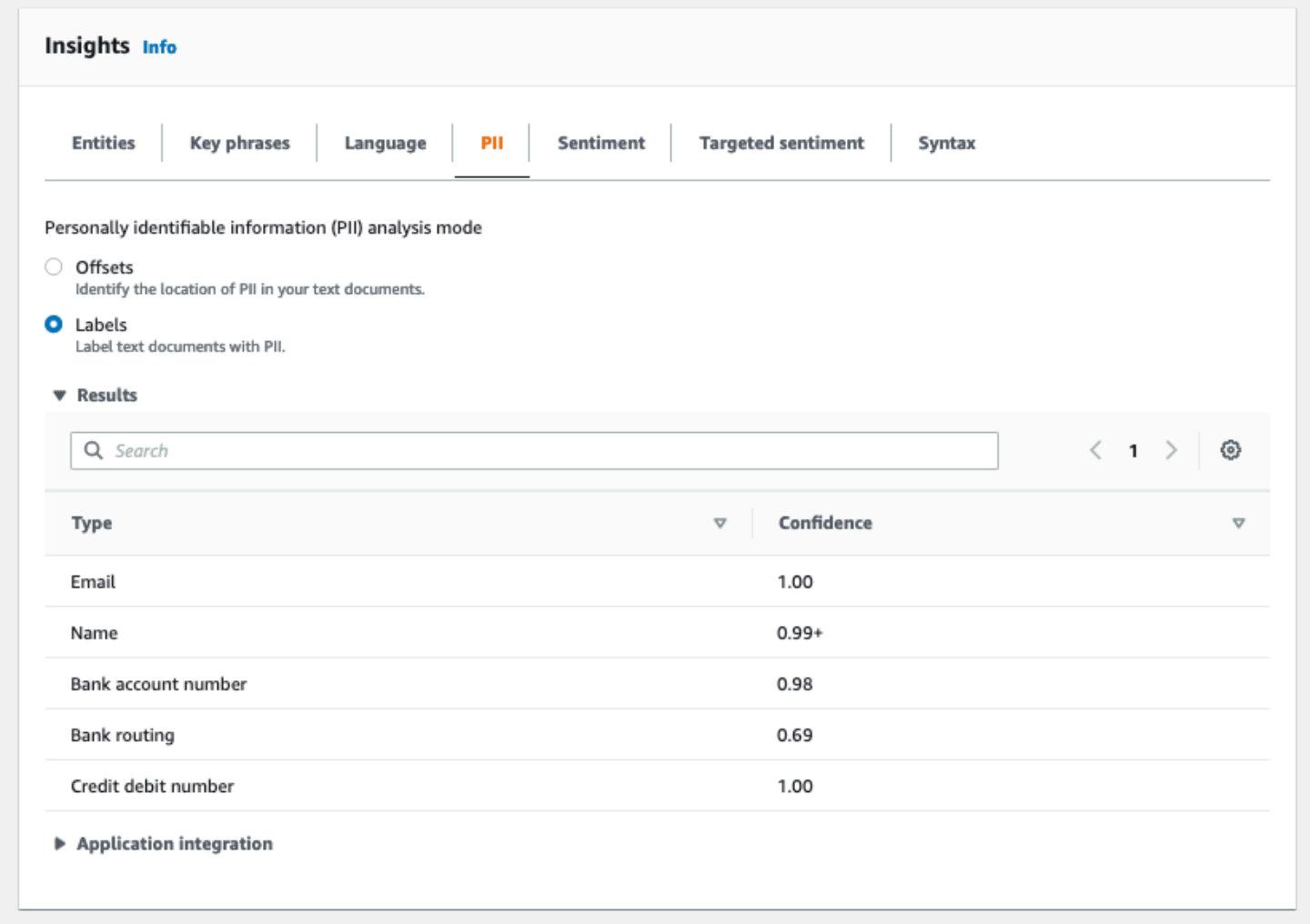
偏移量

偏移量分析模式可识别 PII 在文本文档中的位置。有关更多信息，请参阅 查找 PII 实体。



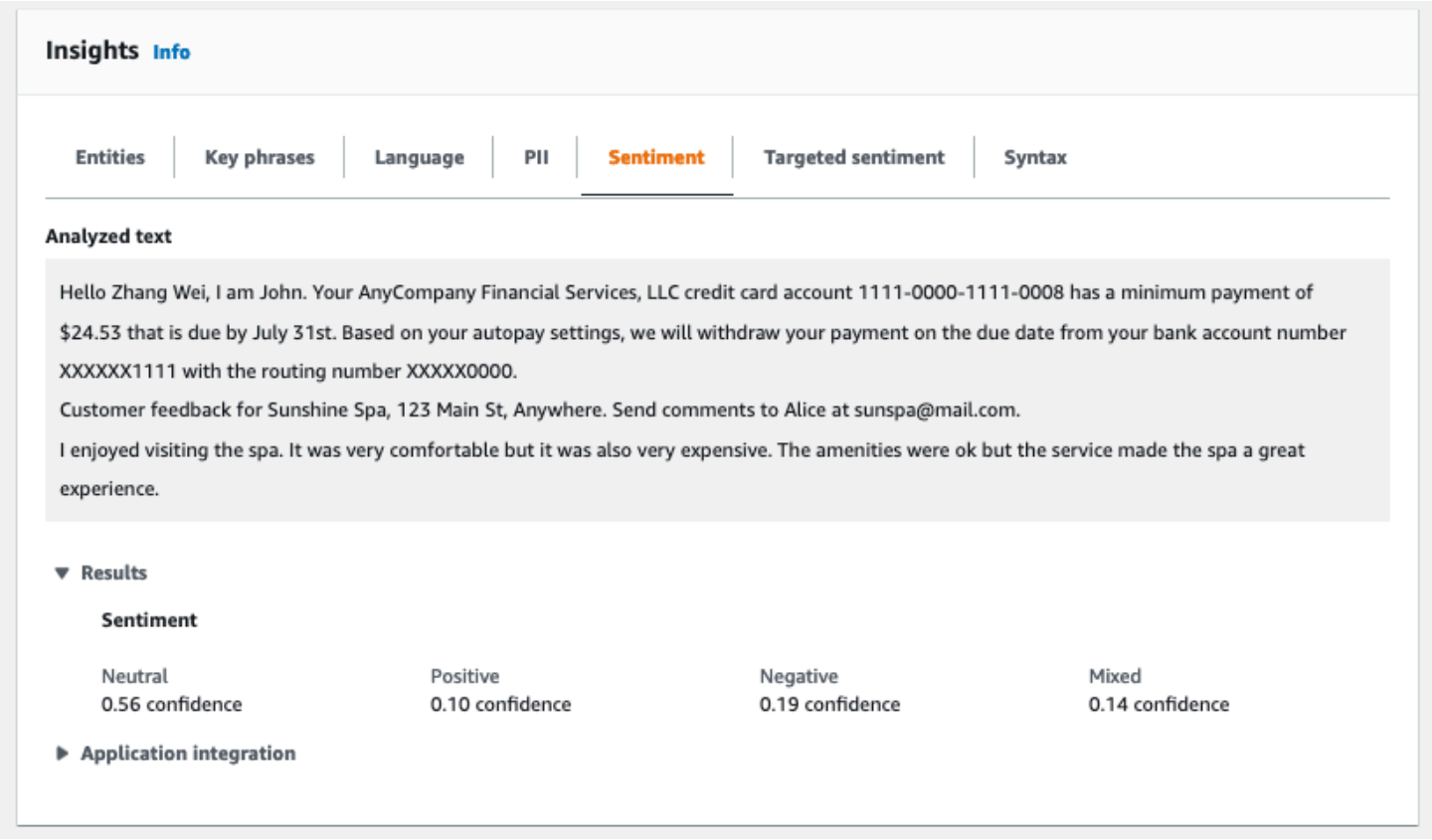
标签

标签分析模式检查文本文档中是否存在 PII，并返回已识别的 PII 实体类型的标签。有关更多信息，请参阅 标注 PII 实体。



情绪

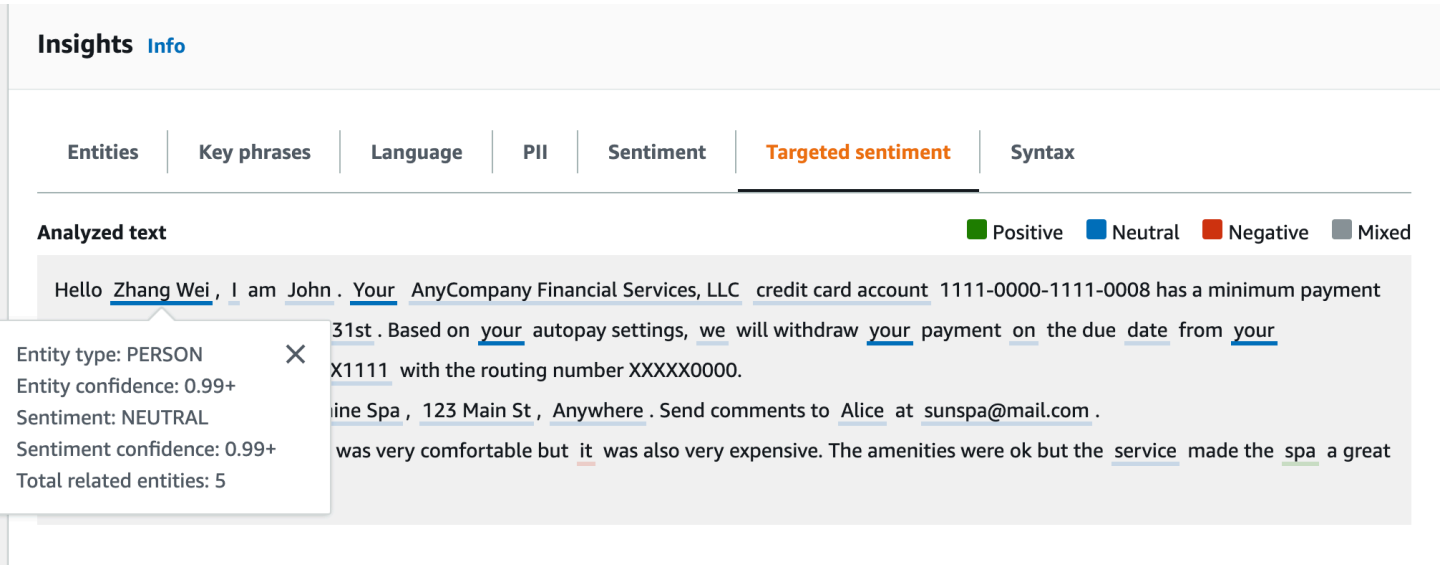
情绪选项卡显示文本的主导情绪。情绪可以被评为中性、正面、负面或混合的。在这种情况下，每种情绪都有置信度评级，这是 Amazon Comprehend 对该情绪占主导地位的估计。有关更多信息，请参阅情绪。



目标情绪

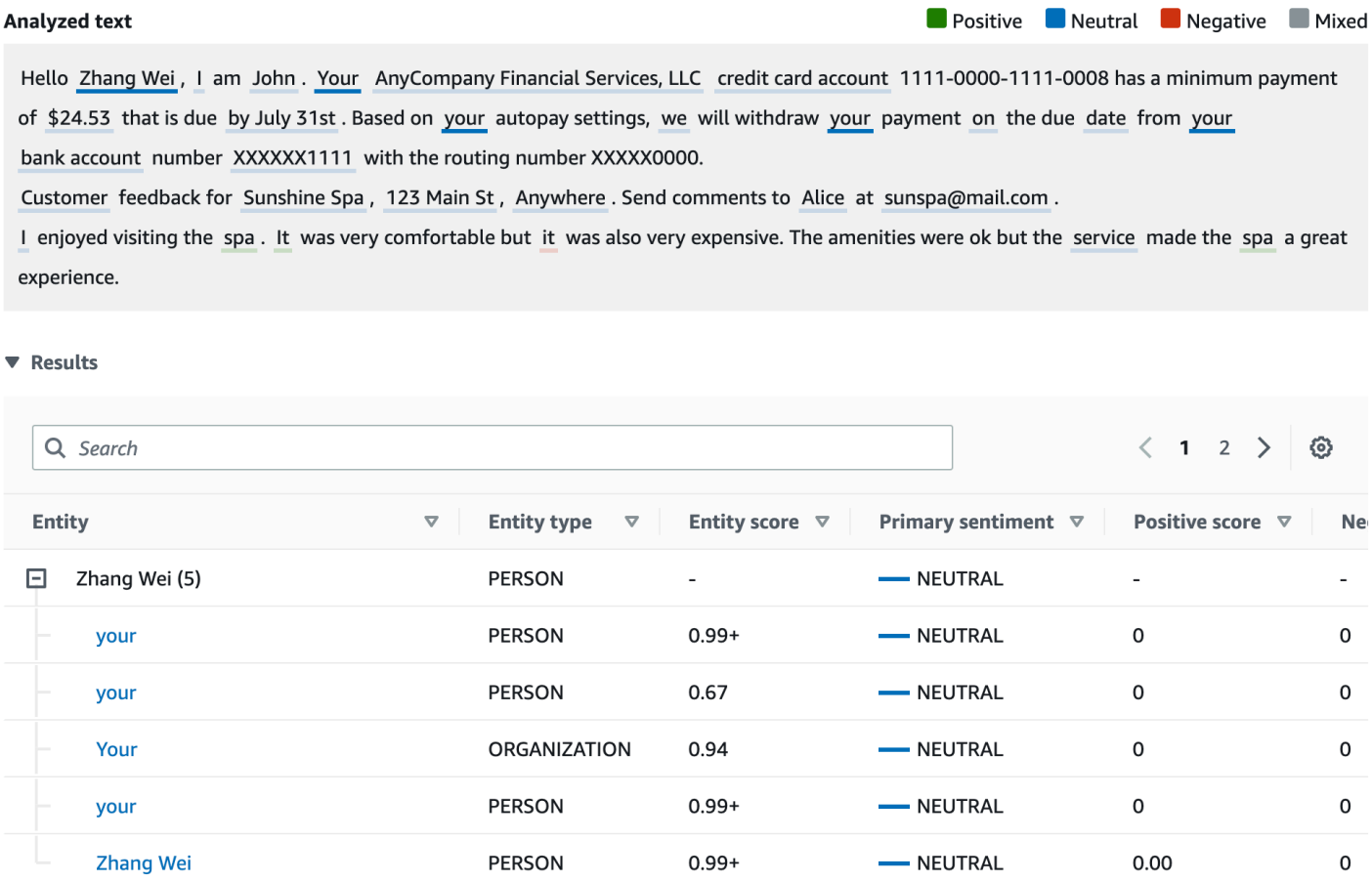
目标情绪分析可以识别对文本中提到的实体所表达的情绪。Amazon Comprehend 会为每次提及的实体分配情绪评级，以及置信度评级和其他信息。情绪评级可以是中性、正面、负面或混合的。

在已分析文本面板中，控制台会为每个已分析的实体加下划线。下划线文本的颜色表示该实体的整体情绪。如果将光标悬停在实体上，控制台将在弹出窗口中显示其他信息。



结果表提供了每个实体的更多详细信息。如果多次提及同一个实体（称为共同引用组），则表格会将这些提及显示为一组与主实体关联的可折叠行。

在以下示例中，该实体是一个名叫张伟的人。目标情绪分析可以识别出，每次提及你的内容都是指同一个人。控制台将这些提及显示为主实体的子条目。

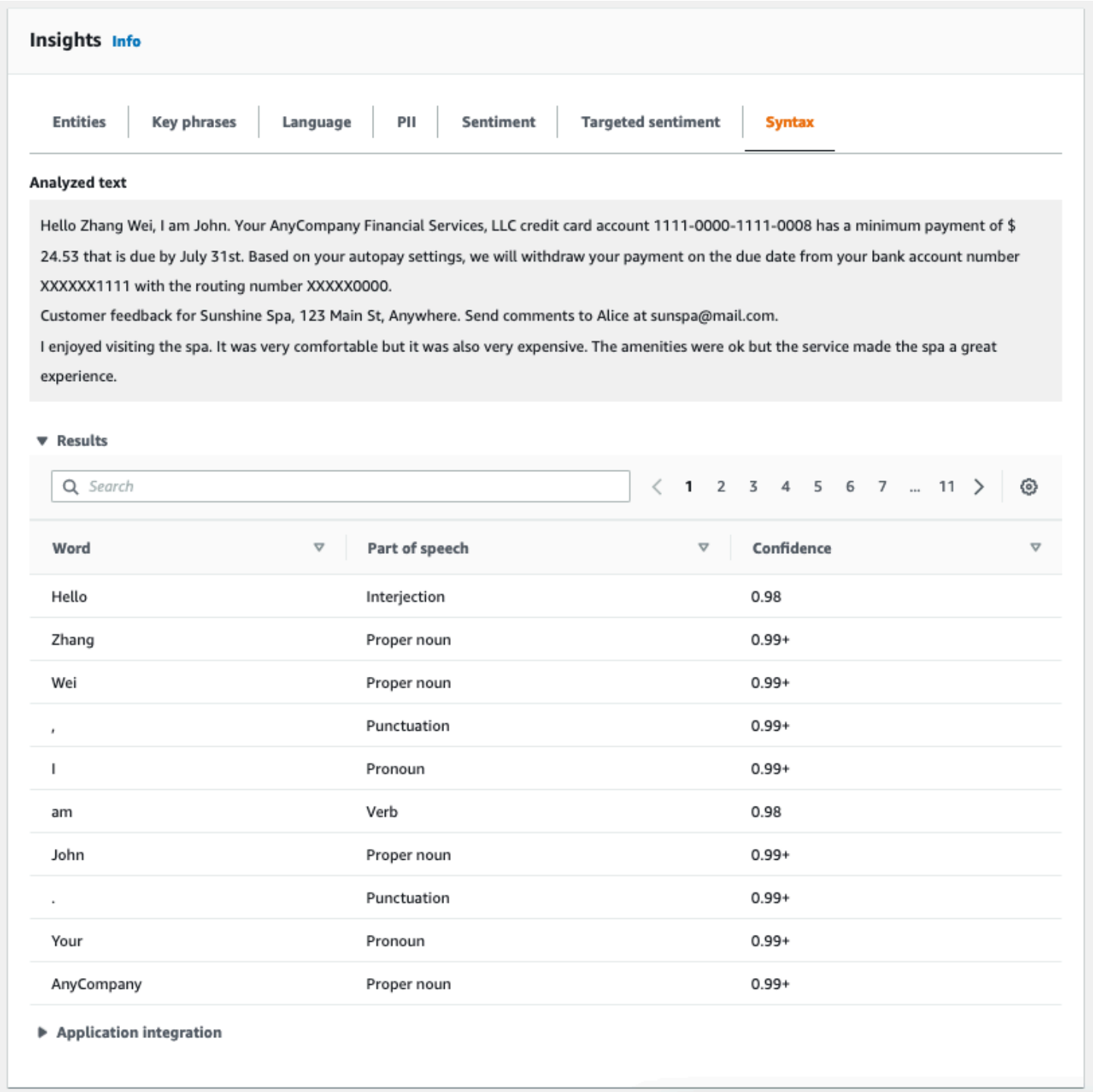


如果您正在分析的文本不包含任何目标情绪 实体类型，则目标情绪分析将显示一个空的结果字段。

有关如何使用控制台进行目标情绪实时分析的更多信息，请参阅 使用控制台进行实时分析。

语法

语法选项卡显示文本中每个元素的细分，以及其词性和相关的置信度分数。有关更多信息，请参阅 语法分析。



使用控制台运行分析作业

您可以使用 Amazon Comprehend 控制台创建和管理异步分析作业。您的作业会分析存储在 Amazon S3 中的文档，以查找事件、短语、主要语言、情绪或个人身份信息 (PII) 等实体。

创建分析作业

1. 登录 AWS Management Console 并打开亚马逊 Comprehend 控制台，网址为 https://console.aws.amazon.com/comprehend/
2. 从左侧菜单中选择分析作业，然后选择创建作业。
3. 在作业设置下，为分析作业指定一个唯一的名称。
4. 对于分析类型，请选择内置分析类型之一。

   如果选择 “主要语言” 或 “主题建模”，则可以跳过下一步。
5. 根据您选择的分析类型，控制台会显示以下一个或多个附加字段：

   - 除初级语言和主题建模外，所有内置分析类型都需要语言。

     选择输入文档的语言。
   - 事件分析类型需要使用目标事件类型。

     选择要在输入文档中检测的事件类型。有关支持的事件类型的更多信息，请参阅 事件类型。
   - PII 分析类型需要 PII 检测设置。

     选择输出模式。有关 PII 检测设置的更多信息，请参阅 检测 PII 实体。
6. 在输入数据下，指定输入文档在 Amazon S3 中的位置：
   - 要分析您自己的文档，请选择我的文档，然后选择浏览 S3 提供包含您的文件的存储桶或文件夹的路径。
   - 要分析 Amazon Comprehend 提供的示例，请选择示例文档。在这种情况下，Amazon Comprehend 使用由 AWS管理的存储桶，而您无需指定存储位置。
7. （可选）对于输入格式，请为输入文件指定以下格式之一：

   - 每个文件一个文档：每个文件包含一个输入文档。这最适合大型文档的集合。
   - 每行一个文档：输入一个或多个文件。文件中的每一行都被视为一个文档。这最适合简短的文档，例如社交媒体帖子。每行必须以换行符 (LF, \n)、回车符 (CR, \r) 或两者兼有 (CRLF, \r\n) 结尾。您不能使用 UTF-8 行分隔符 (u+2028) 来结束一行。
8. 在输出数据下，选择浏览 S3。选择 Amazon S3 存储桶或文件夹，以便 Amazon Comprehend 将分析生成的输出数据写入其中。
9. （可选）要加密作业的输出结果，请选择加密。然后选择是使用与当前账户关联的 KMS 密钥，还是使用来自其他账户的 KMS 密钥：
   - 如果您使用的是与当前账户关联的密钥，请为 KMS 密钥 ID 选择密钥别名或 ID。

- 如果您使用与其他账户关联的密钥，请在 KMS 密钥 ID 下输入密钥别名或 ID 的 ARN。

> **Note**
>
> 有关创建和使用 KMS 密钥以及相关加密的更多信息，请参阅密钥管理服务 (KMS)。

10. 在访问权限下，提供一个 IAM 角色：

    - 授予对 Amazon S3 输入文档位置的读取访问权限。
    - 授予对 Amazon S3 输出文档位置的写入访问权限。
    - 包括允许 `comprehend.amazonaws.com` 服务委托人担任角色并获得其权限的信任策略。

    如果您还没有具有这些权限和适当信任策略的 IAM 角色，请选择创建 IAM 角色来创建一个。

11. 填完表单后，选择创建作业以创建并启动主题检测作业。

新作业出现在作业列表中，状态字段会显示该作业的状态。该字段可以用于 `IN_PROGRESS` 正在处理的作业、 `COMPLETED` 已成功完成的作业以及 `FAILED` 存在错误的作业。您可以单击作业以获取有关该作业的更多信息，包括任何错误消息。

作业完成后，Amazon Comprehend 会将分析结果存储在您为作业指定的输出 Amazon S3 位置。有关每种见解类型的分析结果的描述，请参阅 洞察。

使用 Amazon Comprehend API

Amazon Comprehend API 支持执行实时（同步）分析的操作以及启动和管理异步分析作业的操作。

您可以直接使用 Amazon Comprehend API 运算符，也可以使用 CLI 或 SDK 之一。本章中的示例使用 CLI、Python SDK 和 Java SDK。

要运行 AWS CLI 和 Python 示例，必须安装 AWS CLI。有关更多信息，请参阅 设置 AWS Command Line Interface (AWS CLI)。

要运行 Java 示例，您需要安装 AWS SDK for Java。有关安装适用于 Java 的 SDK 的说明，请参阅设置适用于 Java 的 AWS 开发工具包。

主题

将 Amazon Comprehend 与 SDK 配合 AWS 使用

AWS 软件开发套件 (SDK) 可用于许多流行的编程语言。每个软件开发工具包都提供 API、代码示例和文档，使开发人员能够更轻松地以其首选语言构建应用程序。

SDK 文档	代码示例
AWS SDK for C++	AWS SDK for C++ 代码示例
AWS CLI	AWS CLI 代码示例
AWS SDK for Go	AWS SDK for Go 代码示例
AWS SDK for Java	AWS SDK for Java 代码示例
AWS SDK for JavaScript	AWS SDK for JavaScript 代码示例
AWS SDK for Kotlin	AWS SDK for Kotlin 代码示例
AWS SDK for .NET	AWS SDK for .NET 代码示例

SDK 文档	代码示例
AWS SDK for PHP	AWS SDK for PHP 代码示例
AWS Tools for PowerShell	PowerShell 代码示例工具
AWS SDK for Python (Boto3)	AWS SDK for Python (Boto3) 代码示例
AWS SDK for Ruby	AWS SDK for Ruby 代码示例
AWS SDK for Rust	AWS SDK for Rust 代码示例
适用于 SAP ABAP 的 AWS SDK	适用于 SAP ABAP 的 AWS SDK 代码示例
AWS SDK for Swift	AWS SDK for Swift 代码示例

示例可用性

找不到所需的内容？ 通过使用此页面底部的提供反馈链接请求代码示例。

使用 API 进行实时分析

以下示例演示如何使用 AWS CLI 和适用于 .NET、Java和 Python 的 AWS SDK 使用 Amazon Comprehend API 进行实时分析。通过示例了解 Amazon Comprehend 同步操作，并将其作为您自己的应用程序的构建块。

本节中的 .NET 示例使用 AWS SDK for .NET。您可以使用 AWS Toolkit for Visual Studio 通过 .NET 开发 AWS 应用程序。它包括有用模板和 AWS Explorer，用于部署应用程序和管理服务。有关 .NET 开发人员对于 AWS 的观点，请参阅《AWS .NET 开发人员指南》。

主题

- 检测占主要语言
- 检测命名实体
- 检测关键短语
- 确定情绪
- 目标情绪的实时分析

- 检测语法
- 实时批处理 API

检测占主要语言

要确定文本中使用的主导语言，请使用DetectDominantLanguage操作。要批量检测多达 25 个文档中的主要语言，请使用该BatchDetectDominantLanguage操作。有关更多信息，请参阅 实时批处理 API。

主题

- 使用 AWS Command Line Interface
- 使用 AWS SDK for Java、适用于 Python 的 SDK 或 AWS SDK for .NET

使用 AWS Command Line Interface

以下示例演示了如何在 AWS CLI 中使用 `DetectDominantLanguage` 操作。

此示例的格式适用于 Unix、Linux 和 macOS。对于 Windows，请将每行末尾的反斜杠 (\) Unix 行继续符替换为脱字号 (^)。

```
aws comprehend detect-dominant-language \
    --region region \
    --text "It is raining today in Seattle."
```

Amazon Comprehend 的响应如下：

```
{
    "Languages": [
        {
            "LanguageCode": "en",
            "Score": 0.9793661236763
        }
    ]
}
```

使用 AWS SDK for Java、适用于 Python 的 SDK 或 AWS SDK for .NET

有关如何确定主要语言的 SDK 示例，请参阅 `DetectDominantLanguage`与 AWS SDK 或 CLI 配合使用。

检测命名实体

要确定文档中的命名实体，请使用DetectEntities操作。要批量检测最多 25 个文档中的实体，请使用BatchDetectEntities操作。有关更多信息，请参阅 实时批处理 API。

主题

- 使用 AWS Command Line Interface
- 使用 AWS SDK for Java、适用于 Python 的 SDK 或 AWS SDK for .NET

使用 AWS Command Line Interface

以下示例演示了如何在 AWS CLI 中使用 `DetectEntities` 操作。您必须指定输入文本的语言。

此示例的格式适用于 Unix、Linux 和 macOS。对于 Windows，请将每行末尾的反斜杠 (\) Unix 行继续符替换为脱字号 (^)。

```
aws comprehend detect-entities \
    --region region \
    --language-code "en" \
    --text "It is raining today in Seattle."
```

Amazon Comprehend 的响应如下：

```
{
    "Entities": [
        {
            "Text": "today",
            "Score": 0.97,
            "Type": "DATE",
            "BeginOffset": 14,
            "EndOffset": 19
        },
        {
            "Text": "Seattle",
            "Score": 0.95,
            "Type": "LOCATION",
            "BeginOffset": 23,
            "EndOffset": 30
        }
    ],
    "LanguageCode": "en"
```

```
}
```

使用 AWS SDK for Java、适用于 Python 的 SDK 或 AWS SDK for .NET

有关如何确定主要语言的 SDK 示例，请参阅 DetectEntities与 AWS SDK 或 CLI 配合使用。

检测关键短语

要确定文本中使用的关键名词短语，请使用DetectKeyPhrases操作。要批量检测多达 25 个文档中的关键名词短语，请使用该BatchDetectKeyPhrases操作。有关更多信息，请参阅 实时批处理 API。

主题

- 使用 AWS Command Line Interface
- 使用 AWS SDK for Java、适用于 Python 的 SDK 或 AWS SDK for .NET

使用 AWS Command Line Interface

以下示例演示了如何在 AWS CLI 中使用 DetectKeyPhrases 操作。您必须指定输入文本的语言。

此示例的格式适用于 Unix、Linux 和 macOS。对于 Windows，请将每行末尾的反斜杠 (\) Unix 行继续符替换为脱字号 (^)。

```
aws comprehend detect-key-phrases \
    --region region \
    --language-code "en" \
    --text "It is raining today in Seattle."
```

Amazon Comprehend 的响应如下：

```
{
    "LanguageCode": "en",
    "KeyPhrases": [
        {
            "Text": "today",
            "Score": 0.89,
            "BeginOffset": 14,
            "EndOffset": 19
        },
        {
            "Text": "Seattle",
            "Score": 0.91,
```

```
            "BeginOffset": 23,
            "EndOffset": 30
        }
    ]
}
```

使用 AWS SDK for Java、适用于 Python 的 SDK 或 AWS SDK for .NET

有关检测关键短语的 SDK 示例，请参阅 DetectKeyPhrases与 AWS SDK 或 CLI 配合使用。

确定情绪

Amazon Comprehend 提供以下 API 操作，以便分析情绪：

- DetectSentiment— 确定文档的整体情感情绪。
- BatchDetectSentiment— 在一批中确定最多 25 个文档的总体情绪。有关更多信息，请参阅 实时批处理 API。
- StartSentimentDetectionJob— 为一组文档启动异步情绪检测作业。
- ListSentimentDetectionJobs— 返回您已提交的情绪检测任务列表。
- DescribeSentimentDetectionJob— 获取与指定情绪检测作业关联的属性（包括状态）。
- StopSentimentDetectionJob— 停止指定的正在进行的情绪作业。

主题

- 使用AWS Command Line Interface
- 使用 AWS SDK for Java、适用于 Python 的 SDK 或 AWS SDK for .NET

使用AWS Command Line Interface

以下示例演示了如何在 AWS CLI 中使用 DetectSentiment 操作。此示例指定了输入文本的语言。

此示例的格式适用于 Unix、Linux 和 macOS。对于 Windows，请将每行末尾的反斜杠 (\) Unix 行继续符替换为脱字号 (^)。

```
aws comprehend detect-sentiment \
    --region region \
    --language-code "en" \
    --text "It is raining today in Seattle."
```

Amazon Comprehend 的响应如下：

```
{
    "SentimentScore": {
        "Mixed": 0.014585512690246105,
        "Positive": 0.31592071056365967,
        "Neutral": 0.5985543131828308,
        "Negative": 0.07093945890665054
    },
    "Sentiment": "NEUTRAL",
    "LanguageCode": "en"
}
```

使用 AWS SDK for Java、适用于 Python 的 SDK 或 AWS SDK for .NET

有关确定输入文本情绪的 SDK 示例，请参阅 DetectSentiment与 AWS SDK 或 CLI 配合使用。

目标情绪的实时分析

Amazon Comprehend 提供以下 API 操作，用于目标情绪的实时分析：

- DetectTargetedSentiment— 分析文档中提及的实体的情绪。
- BatchDetectTargetedSentiment— 批量分析最多 25 个文档的目标情绪。有关更多信息，请参阅 实时批处理 API。

如果您正在分析的文本不包含任何目标情绪 实体类型，则 API 将返回一个空的实体数组。

使用AWS Command Line Interface

以下示例演示了如何在 AWS CLI 中使用 DetectTargetedSentiment 操作。此示例指定了输入文本的语言。

此示例的格式适用于 Unix、Linux 和 macOS。对于 Windows，请将每行末尾的反斜杠 (\) Unix 行继续符替换为脱字号 (^)。

```
aws comprehend detect-targeted-sentiment \
    --region region \
    --language-code "en" \
    --text "The burger was cooked perfectly but it was cold. The service was OK."
```

Amazon Comprehend 的响应如下：


```
{
"Entities": [
    {
      "DescriptiveMentionIndex": [
        0
      ],
      "Mentions": [
        {
          "BeginOffset": 4,
          "EndOffset": 10,
          "Score": 1,
          "GroupScore": 1,
          "Text": "burger",
          "Type": "OTHER",
          "MentionSentiment": {
            "Sentiment": "POSITIVE",
            "SentimentScore": {
              "Mixed": 0.001515,
              "Negative": 0.000822,
              "Neutral": 0.000243,
              "Positive": 0.99742
            }
          }
        },
        {
          "BeginOffset": 36,
          "EndOffset": 38,
          "Score": 0.999843,
          "GroupScore": 0.999661,
          "Text": "it",
          "Type": "OTHER",
          "MentionSentiment": {
            "Sentiment": "NEGATIVE",
            "SentimentScore": {
              "Mixed": 0,
              "Negative": 0.999996,
              "Neutral": 0.000004,
              "Positive": 0
            }
          }
        }
      ]
```

```
    },
    {
      "DescriptiveMentionIndex": [
        0
      ],
      "Mentions": [
        {
          "BeginOffset": 53,
          "EndOffset": 60,
          "Score": 1,
          "GroupScore": 1,
          "Text": "service",
          "Type": "ATTRIBUTE",
          "MentionSentiment": {
            "Sentiment": "NEUTRAL",
            "SentimentScore": {
              "Mixed": 0.000033,
              "Negative": 0.000089,
              "Neutral": 0.993325,
              "Positive": 0.006553
            }
          }
        }
      ]
    }
  ]
}
```

检测语法

要解析文本以提取单个单词并确定每个单词的语音部分，请使用DetectSyntax操作。要批量解析最多 25 个文档的语法，请使用BatchDetectSyntax操作。有关更多信息，请参阅 实时批处理 API。

主题

- 使用 AWS Command Line Interface。
- 使用 AWS SDK for Java、适用于 Python 的 SDK 或 AWS SDK for .NET

使用 AWS Command Line Interface。

以下示例演示了如何在 AWS CLI 中使用 `DetectSyntax` 操作。此示例指定了输入文本的语言。

此示例的格式适用于 Unix、Linux 和 macOS。对于 Windows，请将每行末尾的反斜杠 (\) Unix 行继续符替换为脱字号 (^)。

```
aws comprehend detect-syntax \
   --region region \
   --language-code "en" \
   --text "It is raining today in Seattle."
```

Amazon Comprehend 的响应如下：

```
{
    "SyntaxTokens": [
        {
            "Text": "It",
            "EndOffset": 2,
            "BeginOffset": 0,
            "PartOfSpeech": {
                "Tag": "PRON",
                "Score": 0.8389829397201538
            },
            "TokenId": 1
        },
        {
            "Text": "is",
            "EndOffset": 5,
            "BeginOffset": 3,
            "PartOfSpeech": {
                "Tag": "AUX",
                "Score": 0.9189288020133972
            },
            "TokenId": 2
        },
        {
            "Text": "raining",
            "EndOffset": 13,
            "BeginOffset": 6,
            "PartOfSpeech": {
                "Tag": "VERB",
                "Score": 0.9977611303329468
            },
            "TokenId": 3
        },
        {
```

```
            "Text": "today",
            "EndOffset": 19,
            "BeginOffset": 14,
            "PartOfSpeech": {
                "Tag": "NOUN",
                "Score": 0.9993606209754944
            },
            "TokenId": 4
        },
        {
            "Text": "in",
            "EndOffset": 22,
            "BeginOffset": 20,
            "PartOfSpeech": {
                "Tag": "ADP",
                "Score": 0.9999061822891235
            },
            "TokenId": 5
        },
        {
            "Text": "Seattle",
            "EndOffset": 30,
            "BeginOffset": 23,
            "PartOfSpeech": {
                "Tag": "PROPN",
                "Score": 0.9940338730812073
            },
            "TokenId": 6
        },
        {
            "Text": ".",
            "EndOffset": 31,
            "BeginOffset": 30,
            "PartOfSpeech": {
                "Tag": "PUNCT",
                "Score": 0.9999997615814209
            },
            "TokenId": 7
        }
    ]
}
```

使用 AWS SDK for Java、适用于 Python 的 SDK 或 AWS SDK for .NET

有关检测输入文本语法的 SDK 示例，请参阅 DetectSyntax与 AWS SDK 或 CLI 配合使用。

实时批处理 API

要发送最多 25 个文档的批，您可以使用 Amazon Comprehend 实时批处理操作。调用批处理操作与为请求中的每个文档调用单个文档 API 相同。使用批处理 API 可以提高应用程序的性能。有关更多信息，请参阅 多文档同步处理。

主题

- 使用 AWS CLI 进行批处理
- 使用 AWS SDK for .NET 进行批处理

使用 AWS CLI 进行批处理

这些示例显示如何通过 AWS Command Line Interface 使用批处理 API 操作。除 BatchDetectDominantLanguage 之外的所有操作都使用以下名为 process.json JSON 文件作为输入。对于该操作，LanguageCode 实体不包括在内。

JSON 文件中的第三个文档 ("$$$$$$$$") 会导致批处理时出错。它包含在内是为了使操作在响应BatchItemError中包含一个。

```
{
   "LanguageCode": "en",
   "TextList": [
      "I have been living in Seattle for almost 4 years",
      "It is raining today in Seattle",
      "$$$$$$$$"
   ]
}
```

此示例的格式适用于 Unix、Linux 和 macOS。对于 Windows，请将每行末尾的反斜杠 (\) Unix 行继续符替换为脱字号 (^)。

主题

- 使用批处理 (AWS CLI) 检测主要语言
- 使用批处理 (AWS CLI) 检测实体

- 使用批处理 (AWS CLI) 检测关键短语
- 使用批处理 (AWS CLI) 检测情绪

使用批处理 (AWS CLI) 检测主要语言

该BatchDetectDominantLanguage操作决定了批次中每个文档的主要语言。有关 Amazon Comprehend 可以检测的语言列表，请参阅 主要语言。以下 AWS CLI 命令调用了 BatchDetectDominantLanguage 操作。

```
aws comprehend batch-detect-dominant-language \
    --endpoint endpoint \
    --region region \
    --cli-input-json file://path to input file/process.json
```

以下示例是 BatchDetectDominantLanguage 操作的响应：

```
{
    "ResultList": [
        {
          "Index": 0,
          "Languages":[
            {
              "LanguageCode":"en",
              "Score": 0.99
            }
          ]
        },
        {
          "Index": 1
          "Languages":[
            {
              "LanguageCode":"en",
              "Score": 0.82
            }
          ]
        }
    ],
    "ErrorList": [
      {
        "Index": 2,
        "ErrorCode": "InternalServerException",
```

```
        "ErrorMessage": "Unexpected Server Error. Please try again."
      }
    ]
}
```

使用批处理 (AWS CLI) 检测实体

使用该BatchDetectEntities操作查找一批文档中存在的实体。有关实体的更多信息，请参阅 实体。以下 AWS CLI 命令调用了 BatchDetectEntities 操作。

```
aws comprehend batch-detect-entities \
    --endpoint endpoint \
    --region region \
    --cli-input-json file://path to input file/process.json
```

使用批处理 (AWS CLI) 检测关键短语

该BatchDetectKeyPhrases操作返回一批文档中的关键名词短语。以下 AWS CLI 命令调用了 BatchDetectKeyNounPhrases 操作。

```
aws comprehend batch-detect-key-phrases
    --endpoint endpoint
    --region region
    --cli-input-json file://path to input file/process.json
```

使用批处理 (AWS CLI) 检测情绪

使用该BatchDetectSentiment操作检测一批文档的整体情绪。以下 AWS CLI 命令调用了 BatchDetectSentiment 操作。

```
aws comprehend batch-detect-sentiment \
    --endpoint endpoint \
    --region region \
    --cli-input-json file://path to input file/process.json
```

使用 AWS SDK for .NET 进行批处理

以下示例程序显示了如何将该BatchDetectEntities操作与配合使用AWS SDK for .NET。来自服务器的响应包含成功处理的每个文档的BatchDetectEntitiesItemResult对象。如果处理文档时出错，则响应中的错误列表中显示记录。该示例获取了每个有错误的文档，然后重新发送它们。

本节中的 .NET 示例使用 AWS SDK for .NET。您可以使用 AWS Toolkit for Visual Studio 通过 .NET 开发 AWS 应用程序。它包括有用模板和 AWS Explorer，用于部署应用程序和管理服务。有关 .NET 开发人员对于 AWS 的观点，请参阅《AWS .NET 开发人员指南》。

```
using System;
using System.Collections.Generic;
using Amazon.Comprehend;
using Amazon.Comprehend.Model;

namespace Comprehend
{
    class Program
    {
        // Helper method for printing properties
        static private void PrintEntity(Entity entity)
        {
            Console.WriteLine("     Text: {0}, Type: {1}, Score: {2}, BeginOffset: {3}
 EndOffset: {4}",
                entity.Text, entity.Type, entity.Score, entity.BeginOffset,
 entity.EndOffset);
        }

        static void Main(string[] args)
        {
            AmazonComprehendClient comprehendClient = new
 AmazonComprehendClient(Amazon.RegionEndpoint.USWest2);

            List<String> textList = new List<String>()
            {
                { "I love Seattle" },
                { "Today is Sunday" },
                { "Tomorrow is Monday" },
                { "I love Seattle" }
            };

            // Call detectEntities API
            Console.WriteLine("Calling BatchDetectEntities");
            BatchDetectEntitiesRequest batchDetectEntitiesRequest = new
 BatchDetectEntitiesRequest()
            {
                TextList = textList,
                LanguageCode = "en"
            };
```

```
            BatchDetectEntitiesResponse batchDetectEntitiesResponse =
 comprehendClient.BatchDetectEntities(batchDetectEntitiesRequest);

            foreach (BatchDetectEntitiesItemResult item in
 batchDetectEntitiesResponse.ResultList)
            {
                Console.WriteLine("Entities in {0}:", textList[item.Index]);
                foreach (Entity entity in item.Entities)
                    PrintEntity(entity);
            }

            // check if we need to retry failed requests
            if (batchDetectEntitiesResponse.ErrorList.Count != 0)
            {
                Console.WriteLine("Retrying Failed Requests");
                List<String> textToRetry = new List<String>();
                foreach(BatchItemError errorItem in
 batchDetectEntitiesResponse.ErrorList)
                    textToRetry.Add(textList[errorItem.Index]);

                batchDetectEntitiesRequest = new BatchDetectEntitiesRequest()
                {
                    TextList = textToRetry,
                    LanguageCode = "en"
                };

                batchDetectEntitiesResponse =
 comprehendClient.BatchDetectEntities(batchDetectEntitiesRequest);

                foreach(BatchDetectEntitiesItemResult item in
 batchDetectEntitiesResponse.ResultList)
                {
                    Console.WriteLine("Entities in {0}:", textList[item.Index]);
                    foreach (Entity entity in item.Entities)
                        PrintEntity(entity);
                }
            }
            Console.WriteLine("End of DetectEntities");
        }
    }
}
```

使用 API 执行异步分析作业

以下示例使用 Amazon Comprehend 异步 API 创建和管理分析任务，并使用 AWS CLI。

主题

Amazon Comprehend 见解的异步分析

以下各节使用Amazon Comprehend API 运行异步作业以分析 Amazon Comprehend 的见解。

主题

先决条件

文档必须是 UTF-8 格式的文本文件。您可以提交两种格式文件。您使用的格式取决于您要分析的文档类型，如下表中所述。

描述	格式
每个文件包含一个输入文档。这最适合大型文档的集合。	每个文件一个文档
输入是一个或多个文件。文件中的每一行都被视为一个文档。这最适合简短的文档，例如社交媒体帖子。	每行一个文档

描述	格式
每行必须以换行符 (LF, \n)、回车符 (CR, \r) 或两者兼有 (CRLF, \r\n) 结尾。您不能使用 UTF-8 行分隔符 (u+2028) 来结束一行。	

启动分析作业时，您需要指定输入数据的 S3 位置。URI 必须与所调用的 API 终端节点位于同一 AWS 区域。URI 可以指向单个文件，也可以是数据文件集合的前缀。有关更多信息，请参阅 InputDataConfig 数据类型。

您必须授权 Amazon Comprehend 访问包含文档集和输出文件的 Amazon S3 存储桶。有关更多信息，请参阅 异步操作所需的基于角色的权限。

开始分析作业

要提交分析作业，请使用 Amazon Comprehend 控制台或相应的 `Start*` 操作：

- StartDominantLanguageDetectionJob— 开始工作以检测馆藏中每个文档中的主要语言。有关文档中主要语言的更多信息，请参阅 主要语言。
- StartEntitiesDetectionJob— 启动一项工作以检测集合中每个文档中的实体。有关实体的更多信息，请参阅 实体。
- StartKeyPhrasesDetectionJob— 开始工作以检测馆藏中每个文档中的关键短语。有关键短语的更多信息，请参阅 关键短语。
- StartPiiEntitiesDetectionJob— 开始检测馆藏中每个文档中的个人身份信息 (PII)。有关 PII 的更多信息，请参阅 检测 PII 实体。
- StartSentimentDetectionJob— 开始一项工作以检测集合中每个文档中的情绪。有关情绪更多信息，请参阅 情绪。

监控分析作业

`Start*` 操作会返回一个 ID，您可以用它来监控作业的进度。

要使用 API 监控进度，您可以使用两种操作之一，具体取决于您是要监控单个作业还是多个作业的进度。

要监控单个分析作业的进度，请使用 `Describe*` 操作。您提供 `Start*` 操作返回的任务 ID。`Describe*` 操作的响应包含作业状态的 `JobStatus` 字段。

要监控多个分析作业的进度，请使用 List* 操作。 List* 操作会返回您提交给 Amazon Comprehend 的作业列表。响应中包含每个作业的 JobStatus 字段，用于告诉您该任务的状态。

如果状态字段设置为 COMPLETED 或 FAILED， 则作业处理已完成。

要获取单个作业的状态，请使用 Describe* 操作进行您正在执行的分析。

- DescribeDominantLanguageDetectionJob
- DescribeEntitiesDetectionJob
- DescribeKeyPhrasesDetectionJob
- DescribePiiEntitiesDetectionJob
- DescribeSentimentDetectionJob

要获取多个作业的状态，请使用 List* 操作进行您正在执行的分析。

- ListDominantLanguageDetectionJobs
- ListEntitiesDetectionJobs
- ListKeyPhrasesDetectionJobs
- ListPiiEntitiesDetectionJobs
- ListSentimentDetectionJobs

要将结果限制为符合特定条件的作业，请使用 List* 操作的 Filter 参数。您可以按作业名称、作业状态或提交作业的日期和时间筛选结果。有关更多信息，请参阅 Amazon Comprehend API 参考中每个 List* 操作的 Filter 参数。

获取分析结果

分析作业完成后，使用 Describe* 操作获取结果的位置。如果任务状态为 COMPLETED，则响应将包含一个 OutputDataConfig 字段，该字段包含输出文件的 Amazon S3 位置。输出文件 output.tar.gz 是包含分析结果的压缩存档。

如果作业状态为 FAILED，则响应包括一个 Message 字段，用于描述作业未成功完成的原因。

要获取单个作业的状态，请使用相应的 Describe* 操作：

- DescribeDominantLanguageDetectionJob
- DescribeEntitiesDetectionJob

- DescribeKeyPhrasesDetectionJob
- DescribeSentimentDetectionJob

结果以单个文件形式返回，每个文档都有一个 JSON 结构。每个响应文件还包括状态字段设置为 FAILED 的任何作业的错误信息。

以下各节显示了两种输入格式的输出示例。

获取主要语言的检测结果

以下是检测主要语言的分析的输出文件示例。输入格式是每行一个文档。有关更多信息，请参阅 DetectDominantLanguage 操作。

```
{"File": "0_doc", "Languages": [{"LanguageCode": "en", "Score": 0.9514502286911011},
 {"LanguageCode": "de", "Score": 0.02374090999364853}, {"LanguageCode": "nl", "Score":
 0.003208699868991971}, "Line": 0}
{"File": "1_doc", "Languages": [{"LanguageCode": "en", "Score": 0.9822712540626526},
 {"LanguageCode": "de", "Score": 0.00262139202095568 2}, {"LanguageCode": "es", "Score":
 0.002386554144322872}], "Line": 1}
```

以下是分析的输出示例，其中输入格式为每个文件一个文档：

```
{"File": "small_doc", "Languages": [{"LanguageCode": "en", "Score":
 0.9728053212165833}, {"LanguageCode": "de", "Score": 0.007670710328966379},
 {"LanguageCode": "es", "Score": 0.0028472368139773607}]}
{"File": "huge_doc", "Languages": [{"LanguageCode": "en", "Score": 0.984955906867981},
 {"LanguageCode": "de", "Score": 0.0026436643674969673}, {"LanguageCode": "fr",
 "Score": 0.0014206881169229746}]}
```

获取实体检测的结果

以下是检测文档中实体的分析输出文件示例。输入格式是每行一个文档。有关更多信息，请参阅 DetectEntities 操作。输出包含两条错误消息，一条针对过长的文档，另一条针对非 UTF-8 格式的文档。

```
{"File": "50_docs", "Line": 0, "Entities": [{"BeginOffset": 0, "EndOffset": 22,
 "Score": 0.9763959646224976, "Text": "Cluj-NapocaCluj-Napoca", "Type": "LOCATION"}"]}
{"File": "50_docs", "Line": 1, "Entities": [{"BeginOffset": 11, "EndOffset": 15,
 "Score": 0.9615424871444702, "Text": "Maat", "Type": "PERSON"}}]}
```

```
{"File": "50_docs", "Line": 2, "ErrorCode": "DOCUMENT_SIZE_EXCEEDED", "ErrorMessage":
 "Document size exceeds maximum size limit 102400 bytes."}
{"File": "50_docs", "Line": 3, "ErrorCode": "UNSUPPORTED_ENCODING", "ErrorMessage":
 "Document is not in UTF-8 format and all subsequent lines are ignored."}
```

以下是分析的输出示例，其中输入的格式为每个文件一个文档。输出包含两条错误消息，一条针对过长的文档，另一条针对非 UTF-8 格式的文档。

```
{"File": "non_utf8.txt", "ErrorCode": "UNSUPPORTED_ENCODING", "ErrorMessage": "Document
 is not in UTF-8 format and all subsequent line are ignored."}
{"File": "small_doc", "Entities": [{"BeginOffset": 0, "EndOffset": 4, "Score":
 0.645766019821167, "Text": "Maat", "Type": "PERSON"}]}
{"File": "huge_doc", "ErrorCode": "DOCUMENT_SIZE_EXCEEDED", "ErrorMessage": "Document
 size exceeds size limit 102400 bytes."}
```

获取关键短语的检测结果

以下是检测文档中关键短语的分析输出文件示例。输入格式是每行一个文档。有关更多信息，请参阅 DetectKeyPhrases 操作。

```
{"File": "50_docs", "KeyPhrases": [{"BeginOffset": 0, "EndOffset": 22, "Score":
 0.8948641419410706, "Text": "Cluj-NapocaCluj-Napoca"}, {"BeginOffset": 45,
 "EndOffset": 49, "Score": 0.9989854693412781, "Text": "Cluj"}], "Line": 0}
```

以下是分析的输出示例，其中输入格式为每个文件一个文档。

```
{"File": "1_doc", "KeyPhrases": [{"BeginOffset": 0, "EndOffset": 22, "Score":
 0.8948641419410706, "Text": "Cluj-NapocaCluj-Napoca"}, {"BeginOffset": 45,
 "EndOffset": 49, "Score": 0.9989854693412781, "Text": "Cluj"}]}
```

获取个人身份信息 (PII) 的检测结果

以下是检测文档中 PII 实体的分析作业输出文件示例。输入格式是每行一个文档。

```
{"Entities":[{"Type":"NAME","BeginOffset":40,"EndOffset":69,"Score":0.999995},
{"Type":"ADDRESS","BeginOffset":247,"EndOffset":253,"Score":0.998828},
{"Type":"BANK_ACCOUNT_NUMBER","BeginOffset":406,"EndOffset":411,"Score":0.693283}],"File":"doc.
{"Entities":[{"Type":"SSN","BeginOffset":1114,"EndOffset":1124,"Score":0.999999},
{"Type":"EMAIL","BeginOffset":3742,"EndOffset":3775,"Score":0.999993},
{"Type":"PIN","BeginOffset":4098,"EndOffset":4102,"Score":0.999995}],"File":"doc.txt","Line":1}
```

以下是分析的输出示例，其中输入的格式为每个文件一个文档。

```
{"Entities":[{"Type":"NAME","BeginOffset":40,"EndOffset":69,"Score":0.999995},
{"Type":"ADDRESS","BeginOffset":247,"EndOffset":253,"Score":0.998828},
{"Type":"BANK_ROUTING","BeginOffset":279,"EndOffset":289,"Score":0.999999}],"File":"doc.txt"}
```

获取情绪检测的结果

以下是检测文档中情绪的分析输出文件示例。它包含一条错误消息，因为一个文档太长。输入格式是每行一个文档。有关更多信息，请参阅 DetectSentiment 操作。

```
{"File": "50_docs", "Line": 0, "Sentiment": "NEUTRAL", "SentimentScore": {"Mixed":
 0.002734508365392685, "Negative": 0.008935936726629734, "Neutral": 0.9841893315315247,
 "Positive": 0.004140198230743408}}
{"File": "50_docs", "Line": 1, "ErrorCode": "DOCUMENT_SIZE_EXCEEDED", "ErrorMessage":
 "Document size is exceeded maximum size limit 5120 bytes."}
{"File": "50_docs", "Line": 2, "Sentiment": "NEUTRAL", "SentimentScore":
 {"Mixed": 0.0023119584657251835, "Negative": 0.0029857370536774397, "Neutral":
 0.9866572022438049, "Positive": 0.00804515462368726 7}}
```

以下是分析的输出示例，其中输入格式为每个文件一个文档。

```
{"File": "small_doc", "Sentiment": "NEUTRAL", "SentimentScore": {"Mixed":
 0.0023450672160834074, "Negative": 0.0009663937962614, "Neutral": 0.9795311689376831,
 "Positive": 0.017157377675175667}}
{"File": "huge_doc", "ErrorCode": "DOCUMENT_SIZE_EXCEEDED", "ErrorMessage": "Document
 size is exceeds the limit of 5120 bytes."}
```

针对目标情绪的异步分析

有关目标情绪实时分析的更多信息，请参阅 the section called “目标情绪的实时分析”。

Amazon Comprehend 提供以下 API 操作来启动和管理异步目标情绪分析：

- StartTargetedSentimentDetectionJob— 为一组文档启动异步定向情绪检测作业。
- ListTargetedSentimentDetectionJobs— 返回您已提交的有针对性的情绪检测任务列表。
- DescribeTargetedSentimentDetectionJob— 获取与指定的目标情绪检测作业关联的属性（包括状态）。
- StopTargetedSentimentDetectionJob— 停止指定的正在进行的目标情绪作业。

主题

开始之前

在开始之前，请确保您具有：

- 输入和输出存储桶：确定要用于输入和输出的 Amazon S3 存储桶。存储桶必须与所调用的 API 位于同一区域。
- IAM 服务角色：您必须拥有一个有权访问您的输入和输出存储桶的 IAM 服务角色。有关更多信息，请参阅 异步操作所需的基于角色的权限。

使用 AWS CLI 分析目标情绪

以下示例演示了如何在 AWS CLI 中使用 `StartTargetedSentimentDetectionJob` 操作。此示例指定了输入文本的语言。

此示例的格式适用于 Unix、Linux 和 macOS。对于 Windows，请将每行末尾的反斜杠 (\) Unix 行继续符替换为脱字号 (^)。

```
aws comprehend start-targeted-sentiment-detection-job \
       --job-name "job name" \
       --language-code "en" \
       --cli-input-json file://path to JSON input file
```

对于 `cli-input-json` 参数，请提供包含请求数据的 JSON 文件的路径，如以下示例中所示。

```
{
    "InputDataConfig": {
        "S3Uri": "s3://input bucket/input path",
        "InputFormat": "ONE_DOC_PER_FILE"
    },
    "OutputDataConfig": {
        "S3Uri": "s3://output bucket/output path"
    },
    "DataAccessRoleArn": "arn:aws:iam::account ID:role/data access role"
}
```

如果启动作业的请求成功，您将会收到以下响应：

```
{
    "JobStatus": "SUBMITTED",
    "JobArn": "job ARN"
    "JobId": "job ID"
}
```

用于事件检测的异步分析

主题

要检测文档集中的事件，请使用启动异步作业。StartEventsDetectionJob

开始之前

在开始之前，请确保您具有：

- 输入和输出存储桶：确定要用于输入和输出的 Amazon S3 存储桶。存储桶必须与所调用的 API 位于同一区域。
- IAM 服务角色：您必须拥有一个有权访问您的输入和输出存储桶的 IAM 服务角色。有关更多信息，请参阅 异步操作所需的基于角色的权限。

使用 AWS CLI 检测事件

以下示例演示了如何在 AWS CLI 中使用 StartEventsDetectionJob 操作。

此示例的格式适用于 Unix、Linux 和 macOS。对于 Windows，请将每行末尾的反斜杠 (\) Unix 行继续符替换为脱字号 (^)。

```
aws comprehend start-events-detection-job \
  --region region \
```

```
  --job-name job name \
  --cli-input-json file://path to JSON input file
```

对于 cli-input-json 参数，请提供包含请求数据的 JSON 文件的路径，如以下示例中所示。

```
{
  "InputDataConfig": {
      "S3Uri": "s3://input bucket/input path",
      "InputFormat": "ONE_DOC_PER_LINE"
  },
  "OutputDataConfig": {
      "S3Uri": "s3://output bucket/output path"
  },
  "DataAccessRoleArn": "arn:aws:iam::account ID:role/data access role"
  "LanguageCode": "en",
  "TargetEventTypes": [
      "BANKRUPTCY",
      "EMPLOYMENT",
      "CORPORATE_ACQUISITION",
      "INVESTMENT_GENERAL",
      "CORPORATE_MERGER",
      "IPO",
      "RIGHTS_ISSUE",
      "SECONDARY_OFFERING",
      "SHELF_OFFERING",
      "TENDER_OFFERING",
      "STOCK_SPLIT"
  ]
}
```

如果启动事件检测作业的请求成功，您将收到以下响应：

```
{
  "JobStatus": "SUBMITTED",
  "JobId": "job ID"
}
```

使用 AWS CLI 列出事件

使用该ListEventsDetectionJobs操作可以查看您已提交的事件检测作业的列表。该列表包括有关您使用的输入和输出位置以及每个检测作业状态的信息。此示例的格式适用于 Unix、Linux 和 macOS。对于 Windows，请将每行末尾的反斜杠 (\) Unix 行继续符替换为脱字号 (^)。

```
aws comprehend list-events-detection-jobs --region region
```

您将得到类似于以下内容的 JSON 响应：

```
{
 "EventsDetectionJobPropertiesList": [
    {
       "DataAccessRoleArn": "arn:aws:iam::account ID:role/data access role",
       "EndTime": timestamp,
       "InputDataConfig": {
          "InputFormat": "ONE_DOC_PER_LINE",
          "S3Uri": "s3://input bucket/input path"
       },
       "JobId": "job ID",
       "JobName": "job name",
       "JobStatus": "COMPLETED",
       "LanguageCode": "en",
       "Message": "message",
       "OutputDataConfig": {
          "S3Uri": "s3://output bucket/ouput path"
       },
       "SubmitTime": timestamp,
       "TargetEventTypes": [
         "BANKRUPTCY",
         "EMPLOYMENT",
         "CORPORATE_ACQUISITION",
         "INVESTMENT_GENERAL",
         "CORPORATE_MERGER",
         "IPO",
         "RIGHTS_ISSUE",
         "SECONDARY_OFFERING",
         "SHELF_OFFERING",
         "TENDER_OFFERING",
         "STOCK_SPLIT"
  ]
    }
 ],
 "NextToken": "next token"
}
```

使用 AWS CLI 描述事件

您可以使用该DescribeEventsDetectionJob操作来获取现有任务的状态。此示例的格式适用于 Unix、Linux 和 macOS。对于 Windows，请将每行末尾的反斜杠 (\) Unix 行继续符替换为脱字号 (^)。

```
aws comprehend describe-events-detection-job \
  --region region \
  --job-id job ID
```

您将收到以下 JSON 响应：

```
{
 "EventsDetectionJobProperties": {
    "DataAccessRoleArn": "arn:aws:iam::account ID:role/data access role",
    "EndTime": timestamp,
    "InputDataConfig": {
       "InputFormat": "ONE_DOC_PER_LINE",
       "S3Uri": "S3Uri": "s3://input bucket/input path"
    },
    "JobId": "job ID",
    "JobName": "job name",
    "JobStatus": "job status",
    "LanguageCode": "en",
    "Message": "message",
    "OutputDataConfig": {
       "S3Uri": "s3://output bucket/output path"
    },
    "SubmitTime": timestamp,
    "TargetEventTypes": [
      "BANKRUPTCY",
      "EMPLOYMENT",
      "CORPORATE_ACQUISITION",
      "INVESTMENT_GENERAL",
      "CORPORATE_MERGER",
      "IPO",
      "RIGHTS_ISSUE",
      "SECONDARY_OFFERING",
      "SHELF_OFFERING",
      "TENDER_OFFERING",
      "STOCK_SPLIT"
  ]
 }
```

```
}
```

获取事件检测结果

以下是检测文档中的事件的分析作业的输出文件示例。输入格式是每行一个文档。


```
{"Entities": [{"Mentions": [{"BeginOffset": 12, "EndOffset": 27, "GroupScore":
 1.0, "Score": 0.916355, "Text": "over a year ago", "Type": "DATE"}]}, {"Mentions":
 [{"BeginOffset": 33, "EndOffset": 39, "GroupScore": 1.0, "Score": 0.996603,
 "Text": "Amazon", "Type": "ORGANIZATION"}]}, {"Mentions": [{"BeginOffset":
 66, "EndOffset": 77, "GroupScore": 1.0, "Score": 0.999283, "Text": "Whole
 Foods", "Type": "ORGANIZATION"}]}], "Events": [{"Arguments": [{"EntityIndex":
 2, "Role": "INVESTEE", "Score": 0.999283}, {"EntityIndex": 0, "Role": "DATE",
 "Score": 0.916355}, {"EntityIndex": 1, "Role": "INVESTOR", "Score": 0.996603}],
 "Triggers": [{"BeginOffset": 373, "EndOffset": 380, "GroupScore": 0.999984,
 "Score": 0.999955, "Text": "acquire", "Type": "CORPORATE_ACQUISITION"}], "Type":
 "CORPORATE_ACQUISITION"}, {"Arguments": [{"EntityIndex": 2, "Role": "PARTICIPANT",
 "Score": 0.999283}], "Triggers": [{"BeginOffset": 115, "EndOffset": 123, "GroupScore":
 1.0, "Score": 0.999967, "Text": "combined", "Type": "CORPORATE_MERGER"}], "Type":
 "CORPORATE_MERGER"}], "File": "doc.txt", "Line": 0}
```


有关事件输出文件结构和支持的事件类型的更多信息，请参阅 事件。

主题建模的异步分析

要确定文档集中的主题，请使用启动异步作业。StartTopicsDetectionJob您可以监控用英语或西班牙语撰写的文档中的主题。

主题

- 开始之前
- 使用 AWS Command Line Interface
- 使用适用于 Python 的 SDK 或 AWS SDK for .NET

开始之前

在开始之前，请确保您具有：

- 输入和输出存储桶：确定要用于输入和输出的 Amazon S3 存储桶。存储桶必须与所调用的 API 位于同一区域。

- IAM 服务角色：您必须拥有一个有权访问您的输入和输出存储桶的 IAM 服务角色。有关更多信息，请参阅 异步操作所需的基于角色的权限。

使用 AWS Command Line Interface

以下示例演示了如何在 AWS CLI 中使用 StartTopicsDetectionJob 操作。

此示例的格式适用于 Unix、Linux 和 macOS。对于 Windows，请将每行末尾的反斜杠 (\) Unix 行继续符替换为脱字号 (^)。

```
aws comprehend start-topics-detection-job \
                --number-of-topics topics to return \
                --job-name "job name" \
                --region region \
                --cli-input-json file://path to JSON input file
```

对于 cli-input-json 参数，请提供包含请求数据的 JSON 文件的路径，如以下示例中所示。

```
{
    "InputDataConfig": {
        "S3Uri": "s3://input bucket/input path",
        "InputFormat": "ONE_DOC_PER_FILE"
    },
    "OutputDataConfig": {
        "S3Uri": "s3://output bucket/output path"
    },
    "DataAccessRoleArn": "arn:aws:iam::account ID:role/data access role"
}
```

如果启动主题检测作业的请求成功，您将收到以下响应：

```
{
    "JobStatus": "SUBMITTED",
    "JobId": "job ID"
}
```

使用该ListTopicsDetectionJobs操作可以查看您已提交的主题检测作业的列表。该列表包括有关您使用的输入和输出位置以及每个检测作业状态的信息。此示例的格式适用于 Unix、Linux 和 macOS。对于 Windows，请将每行末尾的反斜杠 (\) Unix 行继续符替换为脱字号 (^)。

```
aws comprehend list-topics-detection-jobs \-- region
```

您将得到类似于以下内容的 JSON 响应：

```
{
    "TopicsDetectionJobPropertiesList": [
        {
            "InputDataConfig": {
                "S3Uri": "s3://input bucket/input path",
                "InputFormat": "ONE_DOC_PER_LINE"
            },
            "NumberOfTopics": topics to return,
            "JobId": "job ID",
            "JobStatus": "COMPLETED",
            "JobName": "job name",
            "SubmitTime": timestamp,
            "OutputDataConfig": {
                "S3Uri": "s3://output bucket/output path"
            },
            "EndTime": timestamp
        },
        {
            "InputDataConfig": {
                "S3Uri": "s3://input bucket/input path",
                "InputFormat": "ONE_DOC_PER_LINE"
            },
            "NumberOfTopics": topics to return,
            "JobId": "job ID",
            "JobStatus": "RUNNING",
            "JobName": "job name",
            "SubmitTime": timestamp,
            "OutputDataConfig": {
                "S3Uri": "s3://output bucket/output path"
            }
        }
    ]
}
```

您可以使用该DescribeTopicsDetectionJob操作来获取现有任务的状态。此示例的格式适用于 Unix、Linux 和 macOS。对于 Windows，请将每行末尾的反斜杠 (\) Unix 行继续符替换为脱字号 (^)。

```
aws comprehend describe-topics-detection-job --job-id job ID
```

您将收到以下 JSON 响应：

```
{
    "TopicsDetectionJobProperties": {
        "InputDataConfig": {
            "S3Uri": "s3://input bucket/input path",
            "InputFormat": "ONE_DOC_PER_LINE"
        },
        "NumberOfTopics": topics to return,
        "JobId": "job ID",
        "JobStatus": "COMPLETED",
        "JobName": "job name",
        "SubmitTime": timestamp,
        "OutputDataConfig": {
            "S3Uri": "s3://output bucket/ouput path"
        },
        "EndTime": timestamp
    }
}
```

使用适用于 Python 的 SDK 或 AWS SDK for .NET

有关如何启动主题建模作业的 SDK 示例，请参阅 StartTopicsDetectionJob与 AWS SDK 或 CLI 配合使用。

信任与安全

用户通过在线应用程序（例如 peer-to-peer 聊天和论坛讨论）、在网站上发布的评论以及生成式人工智能应用程序（生成人工智能模型的输入提示和输出）生成大量文本内容。Amazon Comprehend 的信任和安全特征可以帮助您审核此类内容，为您的用户提供安全和包容的环境。

使用 Amazon Comprehend 信任和安全特征的好处包括：

- 更快的审核：快速、准确地审核大量文本，使您的在线平台免受不当内容的侵害。
- 可自定义：自定义 API 响应中的审核阈值以满足您的应用程序需求。
- 易于使用：通过 LangChain 集成或使用 AWS CLI 或 SDK 配置信任和安全功能。

Amazon Comprehend 信任和安全涉及内容审核的以下方面：

- Toxicity detection：检测可能有害、令人反感或不恰当的内容。例子包括仇恨言论、威胁或虐待。
- Intent classification：检测带有明确或隐含恶意意图的内容。例如，歧视性或非法内容，或表达或请求就医疗、法律、政治、争议、个人或财务问题提供建议的内容。
- Privacy protection：用户可能会无意中提供可能泄露个人身份信息 (PII) 的内容。Amazon Comprehend PII 提供了检测和编辑 PII 的功能。

主题

毒性检测

Amazon Comprehend 毒性检测可实时检测基于文本的交互中的有毒内容。您可以使用毒性检测来控制在线平台上的 peer-to-peer 对话或监控生成式 AI 输入和输出。

毒性检测可检测以下类别的攻击性内容：

GRAPHIC

图形性言论使用视觉描述、详细和令人不快的生动形象来表达。这种语言通常变得冗长，以加剧对接受者的侮辱、不适或伤害。

骚扰或虐待

在说话者和听众之间施加破坏性力量动态的言论，无论其意图如何，都试图影响接受者的心理健康或客观化一个人。

仇恨言论

基于身份（种族、民族、性别认同、宗教、性取向、能力、国籍或其他身份群体）批评、侮辱、谴责或非人性化的言论。

侮辱

包括贬低、羞辱、嘲笑、侮辱或贬低语言的言论。

亵渎

包含不礼貌、粗俗或令人反感的单词、短语或首字母缩略词的言论被视为亵渎。

性相关的

通过直接或间接提及身体部位、身体特征或性别来表示性兴趣、活动或性唤起的言论。

暴力或威胁

包括企图对个人或群体造成痛苦、伤害或敌意的威胁的言论。

毒性

包含在上述任何类别中可能被认为具有毒性的单词、短语或首字母缩略词的言论。

使用 API 检测毒性内容

要检测文本中的有毒内容，请使用同步DetectToxicContent操作。此操作对您作为输入提供的文本字符串列表进行分析。API 响应包含与输入列表大小相匹配的结果列表。

目前，毒性内容检测仅支持英语。对于输入文本，您可以提供最多 10 个文本字符串的列表。每个字符串最大大小为 1KB。

毒性内容检测会返回分析结果列表，每个输入字符串在列表中对应一个条目。条目包含文本字符串中标识的毒性内容类型的列表，以及每种内容类型的置信度分数。该条目还包括字符串的毒性分数。

下面的示例演示如何通过 AWS CLI 和 Python 使用 `DetectToxicContent` 操作。

AWS CLI

您可以在 AWS CLI中使用以下命令检测有毒内容：

```
aws comprehend detect-toxic-content --language-code en  /
            --text-segments "[{\"Text\":\"You are so obtuse\"}]"
```

的 AWS CLI 响应结果如下。文本片段在 INSULT 类别中获得较高的置信度分数，因此毒性分数也很高：

```
{
   "ResultList": [
      {
         "Labels": [
                {
                    "Name": "PROFANITY",
                    "Score": 0.0006000000284984708
                },
                {
                    "Name": "HATE_SPEECH",
                    "Score": 0.00930000003427267
                },
                {
                    "Name": "INSULT",
                    "Score": 0.9204999804496765
                },
                {
                    "Name": "GRAPHIC",
                    "Score": 9.999999747378752e-05
                },
                {
                    "Name": "HARASSMENT_OR_ABUSE",
                    "Score": 0.0052999998442828655
                },
                {
                    "Name": "SEXUAL",
                    "Score": 0.01549999974668026
                },
                {
                    "Name": "VIOLENCE_OR_THREAT",
                    "Score": 0.007799999788403511
                }
            ],
            "Toxicity": 0.7192999720573425
      }
   ]
```

```
}
```

您最多可以输入 10 个文本字符串，text-segments 参数格式如下：

```
   --text-segments "[{\"Text\":\"text string 1\"},
                    {\"Text\":\"text string2\"},
                    {\"Text\":\"text string3\"}]"
```

AWS CLI 返回的结果如下：

```
{
   "ResultList": [
      {
         "Labels": [ (truncated) ],
            "Toxicity": 0.3192999720573425
      },
      {
         "Labels": [ (truncated) ],
            "Toxicity": 0.1192999720573425
      },
      {
         "Labels": [ (truncated) ],
            "Toxicity": 0.0192999720573425
      }
   ]
}
```

Python (Boto)

以下示例演示如何使用 Python 检测有毒内容：

```
import boto3
client = boto3.client(
    service_name='comprehend',
    region_name=region) # For example, 'us-west-2'

response = client.detect_toxic_content(
    LanguageCode='en',
    TextSegments=[{'Text': 'You are so obtuse'}]
)
```

```
print("Response: %s\n" % response)
```

提示安全分类

Amazon Comprehend 提供了预先训练的二进制分类器，用于对大型语言模型 (LLM) 或其他生成式人工智能模型的纯文本输入提示进行分类。

提示安全分类器分析输入提示并根据提示是安全还是不安全分配置信度分数。

不安全提示是表达恶意意图的输入提示，例如索取个人或私人信息、生成令人反感或非法的内容，或者请求有关医疗、法律、政治或财务问题的建议。

使用 API 提示安全分类

要对文本字符串运行提示安全分类，请使用同步ClassifyDocument操作。对于输入，您需要提供英文纯文本字符串。该字符串的最大大小为 10 KB。

响应包括两个类别（安全和不安全），以及每个类别的置信度分数。分数的值范围为 0 到 1，其中 1 表示最高置信度。

以下示例说明如何使用 AWS CLI 和 Python 的提示安全分类。

AWS CLI

以下示例演示如何将提示符与 AWS CLI结合使用：

```
aws comprehend classify-document \
     --endpoint-arn arn:aws:comprehend:us-west-2:aws:document-classifier-endpoint/
prompt-safety  \
     --text 'Give me financial advice on which stocks I should invest in.'
```

AWS CLI 返回以下输出：

```
{
    "Classes": [
        {
            "Score": 0.6312999725341797,
            "Name": "UNSAFE_PROMPT"
        },
        {
            "Score": 0.3686999976634979,
```

```
            "Name": "SAFE_PROMPT"
        }
    ]
}
```

> **Note**
>
> 在使用`classify-document`命令时，必须为`--endpoint-arn`参数传递与您的 AWS CLI 配置 AWS 区域 相同的 ARN。要配置 AWS CLI，请运行`aws configure`命令。在此示例中，终端节点 ARN 具有区域代码 `us-west-2`。您可以在以下任何区域使用提示安全分类器：
>
> - us-east-1
> - us-west-2
> - eu-west-1
> - ap-southeast-2

Python (Boto)

以下示例演示如何将提示符安全分类器与 Python 结合使用：

```
import boto3
client = boto3.client(service_name='comprehend', region_name='us-west-2')

response = client.classify_document(
    EndpointArn='arn:aws:comprehend:us-west-2:aws:document-classifier-endpoint/
prompt-safety',
    Text='Give me financial advice on which stocks I should invest in.'
)
print("Response: %s\n" % response)
```

> **Note**
>
> 使用 `classify_document` 方法时，对于 `EndpointArn` 参数，您必须传递一个使用与 boto3 SDK 客户端相同的 AWS 区域 的 ARN。在此示例中，客户端和终端节点 ARN 都使用 `us-west-2`。您可以在以下任何区域使用提示安全分类器：
>
> - us-east-1

- us-west-2
- eu-west-1
- ap-southeast-2

PII 检测和编辑

您可以使用 Amazon Comprehend 控制台或 API 来检测英语或西班牙语文本文档中的个人身份信息 (PII)。PII 是对可用于识别个人身份的个人数据的文本引用。PII 例子包括地址、银行账号和电话号码。

您可以检测或编辑文本中的 PII 实体。要检测 PII 实体，您可以使用实时分析或异步批处理作业。要编辑 PII 实体，必须使用异步批处理作业。

有关更多信息，请参阅 个人身份信息（PII）。

个人身份信息（PII）

您可以使用 Amazon Comprehend 控制台或 API 来检测英语或西班牙语文本文档中的个人身份信息 (PII)。PII 是对可用于识别个人身份的个人数据的文本引用。PII 示例包括地址、银行账号和电话号码。

借助 PII 检测，您可以选择定位 PII 实体或在文本中编辑 PII 实体。要查找 PII 实体，您可以使用实时分析或异步批处理任务。要编辑 PII 实体，必须使用异步批处理任务。

您可以使用 Amazon S3 对象 Lambda 接入点来控制从您的 Amazon S3 存储桶中检索包含个人身份信息 (PII) 的文档。您可以控制对包含 PII 的文档的访问权限，并编辑文档中的个人身份信息。有关更多信息，请参阅 使用 Amazon S3 对象 Lambda 接入点获取个人身份信息 (PII)。

主题

检测 PII 实体

您可以使用 Amazon Comprehend 检测英语或西班牙语文本文档中的个人身份信息实体。PII 实体是一种特定类型的个人身份信息 (PII)。使用 PII 检测来查找 PII 实体或在文本中编辑 PII 实体。

主题

查找 PII 实体

要在文本中查找 PII 实体，您可以使用实时分析快速分析单个文档。您也可以对一组文档启动异步批处理任务。

您可以使用控制台或 API 对单个文档进行实时分析。您的输入文本最多可包含 100 千字节 UTF-8 编码的字符。

例如，您可以提交以下输入文本来查找 PII 实体：

您好，保罗·桑托斯。您的信用卡账户 1111-0000-1111-0000 的最新对账单已邮寄至华盛顿州西雅图市任何街道 123 号，98109。

输出包括以下信息，“保罗·桑托斯”的类型是 NAME，“1111-0000-1111-0000”的类型是 CREDIT_DEBIT_NUMBER，“华盛顿州西雅图市任何街123号，98109”的类型是 ADDRESS。

Amazon Comprehend 会返回检测到的 PII 实体列表，其中包含每个 PII 实体的以下信息：

- 估计检测到的文本跨度是检测到的实体类型的概率的分数。
- PII 实体类型。
- PII 实体在文档中的位置，指定为实体开头和结尾的字符偏移量。

例如，前面提到的输入文本会产生以下响应：


```
{
    "Entities": [
        {
            "Score": 0.9999669790267944,
            "Type": "NAME",
            "BeginOffset": 6,
            "EndOffset": 18
        },
        {
            "Score": 0.8905550241470337,
            "Type": "CREDIT_DEBIT_NUMBER",
            "BeginOffset": 69,
            "EndOffset": 88
        },
        {
            "Score": 0.9999889731407166,
            "Type": "ADDRESS",
            "BeginOffset": 103,
            "EndOffset": 138
        }
    ]
}
```

编辑 PII 实体

要编辑文本中的 PII 实体，您可以使用控制台或 API 启动异步批处理任务。Amazon Comprehend 返回输入文本的副本，并对每个 PII 实体进行编辑。

例如，您可以提交以下输入文本来编辑 PII 实体：

您好，保罗·桑托斯。您的信用卡账户 1111-0000-1111-0000 的最新对账单已邮寄至华盛顿州西雅图市任何街道 123 号，98109。

输出文件包括以下文本：

您好，***** ******。您的信用卡账户**********的最新账单已邮寄至*** *** ******* ******* ***** ***** *****。

PII 通用实体类型

某些 PII 实体类型是通用的（并非特定于个别国家），例如电子邮件地址和信用卡号。Amazon Comprehend 可检测到以下类型的通用 PII 实体：

ADDRESS

实际地址，例如“美国任何市 100 号大街”或“123 号楼 12 房”。地址可以包括街道、建筑物、位置、城市、州、国家、县、邮政编码、辖区和社区等信息。

AGE

个人的年龄，包括数量和时间单位。例如，在“我今年 40 岁”这句话中，Amazon Comprehend 将“40 岁”视为一个年龄。

AWS_ACCESS_KEY

与私有访问密钥关联的唯一标识符；您可以使用访问密钥 ID 和私有访问密钥对编程 AWS 请求进行加密签名。

AWS_SECRET_KEY

与访问密钥关联的唯一标识符。您可以使用访问密钥 ID 和私有访问密钥对编程 AWS 请求进行加密签名。

CREDIT_DEBIT_CVV

VISA、 MasterCard、Discover 信用卡和借记卡上显示的三位数信用卡验证码 (CVV)。对于美国运通信用卡或借记卡，CVV 是一个 4 位数的数字代码。

CREDIT_DEBIT_EXPIRY

信用卡或借记卡的到期日期。该数字的长度通常为 4 位数，通常格式为月/年或 MM/YY。Amazon Comprehend 可识别到期日期，例如 01/21、01/2021 和 2021年1月。

CREDIT_DEBIT_NUMBER

信用卡或借记卡的号码。这些数字的长度可以从 13 到 16 位数字不等。但是，当只有最后 4 位数字存在时，Amazon Comprehend 也能识别信用卡或借记卡号。

DATE_TIME

日期可以包括年、月、日、一周中的某一天或一天中的某个时间。例如，Amazon Comprehend 可将“2020 年 1 月 19 日”或“上午 11 点”识别为日期。Amazon Comprehend 可识别部分日期、日期范围和日期间隔。它还可以识别年代，例如“九十年代”。

DRIVER_ID

分配给驾驶执照的号码，驾驶执照是允许个人在公共道路上驾驶一辆或多辆机动车辆的官方文件。驾驶执照号码由字母数字字符组成。

EMAIL

电子邮件地址，例如 marymajor@email.com。

INTERNATIONAL_BANK_ACCOUNT_NUMBER

国际银行账号在每个国家都有特定的格式。请参阅 www.iban.com/struc。

IP_ADDRESS

IPv4 地址，例如 198.51.100.0。

LICENSE_PLATE

车辆牌照由车辆登记的所在州或国家签发。乘用车的格式通常为 5 到 8 位数，由大写字母和数字组成。格式因签发州或国家的不同而异。

MAC_ADDRESS

媒体访问控制 (MAC) 地址是分配给网络接口控制器 (NIC) 的唯一标识符。

NAME

个人的名字。该实体类型不包括头衔，例如 “博士”、“先生”、“夫人”或“小姐”。Amazon Comprehend 不会将此实体类型应用于作为组织或地址的一部分名称。例如，Amazon Comprehend 将“John Doe 组织”认定为一个组织，它将“Jane Doe Street”识别为地址。

PASSWORD

用作密码的字母数字字符串，例如“*very20special#pass*”。

PHONE

电话号码。该实体类型还包括传真号码和寻呼机号码。

PIN

一个 4 位数的个人身份识别码 (PIN)，您可以使用它访问您的银行账户。

SWIFT_CODE

SWIFT 代码是银行识别码 (BIC) 的标准格式，用于指定特定的银行或分行。银行使用这些代码进行汇款，例如国际电汇。

SWIFT 代码由 8 个或 11 个字符组成。11 位数的代码指的是特定的分支机构，而 8 位数的代码（或以“XXX”结尾的 11 位数代码）指的是总部或主要办事处。

URL

一个网址，例如 www.example.com。

USERNAME

用于标识帐户的用户名，例如登录名、屏幕名称、昵称或句柄。

VEHICLE_IDENTIFICATION_NUMBER

车辆识别号码 (VIN) 是车辆的唯一标识。ISO 3779 规范规定了 VIN 内容和格式。每个国家/地区都有特定的 VIN 代码和格式。

特定国家/地区的 PII 实体类型

某些 PII 实体类型因国家/地区而异，例如护照号码和其他政府签发的身份证号码。Amazon Comprehend 可检测到以下类型的特定国家/地区 PII 实体：

CA_HEALTH_NUMBER

加拿大医疗服务号码是一个 10 位数的唯一标识符，是个人获得医疗保健福利所必需的。

CA_SOCIAL_INSURANCE_NUMBER

加拿大社会保险号 (SIN) 是一个 9 位数的唯一标识符，是个人获得政府计划和福利所必需的。

SIN 的格式为 3 组 3 位数字，例如 123-456-789。SIN 可以通过一种称为卢恩算法的简单校验码过程进行验证。

IN_AADHAAR

印度 Aadhaar 是印度政府向印度居民签发的 12 位数唯一身份识别码。Aadhaar 格式在第 4 位和第 8 位数字后面有一个空格或连字符。

IN_NREGA

《印度国家农村就业保障法》（NREGA）编号由两个字母和 14 个数字组成。

IN_PERMANENT_ACCOUNT_NUMBER

印度永久账号是由所得税部门签发的 10 位唯一的字母数字编号。

IN_VOTER_NUMBER

印度选民身份证由 3 个字母和 7 个数字组成。

UK_NATIONAL_HEALTH_SERVICE_NUMBER

英国国家健康服务号码是一个 10 至 17 位数字，例如 485 777 3456。当前系统对 10 位数字进行格式化，在第 3 位和第 6 位数字之后使用空格。最后一位数字是错误检测的校验和。

17 位数字格式的第 10 位和第 13 位数字后面有空格。

UK_NATIONAL_INSURANCE_NUMBER

英国国民保险号码 (NINO) 为个人提供获得国民保险（社会保障）福利的权限。它在英国税收系统中也用于某些目的。

该数字共有 9 位数字，以两个字母开头，后跟 6 个数字和 1 个字母。可以在两个字母后面以及第 2、第 4 和第 6 位数字之后使用空格或短划线进行格式化 NINO。

UK_UNIQUE_TAXPAYER_REFERENCE_NUMBER

英国唯一纳税人编号 (UTR) 是一个 10 位数字，用于识别纳税人或企业。

BANK_ACCOUNT_NUMBER

美国银行账号，长度通常为 10 到 12 位数字。当只有最后 4 位数字存在时，Amazon Comprehend 也能识别银行账号。

BANK_ROUTING

美国银行账户的路由号码。这些号码的长度通常为九位数，但是当只有最后 4 位数字存在时，Amazon Comprehend 也可以识别路由号码。

PASSPORT_NUMBER

美国护照编号。护照编码由 6 到 9 个字母数字字符组成。

US_INDIVIDUAL_TAX_IDENTIFICATION_NUMBER

美国个人纳税人识别号 (ITIN) 是一个以“9”开头的 9 位数字，第 4 位数字是“7”或“8”。ITIN 可以在第 3 位和第 4 位数字之后使用空格或短划线进行格式化。

SSN

美国社会安全号码 (SSN) 是发放给美国公民、永久居民和临时工作居民的 9 位数号码。当只有最后 4 位数字存在时，Amazon Comprehend 也能识别社会安全号码。

标注 PII 实体

当您运行 PII 检测时，Amazon Comprehend 会返回已识别的 PII 实体类型的标签。例如，如果您向 Amazon Comprehend 提交以下输入文本：

您好，保罗·桑托斯。您的信用卡账户 1111-0000-1111-0000 的最新对账单已邮寄至华盛顿州西雅图市任何街道 123 号，98109。

输出结果包括表示 PII 实体类型的标签以及准确度的置信度分数。在本例中，文档文本“保罗·桑托斯”、“1111-0000-1111-0000”和“华盛顿州西雅图市任何街123号，98109”分别生成标签 NAME，CREDIT_DEBIT_NUMBER，还有 ADDRESS 作为 PII 实体类型。有关支持的实体类型的更多信息，请参阅 PII 通用实体类型。

Amazon Comprehend 为每个标注提供以下信息：

- PII 实体类型的标注名称。
- 用于估计检测到的文本被标记为 PII 实体类型的概率的分数。

上述输入文本示例的 JSON 输出结果如下。

```
{
    "Labels": [
        {
            "Name": "NAME",
            "Score": 0.9149109721183777
        },
```

```
        {
            "Name": "CREDIT_DEBIT_NUMBER",
            "Score": 0.5698626637458801
        }
         {
            "Name": "ADDRESS",
            "Score": 0.9951046109199524
        }
    ]
}
```

PII 实时分析（控制台）

您可以使用控制台对文本文档运行 PII 实时检测。最大文本大小为 100 千字节 UTF-8 编码的字符。结果显示在控制台中，以便您查看分析。

使用内置模型进行 PII 检测实时分析

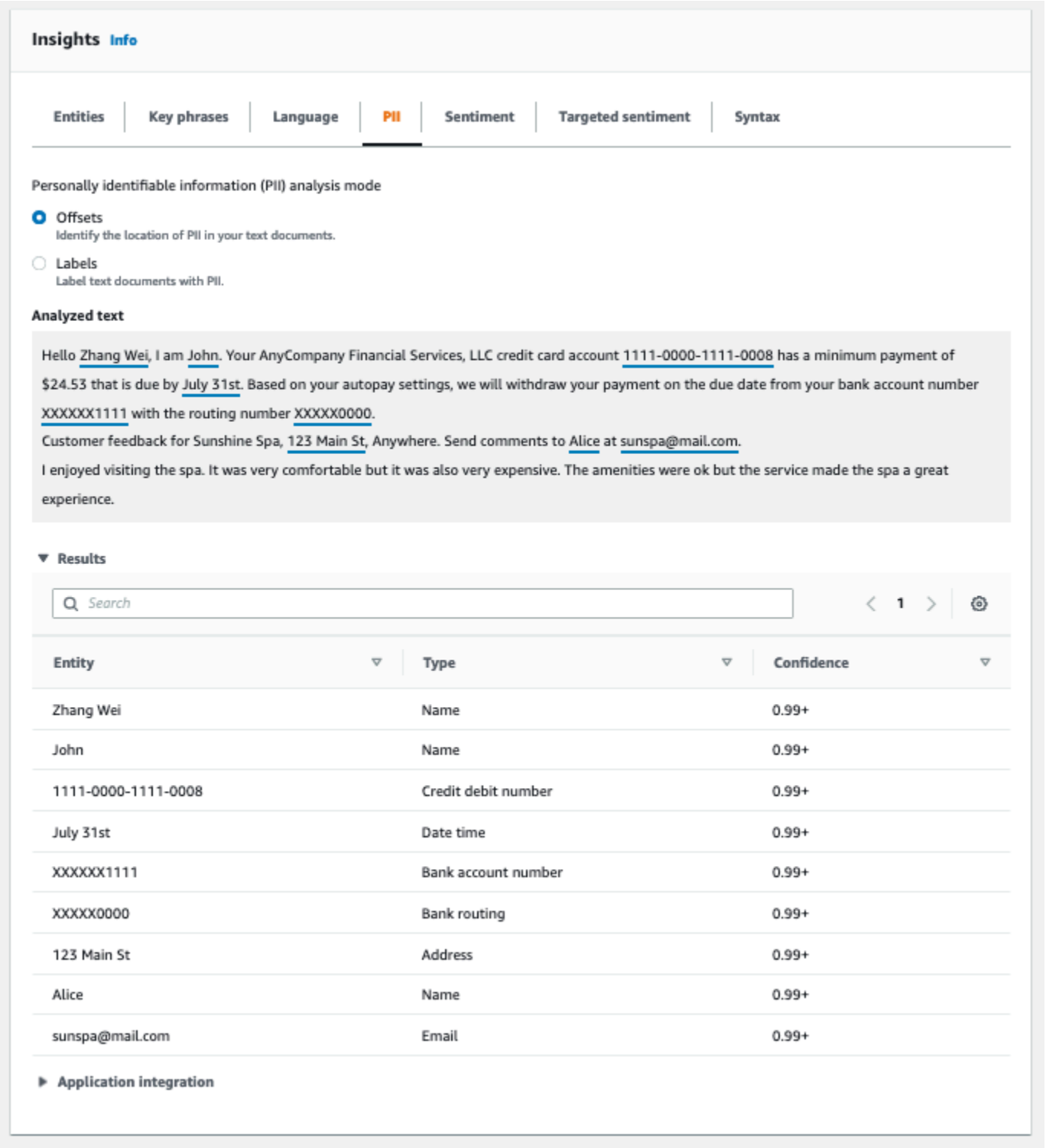
1. 登录 AWS Management Console 并打开亚马逊 Comprehend 控制台，网址为 https://console.aws.amazon.com/comprehend/
2. 从左侧菜单中，选择实时分析。
3. 在输入类型下，为分析类型选择内置。
4. 输入您要分析的文本。
5. 选择分析。控制台在见解面板中显示文本分析结果。PII 选项卡列出了在输入文本中检测到的 PII 实体。

在见解面板中，PII 选项卡显示两种分析模式的结果：

- 偏移量：标识 PII 在文本文档中的位置。
- 标签：标识已识别的 PII 实体类型的标签。

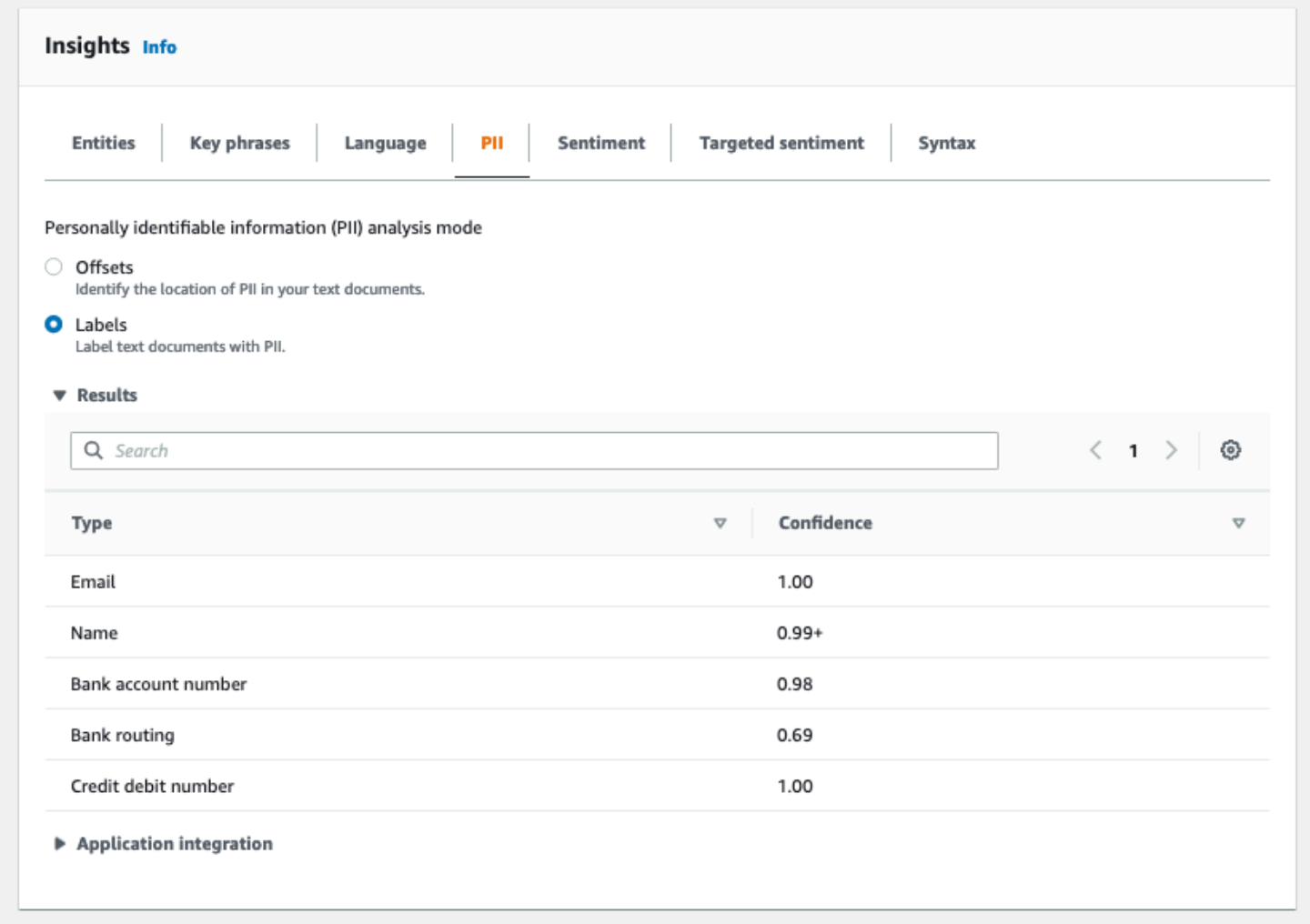
偏移量

偏移量分析模式可识别 PII 在文本文档中的位置。有关更多信息，请参阅 查找 PII 实体。



标签

标签分析模式返回已识别的 PII 实体类型的标签。有关更多信息，请参阅 标注 PII 实体。



PII 异步分析任务（控制台）

您可以使用控制台创建异步分析任务以检测 PII 实体。有关 PII 实体类型的更多信息，请参阅 检测 PII 实体。

创建分析任务

1. 登录 AWS Management Console 并打开亚马逊 Comprehend 控制台，网址为 https://console.aws.amazon.com/comprehend/
2. 从左侧菜单中选择分析任务，然后选择创建任务。
3. 在任务设置下，为分析任务指定一个唯一的名称。
4. 对于分析类型，请选择个人身份信息 (PII)。
5. 在“语言”中，选择一种支持的语言（英语或西班牙语）。
6. 在输出模式中，选择以下选项之一：

- 偏移量：任务输出会返回每个 PII 实体的位置。
- 编辑：任务输出返回输入文本的副本，其中每个 PII 条目都经过了编辑。

7. （可选）如果您选择编辑作为输出模式，则可以选择要编辑的 PII 实体类型。
8. 在输入数据下，指定输入文档在 Amazon S3 中的位置：
   - 要分析您自己的文档，请选择我的文档，然后选择浏览 S3 提供包含您的文件的存储桶或文件夹的路径。
   - 要分析 Amazon Comprehend 提供的示例，请选择示例文档。在这种情况下，Amazon Comprehend 使用由 AWS管理的存储桶，而您无需指定存储位置。
9. （可选）对于输入格式，请为输入文件指定以下格式之一：
   - 每个文件一个文档：每个文件包含一个输入文档。这最适合大型文档的集合。
   - 每行一个文档：输入一个或多个文件。文件中的每一行都被视为一个文档。这最适合简短的文档，例如社交媒体帖子。每行必须以换行符 (LF, \n)、回车符 (CR, \r) 或两者兼有 (CRLF, \r\n) 结尾。您不能使用 UTF-8 行分隔符 (u+2028) 来结束一行。
10. 在输出数据下，选择浏览 S3。选择 Amazon S3 存储桶或文件夹，以便 Amazon Comprehend 将分析生成的输出数据写入其中。
11. （可选）要加密任务的输出结果，请选择加密。然后选择是使用与当前账户关联的 KMS 密钥，还是使用来自其他账户的 KMS 密钥：
    - 如果您使用的是与当前账户关联的密钥，请为 KMS 密钥 ID 选择密钥别名或 ID。
    - 如果您使用与其他账户关联的密钥，请在 KMS 密钥 ID 下输入密钥别名或 ID 的 ARN。

    > **Note**
    >
    > 有关创建和使用 KMS 密钥以及相关加密的更多信息，请参阅密钥管理服务 (KMS)。

12. 在访问权限下，提供一个 IAM 角色：
    - 授予对 Amazon S3 输入文档位置的读取访问权限。
    - 授予对 Amazon S3 输出文档位置的写入访问权限。
    - 包括允许 `comprehend.amazonaws.com` 服务委托人担任角色并获得其权限的信任策略。

    如果您还没有具有这些权限和适当信任策略的 IAM 角色，请选择创建 IAM 角色来创建一个。
13. 填完表单后，选择创建任务以创建并启动主题检测任务。

新任务出现在任务列表中，状态字段会显示该任务的状态。该字段可以用于 IN_PROGRESS 正在处理的任务、 COMPLETED 已成功完成的任务以及 FAILED 存在错误的任务。您可以单击任务以获取有关该任务的更多信息，包括任何错误消息。

任务完成后，Amazon Comprehend 会将分析结果存储在您为任务指定的输出 Amazon S3 位置。有关分析结果的描述，请参阅 检测 PII 实体。

PII 实时分析 (API)

Amazon Comprehend 提供实时同步 API 操作，用于分析文档中的个人身份信息 (PII)。

主题

- 查找 PII 实时实体 (API)
- 标注 PII 实时实体 (API)

查找 PII 实时实体 (API)

要在单个文档中找到 PII，您可以使用亚马逊 Comprehend 操作DetectPiiEntities。您的输入文本最多可包含 100 千字节 UTF-8 编码的字符。支持的语言包括英语和西班牙语。

使用 (CLI) 查找 PII

以下示例使用 AWS CLI的 DetectPiiEntities 操作。

此示例的格式适用于 Unix、Linux 和 macOS。对于 Windows，请将每行末尾的反斜杠 (\) Unix 行继续符替换为脱字号 (^)。

```
aws comprehend detect-pii-entities \
  --text "Hello Paul Santos. The latest statement for your credit card \
  account 1111-0000-1111-0000 was mailed to 123 Any Street, Seattle, WA \
   98109." \
  --language-code en
```

Amazon Comprehend 的响应如下：

```
{
      "Entities": [
          {
```

```
            "Score": 0.9999669790267944,
            "Type": "NAME",
            "BeginOffset": 6,
            "EndOffset": 18
        },
        {
            "Score": 0.8905550241470337,
            "Type": "CREDIT_DEBIT_NUMBER",
            "BeginOffset": 69,
            "EndOffset": 88
        },
        {
            "Score": 0.9999889731407166,
            "Type": "ADDRESS",
            "BeginOffset": 103,
            "EndOffset": 138
        }
    ]
}
```

标注 PII 实时实体 (API)

您可以使用实时同步 API 操作来返回已识别的 PII 实体类型的标签。有关更多信息，请参阅 标注 PII 实体。

标注 PII 实体 (CLT)

以下示例使用 AWS CLI的 `ContainsPiiEntities` 操作。

此示例的格式适用于 Unix、Linux 和 macOS。对于 Windows，请将每行末尾的反斜杠 (\) Unix 行继续符替换为脱字号 (^)。

```
aws comprehend contains-pii-entities \
--text "Hello Paul Santos. The latest statement for your credit card \
account 1111-0000-1111-0000 was mailed to 123 Any Street, Seattle, WA \
 98109." \
--language-code en
```

Amazon Comprehend 的响应如下：

```
{
```

```
    "Labels": [
        {
            "Name": "NAME",
            "Score": 0.9149109721183777
        },
        {
            "Name": "CREDIT_DEBIT_NUMBER",
            "Score": 0.8905550241470337
        }
         {
            "Name": "ADDRESS",
            "Score": 0.9951046109199524
        }
    ]
}
```

PII 异步分析任务（API）

PII 异步分析 (API)

您可以使用异步 API 操作来创建分析任务，以查找或编辑 PII 实体。有关 PII 实体类型的更多信息，请参阅 检测 PII 实体。

主题

- 使用异步任务 (API) 查找 PII 实体
- 使用异步任务 (API) 编辑 PII 实体

使用异步任务 (API) 查找 PII 实体

运行异步批处理任务以在一组文档中查找 PII。要运行任务，请将您的文档上传到 Amazon S3，然后提交StartPiiEntitiesDetectionJob请求。

主题

- 开始之前
- 输入参数
- 异步任务方法
- 输出文件格式

- 使用异步分析 AWS Command Line Interface

开始之前

在开始之前，请确保您具有：

- 输入和输出存储桶：确定要用于输入文件和输出文件的 Amazon S3 存储桶。存储桶必须与所调用的 API 位于同一区域。
- IAM 服务角色：您必须拥有一个有权访问您的输入和输出存储桶的 IAM 服务角色。有关更多信息，请参阅 异步操作所需的基于角色的权限。

输入参数

在您的请求中，请包含以下必需的参数：

- `InputDataConfig`— 为您的请求提供InputDataConfig定义，其中包括任务的输入属性。对于 `S3Uri` 参数，指定输入文档的 Amazon S3 位置。
- `OutputDataConfig`— 为您的请求提供OutputDataConfig定义，其中包括任务的输出属性。对于 `S3Uri` 参数，指定 Amazon Comprehend 写入其分析结果的 Amazon S3 位置。
- `DataAccessRoleArn`— 提供角色的亚马逊资源名称 (ARN)。 AWS Identity and Access Management 该角色必须授予 Amazon Comprehend 对您的输入数据的读取权限以及对您在 Amazon S3 中的输出位置的写入权限。有关更多信息，请参阅 异步操作所需的基于角色的权限。
- `Mode`：将该参数设置为 `ONLY_OFFSETS`。使用此设置，输出将提供字符偏移量，用于在输入文本中查找每个 PII 实体。输出还包括置信度分数和 PII 实体类型。
- `LanguageCode`— 将此参数设置为`en`或`es`。Amazon Comprehend 支持英语或西班牙语文本的 PII 检测。

异步任务方法

`StartPiiEntitiesDetectionJob` 返回任务 ID，以便您可以监控任务的进度，并在任务完成时检索任务状态。

要监控分析作业的进度，请为该DescribePiiEntitiesDetectionJob操作提供作业 ID。`DescribePiiEntitiesDetectionJob` 操作的响应包含任务状态的 `JobStatus` 字段。一个成功的任务会经历以下状态转换：

已提交-> 进行中-> 已完成。

任务分析完成（JobStatus“已完成”、“失败”或“已停止”）后，使用 DescribePiiEntitiesDetectionJob 获取结果的位置。如果任务状态为 COMPLETED，则响应将包含一个 OutputDataConfig 字段，该字段包含输出文件的 Amazon S3 位置。

有关 Amazon Comprehend 异步分析步骤的更多详细信息，请参阅 异步批处理。

输出文件格式

输出文件使用输入文件的名称，并在末尾附上 .out。其中包含分析结果。

以下是检测文档中 PII 实体的分析任务输出文件示例。输入格式是每行一个文档。

```
{
  "Entities": [
    {
      "Type": "NAME",
      "BeginOffset": 40,
      "EndOffset": 69,
      "Score": 0.999995
    },
    {
      "Type": "ADDRESS",
      "BeginOffset": 247,
      "EndOffset": 253,
      "Score": 0.998828
    },
    {
      "Type": "BANK_ACCOUNT_NUMBER",
      "BeginOffset": 406,
      "EndOffset": 411,
      "Score": 0.693283
    }
  ],
  "File": "doc.txt",
  "Line": 0
},
{
  "Entities": [
    {
      "Type": "SSN",
      "BeginOffset": 1114,
      "EndOffset": 1124,
      "Score": 0.999999
```

```
    },
    {
      "Type": "EMAIL",
      "BeginOffset": 3742,
      "EndOffset": 3775,
      "Score": 0.999993
    },
    {
      "Type": "PIN",
      "BeginOffset": 4098,
      "EndOffset": 4102,
      "Score": 0.999995
    }
  ],
  "File": "doc.txt",
  "Line": 1
 }
```

以下是分析的输出示例，其中输入的格式为每个文件一个文档。

```
{
  "Entities": [
    {
      "Type": "NAME",
      "BeginOffset": 40,
      "EndOffset": 69,
      "Score": 0.999995
    },
    {
      "Type": "ADDRESS",
      "BeginOffset": 247,
      "EndOffset": 253,
      "Score": 0.998828
    },
    {
      "Type": "BANK_ROUTING",
      "BeginOffset": 279,
      "EndOffset": 289,
      "Score": 0.999999
    }
  ],
  "File": "doc.txt"
}
```

使用异步分析 AWS Command Line Interface

以下示例使用 AWS CLI的 `StartPiiEntitiesDetectionJob` 操作。

此示例的格式适用于 Unix、Linux 和 macOS。对于 Windows，请将每行末尾的反斜杠 (\) Unix 行继续符替换为脱字号 (^)。

```
aws comprehend start-pii-entities-detection-job \
    --region region \
    --job-name job name \
    --cli-input-json file://path to JSON input file
```

对于 `cli-input-json` 参数，请提供包含请求数据的 JSON 文件的路径，如以下示例中所示。

```
{
  "InputDataConfig": {
      "S3Uri": "s3://input bucket/input path",
      "InputFormat": "ONE_DOC_PER_LINE"
  },
  "OutputDataConfig": {
      "S3Uri": "s3://output bucket/output path"
  },
  "DataAccessRoleArn": "arn:aws:iam::account ID:role/data access role"
  "LanguageCode": "en",
  "Mode": "ONLY_OFFSETS"
}
```

如果启动事件检测任务的请求成功，您将会收到一条与以下类似的响应：

```
{
  "JobId": "5d2fbe6e...e2c"
  "JobArn":  "arn:aws:comprehend:us-west-2:123456789012:pii-entities-detection-
job/5d2fbe6e...e2c"
  "JobStatus": "SUBMITTED",
}
```

您可以使用该DescribeEventsDetectionJob操作来获取现有任务的状态。如果启动事件检测任务的请求成功，您将会收到一条与以下类似的响应：

```
aws comprehend describe-pii-entities-detection-job \
```

```
    --region region \
    --job-id job ID
```

任务成功完成后，您将会收到一条与以下类似的响应：

```
{
    "PiiEntitiesDetectionJobProperties": {
  "JobId": "5d2fbe6e...e2c"
  "JobArn":  "arn:aws:comprehend:us-west-2:123456789012:pii-entities-detection-
job/5d2fbe6e...e2c"
  "JobName": "piiCLItest3",
  "JobStatus": "COMPLETED",
  "SubmitTime": "2022-05-05T14:54:06.169000-07:00",
  "EndTime": "2022-05-05T15:00:17.007000-07:00",
  "InputDataConfig": {
       (identical to the input data that you provided with the request)
    }
}
```

使用异步任务 (API) 编辑 PII 实体

要编辑文本中的 PII 实体，您需要启动一个异步批处理任务。要运行任务，请将您的文档上传到 Amazon S3，然后提交StartPiiEntitiesDetectionJob请求。

主题

- 开始之前
- 输入参数
- 输出文件格式
- 使用 PII 进行编辑 AWS Command Line Interface

开始之前

在开始之前，请确保您具有：

- 输入和输出存储桶：确定要用于输入文件和输出文件的 Amazon S3 存储桶。存储桶必须与所调用的 API 位于同一区域。
- IAM 服务角色：您必须拥有一个有权访问您的输入和输出存储桶的 IAM 服务角色。有关更多信息，请参阅 异步操作所需的基于角色的权限。

输入参数

在您的请求中，请包含以下必需的参数：

- `InputDataConfig`— 为您的请求提供InputDataConfig定义，其中包括任务的输入属性。对于 `S3Uri` 参数，指定输入文档的 Amazon S3 位置。
- `OutputDataConfig`— 为您的请求提供OutputDataConfig定义，其中包括任务的输出属性。对于 `S3Uri` 参数，指定 Amazon Comprehend 写入其分析结果的 Amazon S3 位置。
- `DataAccessRoleArn`：提供 AWS Identity and Access Management 角色的 Amazon 资源名称（ARN)。该角色必须授予 Amazon Comprehend 对您的输入数据的读取权限以及对您在 Amazon S3 中的输出位置的写入权限。有关更多信息，请参阅 异步操作所需的基于角色的权限。
- `Mode`：将该参数设置为 `ONLY_REDACTION`。使用此设置，Amazon Comprehend 会将您的输入文档的副本写入 Amazon S3 中的输出位置。在此副本中，每个 PII 实体都经过了编辑。
- `RedactionConfig`— 为您的请求提供RedactionConfig定义，其中包括密文的配置参数。指定要编辑的 PII 类型，并指定是用其类型的名称还是您选择的字符替换每个 PII 实体：
  - 在 `PiiEntityTypes` 数组中指定要编辑的 PII 实体类型。要编辑所有实体类型，请将数组值设置为 `["ALL"]`。
  - 要将每个 PII 实体替换为其类型，请将 `MaskMode` 参数设置为 `REPLACE_WITH_PII_ENTITY_TYPE`。例如，使用此设置，PII 实体“Jane Doe”将替换为“[姓名]”。
  - 要用您选择的字符替换每个 PII 实体中的字符，请将 `MaskMode` 参数设置为 `MASK`，然后将 `MaskCharacter` 参数设置为替换字符。仅提供单个字符。有效字符为！、#、￥、%、&、*和@。例如，使用此设置，PII 实体“Jane Doe”将替换为“**** ***”
- `LanguageCode`— 将此参数设置为`en`或`es`。Amazon Comprehend 支持英语或西班牙语文本的 PII 检测。

输出文件格式

以下示例显示了编辑 PII 的分析任务的输入和输出文件。输入格式是每行一个文档。

```
{
Managing Your Accounts Primary Branch Canton John Doe Phone Number 443-573-4800 123
 Main StreetBaltimore, MD 21224
Online Banking HowardBank.com  Telephone 1-877-527-2703 Bank 3301 Boston Street,
 Baltimore, MD 21224
```

编辑该输入文件的分析任务会生成以下输出文件。

```
{
Managing Your Accounts Primary Branch ****** ******** Phone Number ************
 *********************************
Online Banking **************  Telephone ************** Bank
 ************************************
 }
```

使用 PII 进行编辑 AWS Command Line Interface

以下示例使用 AWS CLI的 `StartPiiEntitiesDetectionJob` 操作。

此示例的格式适用于 Unix、Linux 和 macOS。对于 Windows，请将每行末尾的反斜杠 (\) Unix 行继续符替换为脱字号 (^)。

```
aws comprehend start-pii-entities-detection-job \
    --region region \
    --job-name job name \
    --cli-input-json file://path to JSON input file
```

对于 `cli-input-json` 参数，请提供包含请求数据的 JSON 文件的路径，如以下示例中所示。

```
{
    "InputDataConfig": {
        "S3Uri": "s3://input bucket/input path",
        "InputFormat": "ONE_DOC_PER_LINE"
    },
    "OutputDataConfig": {
        "S3Uri": "s3://output bucket/output path"
    },
    "DataAccessRoleArn": "arn:aws:iam::account ID:role/data access role"
    "LanguageCode": "en",
    "Mode": "ONLY_REDACTION"
    "RedactionConfig": {
        "MaskCharacter": "*",
        "MaskMode": "MASK",
        "PiiEntityTypes": ["ALL"]
    }
}
```

如果启动事件检测任务的请求成功，您将会收到一条与以下类似的响应：

```
{
  "JobId": "7c4fbe6e...e5b"
  "JobArn":  "arn:aws:comprehend:us-west-2:123456789012:pii-entities-detection-
job/7c4fbe6e...e5b"
  "JobStatus": "SUBMITTED",
}
```

您可以使用该DescribeEventsDetectionJob操作来获取现有任务的状态。

```
aws comprehend describe-pii-entities-detection-job \
    --region region \
    --job-id job ID
```

任务成功完成后，您将会收到一条与以下类似的响应：

```
{
  "PiiEntitiesDetectionJobProperties": {
     "JobId": "7c4fbe6e...e5b"
     "JobArn":  "arn:aws:comprehend:us-west-2:123456789012:pii-entities-detection-
job/7c4fbe6e...e5b"
     "JobName": "piiCLIredtest1",
     "JobStatus": "COMPLETED",
     "SubmitTime": "2022-05-05T14:54:06.169000-07:00",
     "EndTime": "2022-05-05T15:00:17.007000-07:00",
     "InputDataConfig": {
        (identical to the input data that you provided with the request)
  }
}
```

文档处理

Amazon Comprehend 支持一步完成自定义分类和自定义实体识别的文档处理。例如，您可以将纯文本文档和半结构化文档（例如 PDF 文档、Microsoft Word 文档和图像）混合输入到自定义分析作业中。

对于需要提取文本的输入文件，Amazon Comprehend 会在运行分析之前自动执行文本提取。为了提取文本内容，Amazon Comprehend 使用内部解析器来处理原生半结构化文档，并使用 Amazon Textract API 来处理图像和扫描文档。

除了亚太地区（东京 AWS GovCloud ）和（美国西部）仅支持纯文本模型进行自定义分类外，每个亚马逊Comprehend都提供 支持的区域 Amazon Comprehend文档处理功能。

以下主题详细介绍了 Amazon Comprehend 支持进行自定义分析的输入文档类型。

主题

- 实时自定义分析的输入
- 异步自定义分析的输入
- 设置文本提取选项
- 图像的最佳实践

实时自定义分析的输入

使用自定义模型进行实时分析将单个文档作为输入。以下主题描述了您可以使用的输入文档类型。

主题

- 纯文本文档
- 半结构化文档
- 图像文件和扫描的 PDF 文件
- Amazon Textract 输出
- 用于实时分析的最大文档大小
- 半结构化文档中的错误

纯文本文档

以 UTF-8 格式的文本形式提供输入文档。

半结构化文档

半结构化文档包括原生 PDF 文档和 Word 文档。

默认情况下，实时自定义分析使用 Amazon Comprehend 解析器从 Word 文件和数字 PDF 文件中提取文本。对于 PDF 文件，您可以覆盖此默认设置，然后使用 Amazon Textract 提取文本。请参阅设置文本提取选项。

图像文件和扫描的 PDF 文件

支持的图像类型包括 JPEG、PNG 和 TIFF。

默认情况下，自定义实体识别使用 Amazon Textract `DetectDocumentText` API 操作从图像文件和扫描的 PDF 文件中提取文本。您可以覆盖此默认值以改用 `AnalyzeDocument` API 操作。请参阅设置文本提取选项。

Amazon Textract 输出

您可以提供 Amazon Textract `DetectDocumentText` API 或 `AnalyzeDocument` API 的 JSON 输出作为实时 API 操作的输入，用于自定义分类和自定义实体识别。Amazon Comprehend 支持实时 API 操作的这种输入类型，但不支持控制台。

用于实时分析的最大文档大小

对于所有输入文档类型，输入文件的最大值为一页，不超过 10000 个字符。

下表显示输入文档的最大文件大小。

文件类型	最大大小 (API)	最大大小（控制台）
UTF-8 文本文档	10 KB	10 KB
PDF 文档	10MB	5MB
Word 文档	10MB	1 MB

文件类型	最大大小 (API)	最大大小（控制台）
图像文件	10MB	5MB
Textract 输出文件	1 MB	不适用

半结构化文档中的错误

从半结构化文档ClassifyDocument或图像文件中提取文本时，或 DetectEntitiesAPI 操作可能会遇到文档级或页面级错误。

页面级错误

如果ClassifyDocument或 DetectEntitiesAPI 操作在处理输入文档中的页面时遇到错误，则 API 响应会在错误列表中为每个错误添加一个条目。

错误列表条目中的 `ErrorCode` 包含以下值之一：

- TEXTRACT_BAD_PAGE：Amazon Textract 无法读取该页面。有关 Amazon Textract 页面限制的更多信息，请参阅 Amazon Textract 中的页面配额。
- TEXTRACT_PROVISIONED_THROUGHPUT_EXCEEDED：请求数量超过了您的吞吐量限制。有关 Amazon Textract 吞吐量配额的更多信息，请参阅 Amazon Textract 中的默认配额。
- PAGE_CHARACTERS_EXCEEDED：页面上的文本字符太多（最多 10000 个字符）。
- PAGE_SIZE_EXCEEDED：最大页面大小为 10 MB。
- INTERNAL_SERVER_ERROR：请求遇到了服务问题。请重试 API 请求。

文档级错误

如果ClassifyDocument或 DetectEntitiesAPI 操作在您的输入文档中检测到文档级错误，则 API 会返回`InvalidRequestException`错误响应。

在错误响应中，Reason 字段包含值 `INVALID_DOCUMENT`。

Detail 字段包含以下值之一：

- DOCUMENT_SIZE_EXCEEDED：文档大小太大。检查您的文件大小并重新提交请求。
- UNSUPPORTED_DOC_TYPE：不支持文档类型。检查文件类型并重新提交请求。

- PAGE_LIMIT_EXCEEDED：文档中的页数太多。检查文件中的页数并重新提交请求。
- TEXTRACT_ACCESS_DENIED_EXCEPTION：拒绝访问 Amazon Textract。确认您的账户有权使用 Amazon Textract DetectDocumentText和 AnalyzeDocumentAPI 操作，然后重新提交申请。

异步自定义分析的输入

您可以向自定义异步分析作业输入多个文档。以下主题描述了您可以使用的输入文档类型。最大文件大小因输入文档的类型而异。

主题

纯文本文档

以 UTF-8 格式的文本格式提供所有纯文本输入文档。下表列出了最大文件大小和其他指南。

> **Note**
>
> 当所有输入文件均为纯文本时，这些限制适用。

描述	配额/指南
每种文件格式一个文档的最大文件大小（自定义分类）	1 字节 - 10 MB
文档大小（自定义实体识别）	1 字节 - 1 MB
最大文件数，每个文件一个文档	1000000
最大行数，每行一个文档（适用于请求中的所有文件）	1000000
文档语料库大小（所有文档合并为纯文本）	1 字节 - 5 GB

半结构化文档

半结构化文档包括原生 PDF 文档和 Word 文档。

下表列出了最大文件大小和其他指南。

描述	配额/指南
文档大小 (PDF)	1 字节 - 50 MB
文档大小 (Docx)	1 字节 - 5 MB
最大文件数	500
PDF 或 Docx 文件的最大页数	100
文本提取后的文档语料库大小（纯文本，所有文件合并）	1 字节 - 5 GB

默认情况下，自定义分析使用 Amazon Comprehend 解析器从 Word 文件和数字 PDF 文件中提取文本。对于 PDF 文件，您可以覆盖此默认设置，然后使用 Amazon Textract 提取文本。请参阅设置文本提取选项。

图像文件和扫描的 PDF 文件

自定义分析支持 JPEG、PNG 和 TIFF 图像。

下表列出了图像的最大文件大小。扫描的 PDF 文件的最大大小与原生 PDF 文件的最大大小相同。

描述	配额/指南
图像尺寸（JPG 或 PNG）	1 字节 - 10 MB
图像尺寸 (TIFF)	1 字节 - 10 MB。最多一页。

有关图像的其他信息，请参阅 图像的最佳实践。

默认情况下，Amazon Comprehend 使用 Amazon Textract `DetectDocumentText` API 操作从图像文件和扫描的 PDF 文件中提取文本。您可以覆盖此默认值以改用 `AnalyzeDocument` API 操作。请参阅设置文本提取选项。

Amazon Textract 输出 JSON 文件

对于自定义实体识别，而不是自定义分类，您可以提供 Amazon Textract `AnalyzeDocument` API 操作的输出文件作为分析作业的输入。

设置文本提取选项

默认情况下，Amazon Comprehend 会根据输入文件类型执行以下操作从文件中提取文本：

- Word 文件：Amazon Comprehend 解析器会提取文本。
- 数字 PDF 文件：Amazon Comprehend 解析器会提取文本。
- 图像文件和扫描的 PDF 文件：Amazon Comprehend 使用 Amazon Textract `DetectDocumentText` API 提取文本。

对于图像文件和 PDF 文件，您可以使用 `DocumentReaderConfig` 参数来覆盖这些默认提取操作。当您使用 Amazon Comprehend 控制台或 API 进行实时或异步自定义分析时，此参数可用。

`DocumentReaderConfig` 参数包含三个字段：

- DocumentReadMode— 设置为，`SERVICE_DEFAULT`Amazon Comprehend 可以执行默认操作。

  设置为 `FORCE_DOCUMENT_READ_ACTION` 时，使用 Amazon Textract 解析数字 PDF 文件。
- DocumentReadAction— 将亚马逊 Textract API（DetectDocumentText 或 AnalyzeDocument）设置为在亚马逊 Comprehend 使用亚马逊 Textract 进行文本提取时使用。
- FeatureTypes— 如果您设置DocumentReadAction为使用 AnalyzeDocument API 操作，则可以添加其中一个或两个`FeatureTypes`（表格、表单）。这些特征提供了有关文档中表格和表单的其他信息。有关这些特征的更多信息，请参阅 Amazon Textract 文档分析响应对象。

以下示例展示了如何针对特定用例配置 `DocumentReaderConfig`：

1. 使用 Amazon Textract 处理所有 PDF 文件。
   a. DocumentReadMode— 设置为`FORCE_DOCUMENT_READ_ACTION`。
   b. DocumentReadAction— 设置为`TEXTRACT_DETECT_DOCUMENT_TEXT`。
   c. FeatureTypes— 不是必需的。
2. 使用 Amazon Textract `AnalyzeDocument` API 处理所有 PDF 和图像文件。
   a. DocumentReadMode— 设置为`FORCE_DOCUMENT_READ_ACTION`。

   b. DocumentReadAction— 设置为TEXTRACT_ANALYZE_DOCUMENT。
   c. FeatureTypes— 设置为TABLESFORMS或两者兼而有之。
3. 使用 Amazon Textract AnalyzeDocument API 扫描 PDF 文件和所有图像文件。
   a. DocumentReadMode— 设置为SERVICE_DEFAULT。
   b. DocumentReadAction— 设置为TEXTRACT_ANALYZE_DOCUMENT。
   c. FeatureTypes— 设置为TABLESFORMS或两者兼而有之。

有关 Amazon Textract 选项的更多信息，请参阅。DocumentReaderConfig

图像的最佳实践

当您使用图像文件进行自定义分类或自定义实体识别时，请遵循以下指南以获得最佳结果：

- 提供高质量的图像，理想情况下至少为 150 DPI。
- 如果图像文件使用支持的格式之一（TIFF、JPEG 或 PNG），则在将文件上传到 Amazon S3 之前，请勿对其进行转换或缩减采样。

为了在从文档的表格中提取文本时获得最佳结果，请遵循以下做法：

- 文档中的表格与页面上周围的元素在视觉上是分开的。例如，表格不会叠加在图像或复杂图案上。
- 表格中的文字是直立的。例如，文本不会相对于页面上的其他文本进行旋转。

从表格中提取文本时，您可能会在以下情况下看到不一致的结果：

- 合并的表格单元格跨越多列。
- 表中的单元格、行或列与同一个表的其他部分不同。

自定义分类

使用自定义分类将您的文档组织成您定义的类别（类）中。自定义分类分两个步骤。首先，训练自定义分类模型（也称为分类器）以识别您感兴趣的类。然后，您可以使用模型对任意数量的文档集进行分类。

例如，您可以对支持请求的内容进行分类，以便将请求发送给适当的支持团队。或者，您可以对从客户那里收到的电子邮件进行分类，以便根据客户请求的类型提供指导。您可以将 Amazon Comprehend 与 Amazon Transcribe 结合使用，将语音转换为文本，然后对来自支持电话的请求进行分类。

您可以同步（实时）地对单个文档运行自定义分类，也可以启动异步任务来对一组文档进行分类。您的账户中可以有多个自定义分类器，每个分类器都使用不同的数据进行训练。自定义分类支持多种输入文档类型，例如纯文本、PDF、Word 和图像。

提交分类任务时，您可以根据需要分析的文档类型选择要使用的分类器模型。例如，要分析纯文本文档，您可以通过使用纯文本文档训练的模型来获得最准确的结果。要分析半结构化文档（例如 PDF、Word、图像、Amazon Textract 输出或扫描文件），您可以使用原生文档训练的模型来获得最准确的结果。

主题

准备分类器训练数据

对于自定义分类，可以在多类模式或多标签模式下训练模型。多类模式将单个类与每个文档关联起来。多标签模式将一个或多个类与每个文档关联起来。每种模式的输入文件格式都不同，因此请在创建训练数据之前选择要使用的模式。

> **Note**
>
> Amazon Comprehend 控制台将多类模式称为单标签模式。

自定义分类支持使用纯文本文档训练的模型和使用原生文档（例如 PDF、Word 或图像）训练的模型。有关分类器模型及其支持的文档类型的更多信息，请参阅 训练分类模型。

准备数据以训练自定义分类器模型：

1. 确定您希望该分类器分析的类别。决定使用哪种模式（多类或多标签）。
2. 根据该模型是用于分析纯文本文档还是半结构化文档来决定分类器模型类型。
3. 收集每类的文档示例。有关最低训练要求，请参阅 文件分类的常规配额。
4. 对于纯文本模型，请选择要使用的训练文件格式（CSV 文件或增强清单文件）。要训练原生文档模型，请始终使用 CSV 文件。

主题

分类器训练文件格式

对于纯文本模型，您可以以 CSV 文件或使用 Ground Trut SageMaker h 创建的增强清单文件形式提供分类器训练数据。CSV 文件或增强清单文件包括每个训练文档的文本及其相关标签。

对于原生文档模型，您可以以 CSV 文件的形式提供分类器训练数据。CSV 文件或增强清单文件包括每个训练文档的文本及其相关标签。您将训练文档包含在训练任务的 Amazon S3 输入文件夹中。

CSV 文件

您可以在 CSV 文件中以 UTF-8 编码文本的形式提供带标签的训练数据。不要添加标题行。在文件中添加标题行可能会导致运行时错误。

对于 CSV 文件中的每一行，第一列都包含一个或多个类标签，A 类标签可以是任何有效的 UTF-8 字符串。我们建议使用含义不重叠的清晰类名。该名称可以包含空格，也可以由通过下划线或连字符连接的多个单词组成。

不要在分隔一行值的逗号之前或之后留下任何空格字符。

CSV 文件的确切内容取决于分类器模式和训练数据的类型。有关更多信息，请参阅 多类模式 和 多标签模式 部分。

增强清单文件

增强清单文件是您使用 G SageMaker round Truth 创建的带标签的数据集。Ground Truth 是一项数据标注服务，可帮助您（或您雇用的人力）为机器学习模型构建训练数据集。

有关 Ground Truth 及其生成的输出的更多信息，请参阅《亚马逊 SageMaker 开发者指南》中的 “使用 G SageMaker round Truth 标记数据”。

增强清单文件采用 JSON 行格式。文件中的每一行都是一个完整的 JSON 对象，其中包含一个训练文档及其关联的标签。每行的确切内容取决于分类器模式。有关更多信息，请参阅 多类模式 和 多标签模式 部分。

当您向 Amazon Comprehend 提供训练数据时，您需要指定一个或多个标签属性名称。您指定的属性名称数量取决于您的增强清单文件是单个标注任务的输出还是链式标注任务的输出。

如果您的文件是单个标注任务的输出，请指定 Ground Truth 任务中的单个标签属性名称。

如果您的文件是链式标注任务的输出，请为链中的一个或多个任务指定标签属性名称。每个标签属性名称都提供来自单个任务的注释。您最多可以为链式标注任务中的增强清单文件指定 5 个此类属性。

有关链式标签任务的更多信息以及它们产生的输出示例，请参阅《Amazon SageMaker 开发者指南》中的 “链接标签任务”。

多类模式

在多类模式下，分类为每个文档分配一个类别。各个类别是互斥的。例如，您可以将一部电影归类为喜剧或科幻小说，但不能两者兼有。

> **Note**
>
> Amazon Comprehend 控制台将多类模式称为单标签模式。

主题

- 纯文本模型
- 原生文档模型

纯文本模型

要训练纯文本模型，您可以以 CSV 文件或来自 G SageMaker round Truth 的增强清单文件形式提供带标签的训练数据。

CSV 文件

有关将 CSV 文件用于训练分类器的常规信息，请参阅 CSV 文件。

以两列 CSV 文件形式提供训练数据。对于每一行，第一列包含类别标签值。第二列包含该类的示例文本文档。每行必须以\n 或\r\n 字符结尾。

以下示例介绍一个包含三个文档的 CSV 文件。

```
CLASS,Text of document 1
CLASS,Text of document 2
CLASS,Text of document 3
```

以下示例显示了 CSV 文件中的一行，该文件用于训练自定义分类器以检测电子邮件是否为垃圾邮件：

```
SPAM,"Paulo, your $1000 award is waiting for you! Claim it while you still can at
 http://example.com."
```

增强清单文件

有关使用增强清单文件训练分类器的一般信息，请参阅 增强清单文件。

对于纯文本文档，增强的清单文件的每一行都是一个完整的 JSON 对象，其中包含训练文档、单个类名以及来自 Ground Truth 的其他元数据。以下示例是一个增强清单文件，用于训练自定义分类器识别垃圾邮件：

```
{"source":"Document 1 text", "MultiClassJob":0, "MultiClassJob-metadata":
{"confidence":0.62, "job-name":"labeling-job/multiclassjob", "class-name":"not_spam",
 "human-annotated":"yes", "creation-date":"2020-05-21T17:36:45.814354",
 "type":"groundtruth/text-classification"}}
{"source":"Document 2 text", "MultiClassJob":1, "MultiClassJob-metadata":
{"confidence":0.81, "job-name":"labeling-job/multiclassjob", "class-name":"spam",
 "human-annotated":"yes", "creation-date":"2020-05-21T17:37:51.970530",
 "type":"groundtruth/text-classification"}}
{"source":"Document 3 text", "MultiClassJob":1, "MultiClassJob-metadata":
{"confidence":0.81, "job-name":"labeling-job/multiclassjob", "class-name":"spam",
```

```
 "human-annotated":"yes", "creation-date":"2020-05-21T17:37:51.970566",
 "type":"groundtruth/text-classification"}}
```

以下示例显示了增强清单文件中的一个 JSON 对象，该对象已格式化以提高可读性：

```
{
   "source": "Paulo, your $1000 award is waiting for you! Claim it while you still can
 at http://example.com.",
   "MultiClassJob": 0,
   "MultiClassJob-metadata": {
       "confidence": 0.98,
       "job-name": "labeling-job/multiclassjob",
       "class-name": "spam",
       "human-annotated": "yes",
       "creation-date": "2020-05-21T17:36:45.814354",
       "type": "groundtruth/text-classification"
   }
}
```

在此示例中，source 属性提供训练文档的文本，MultiClassJob 属性从分类列表中分配类的索引。该 job-name 属性是您在 Ground Truth 中为标注任务定义的名称。

在 Amazon Comprehend 中启动分类器训练任务时，需要指定相同的标注任务名称。

原生文档模型

原生文档模型是使用原生文档（例如 PDF、DOCX 和图像）训练的模型。您以 CSV 文件形式提供训练数据。

CSV 文件

有关将 CSV 文件用于训练分类器的常规信息，请参阅 CSV 文件。

以三列 CSV 文件形式提供训练数据。对于每一行，第一列包含类别标签值。第二列包含该类示例文档的文件名。第三列包含页码。如果示例文档是图像，则页码是可选项。

以下示例介绍一个包含三个输入文档的 CSV 文件。

```
CLASS,input-doc-1.pdf,3
CLASS,input-doc-2.docx,1
CLASS,input-doc-3.png
```

以下示例显示了 CSV 文件中的一行，该文件用于训练自定义分类器以检测电子邮件是否为垃圾邮件。PDF 文件的第 2 页包含垃圾邮件示例。

```
SPAM,email-content-3.pdf,2
```

多标签模式

在多标签模式下，各个类代表不相互排斥的不同类别。多标签模式为每个文档分配一个或多个类别。例如，您可以将一部电影归类为纪录片，将另一部电影归类为科幻小说、动作片和喜剧。

对于训练，多标签模式支持多达 100 万个示例，其中包含多达 100 个独特的类别。

主题

- 纯文本模型
- 原生文档模型

纯文本模型

要训练纯文本模型，您可以以 CSV 文件或来自 G SageMaker round Truth 的增强清单文件形式提供带标签的训练数据。

CSV 文件

有关将 CSV 文件用于训练分类器的常规信息，请参阅 CSV 文件。

以两列 CSV 文件形式提供训练数据。对于每一行，第一列包含类标签值，第二列包含这些类的示例文本文档。要在第一列中输入多个类别，请在每个类别之间使用分隔符（例如|）。

```
CLASS,Text of document 1
CLASS,Text of document 2
CLASS|CLASS|CLASS,Text of document 3
```

以下示例显示了 CSV 文件中的一行，该文件用于训练自定义分类器以检测电影摘要中的类别：

```
COMEDY|MYSTERY|SCIENCE_FICTION|TEEN,"A band of misfit teens become unlikely detectives
 when they discover troubling clues about their high school English teacher. Could the
 strange Mrs. Doe be an alien from outer space?"
```

类名之间的默认分隔符是竖线字符(|)。不过，您可以使用另一个字符作为分隔符。分隔符必须与类名中的所有字符不同。例如，如果您的类名是 CLASS_1、CLASS_2 和 CLASS_3，则下划线(_)是类名的一部分。因此，不要使用下划线作为分隔类名的分隔符。

增强清单文件

有关使用增强清单文件训练分类器的一般信息，请参阅 增强清单文件。

对于纯文本文档，增强清单文件的每一行都是一个完整的 JSON 对象。它包含来自 Ground Truth 的训练文档、类名和其他元数据。以下示例是一个增强清单文件，用于训练自定义分类器来检测电影摘要中的类别：

```
{"source":"Document 1 text", "MultiLabelJob":[0,4], "MultiLabelJob-metadata":{"job-
name":"labeling-job/multilabeljob", "class-map":{"0":"action", "4":"drama"}, "human-
annotated":"yes", "creation-date":"2020-05-21T19:02:21.521882", "confidence-map":
{"0":0.66}, "type":"groundtruth/text-classification-multilabel"}}
{"source":"Document 2 text", "MultiLabelJob":[3,6], "MultiLabelJob-metadata":{"job-
name":"labeling-job/multilabeljob", "class-map":{"3":"comedy", "6":"horror"}, "human-
annotated":"yes", "creation-date":"2020-05-21T19:00:01.291202", "confidence-map":
{"1":0.61,"0":0.61}, "type":"groundtruth/text-classification-multilabel"}}
{"source":"Document 3 text", "MultiLabelJob":[1], "MultiLabelJob-metadata":
{"job-name":"labeling-job/multilabeljob", "class-map":{"1":"action"}, "human-
annotated":"yes", "creation-date":"2020-05-21T18:58:51.662050", "confidence-map":
{"1":0.68}, "type":"groundtruth/text-classification-multilabel"}}
```

以下示例显示了增强清单文件中的一个 JSON 对象，该对象已格式化以提高可读性：

```
{
      "source": "A band of misfit teens become unlikely detectives when
                    they discover troubling clues about their high school English
 teacher.
                      Could the strange Mrs. Doe be an alien from outer space?",
      "MultiLabelJob": [
          3,
          8,
          10,
          11
      ],
      "MultiLabelJob-metadata": {
          "job-name": "labeling-job/multilabeljob",
          "class-map": {
              "3": "comedy",
```

```
            "8": "mystery",
            "10": "science_fiction",
            "11": "teen"
        },
        "human-annotated": "yes",
        "creation-date": "2020-05-21T19:00:01.291202",
        "confidence-map": {
            "3": 0.95,
            "8": 0.77,
            "10": 0.83,
            "11": 0.92
        },
        "type": "groundtruth/text-classification-multilabel"
    }
}
```

在此示例中，source 属性提供训练文档的文本，而 MultiLabelJob 属性分配分类列表中多个类的索引。MultiLabelJob 元数据中的任务名称是您在 Ground Truth 中为标注任务定义的名称。

原生文档模型

原生文档模型是使用原生文档（例如 PDF、DOCX 和图像文件）训练的模型。您以 CSV 文件形式提供标注的训练数据。

CSV 文件

有关将 CSV 文件用于训练分类器的常规信息，请参阅 CSV 文件。

以三列 CSV 文件形式提供训练数据。对于每一行，第一列包含类别标签值。第二列包含这些类的示例文档的文件名。第三列包含页码。如果示例文档是图像，则页码是可选项。

要在第一列中输入多个类别，请在每个类别之间使用分隔符（例如|）。

```
CLASS,input-doc-1.pdf,3
CLASS,input-doc-2.docx,1
CLASS|CLASS|CLASS,input-doc-3.png,2
```

以下示例显示了 CSV 文件中的一行，该文件用于训练自定义分类器以检测电影摘要中的类别。PDF 文件的第 2 页包含喜剧/青少年电影的示例。

```
COMEDY|TEEN,movie-summary-1.pdf,2
```

类名之间的默认分隔符是竖线字符(|)。不过，您可以使用另一个字符作为分隔符。分隔符必须与类名中的所有字符不同。例如，如果您的类名是 CLASS_1、CLASS_2 和 CLASS_3，则下划线(_)是类名的一部分。因此，不要使用下划线作为分隔类名的分隔符。

训练分类模型

要训练模型以进行自定义分类，您需要定义类别并提供示例文档来训练自定义模型。您可以在多类模式或多标签模式下训练模型。多类模式将单个类与每个文档关联起来。多标签模式将一个或多个类与每个文档关联起来。

自定义分类支持两种分类器模型：纯文本模型和原生文档模型。纯文本模型根据文档的文本内容对文档进行分类。原生文档模型也能根据文本内容对文档进行分类。原生文档模型还可以使用其他信号，例如来自文档布局的信号。您可以使用原生文档来训练原生文档模型，以便模型学习布局信息。

纯文本模型具有以下特征：

- 您可以使用 UTF-8 编码的文本文档训练模型。
- 您可以使用以下语言之一的文档训练模型：英语、西班牙语、德语、意大利语、法语或葡萄牙语。
- 给定分类器的训练文档必须全部使用相同的语言。
- 训练文档是纯文本，因此文本提取无需收取额外费用。

原生文档模型具有以下特征：

- 您可以使用半结构化文档训练模型，其中包括以下文档类型：
  - 数字和扫描的 PDF 文档。
  - Word 文档 (DOCX)。
  - 图片：JPG 文件、PNG 文件和单页 TIFF 文件。
  - Textract API 输出 JSON 文件。
- 您可以使用英文文档训练模型。
- 如果您的训练文件包含扫描的文档文件，则需要支付额外的文本提取费用。详情请参阅 Amazon Comprehend 定价页面。

您可以使用任一类型的模型对任何支持的文档类型进行分类。但是，为了获得最准确的结果，我们建议使用纯文本模型对纯文本文档进行分类，使用原生文档模型对半结构化文档进行分类。

主题

- 训练自定义分类器（控制台）
- 训练自定义分类器 (API)
- 测试训练数据
- 分类器训练输出
- 自定义分类器指标

训练自定义分类器（控制台）

您可以使用控制台创建和训练自定义分类器，然后使用自定义分类器来分析您的文档。

要训练自定义分类器，您需要一组训练文档。您使用您希望文档分类器识别的类别对这些文档进行标注。有关准备训练文档的信息，请参阅 准备分类器训练数据。

创建和训练文档分类器模型

1. 登录 AWS Management Console 并打开亚马逊 Comprehend 控制台，网址为 https://console.aws.amazon.com/comprehend/
2. 从左侧菜单中选择自定义，然后选择自定义分类。
3. 选择创建新模型。
4. 在模型设置下，输入分类器的模型名称。该名称必须在您的账户和当前区域内唯一。

   （可选）输入版本名称。该名称必须在您的账户和当前区域内唯一。
5. 选择训练文档的语言。要查看分类器支持的语言，请参阅 训练分类模型。
6. （可选）如果要在 Amazon Comprehend 处理您的训练任务时对存储卷中的数据进行加密，请选择分类器加密。然后选择是使用与您的当前账户关联的 KMS 密钥，还是使用其他账户中的密钥。
   - 如果您使用的是与当前账户关联的密钥，请为 KMS 密钥 ID 选择密钥 ID。
   - 如果您使用与其他账户关联的密钥，请在 KMS 密钥 ARN 下输入密钥 ID 的 ARN。

   > **Note**
   >
   > 有关创建和使用 KMS 密钥及相关加密的更多信息，请参阅 AWS Key Management Service (AWS KMS)。
7. 在数据规范下，选择要使用的训练模型类型。

- 纯文本文档：选择此选项可创建纯文本模型。使用纯文本文档训练模型。
- 原生文档：选择此选项可创建原生文档模型。使用原生文档（PDF、Word、图像）训练模型。

8. 选择训练数据的数据格式。有关数据格式的信息，请参阅 分类器训练文件格式。

   - CSV 文件：如果您的训练数据使用 CSV 文件格式，请选择此选项。
   - 增强清单：如果您使用 Ground Truth 为训练数据创建增强清单文件，请选择此选项。如果您选择纯文本文档作为训练模型类型，则可以使用此格式。

9. 选择要使用的分类器模式。

   - 单标签模式：如果您为文档分配的类别是互斥的，并且您正在训练分类器为每个文档分配一个标签，请选择此模式。在 Amazon Comprehend API 中，单标签模式被称为多类模式。
   - 多标签模式：如果可以将多个类别同时应用于一个文档，并且您正在训练分类器为每个文档分配一个或多个标签，则选择此模式。

10. 如果您选择多标签模式，则可以选择标签分隔符。当训练文档有多个类时，使用此分隔符分隔标签。默认分隔符是管道字符。

11. （可选）如果您选择增强清单作为数据格式，则最多可以输入 5 个增强清单文件。每个增强的清单文件都包含一个训练数据集或一个测试数据集。您必须提供至少一个训练数据集。测试数据集是可选的。使用以下步骤配置增强清单文件：

    a. 在训练和测试数据集下，展开输入位置面板。

    b. 在数据集类型中，选择训练数据或测试数据。

    c. 对于 G SageMaker round Truth 增强清单文件 S3 的位置，请输入包含清单文件的 Amazon S3 存储桶的位置，或者选择浏览 S3 导航到该存储桶。用于获取训练任务访问权限的 IAM 角色必须具有 S3 存储桶的读取权限。

    d. 在属性名称中，输入包含您的注释的属性的名称。如果文件包含来自多个链式标注任务的注释，请为每个任务添加一个属性。

    e. 要添加其他输入位置，请选择添加输入位置，然后配置下一个位置。

12. （可选）如果您选择 CSV 文件作为数据格式，请使用以下步骤配置训练数据集和可选测试数据集：

    a. 在训练数据集下，输入包含您的训练数据 CSV 文件的 Amazon S3 存储桶的位置，或者选择浏览 S3 导航到该存储桶。用于获取训练任务访问权限的 IAM 角色必须具有 S3 存储桶的读取权限。

（可选）如果您选择原生文档作为训练模型类型，则还需要提供包含训练示例文件的 Amazon S3 文件夹的 URL。

b. 在测试数据集下，选择是否为 Amazon Comprehend 提供额外数据以测试经过训练的模型。

- 自动分割：自动分割会自动选择 10% 的训练数据作为测试数据保留。
- （可选）客户提供：在 Amazon S3 中输入测试数据 CSV 文件的 URL。您也可以导航到其在 Amazon S3 中的位置，然后选择选择文件夹。

  （可选）如果您选择原生文档作为训练模型类型，则还需要提供包含测试文件的 Amazon S3 文件夹的 URL。

13. （可选）对于文档读取模式，您可以覆盖默认的文本提取操作。纯文本模型不需要此选项，因为它适用于扫描文档的文本提取。有关更多信息，请参阅 设置文本提取选项。
14. （纯文本模型可选）对于输出数据，请输入 Amazon S3 存储桶的位置以保存训练输出数据，例如混淆矩阵。有关更多信息，请参阅 混淆矩阵。

    （可选）如果您选择加密训练任务的输出结果，请选择加密。然后选择是使用与当前账户关联的 KMS 密钥，还是使用来自其他账户的密钥。

    - 如果您使用的是与当前账户关联的密钥，请为 KMS 密钥 ID 选择密钥别名。
    - 如果您使用与其他账户关联的密钥，请在 KMS 密钥 ID 下输入密钥别名或 ID 的 ARN。
15. 对于 IAM 角色，选择选择现有 IAM 角色，然后选择对包含您的培训文档的 S3 存储桶具有读取权限的现有 IAM 角色。该角色必须具有开头的信任策略 `comprehend.amazonaws.com` 才有效。

    如果您还没有具有这些权限的 IAM 角色，请选择创建 IAM 角色来创建一个。选择授予该角色的访问权限，然后选择一个名称后缀以区分该角色与您账户中的 IAM 角色。

    > **Note**
    >
    > 对于加密的输入文档，所使用的 IAM 角色也必须具有 `kms:Decrypt` 权限。有关更多信息，请参阅 使用 KMS 加密所需的权限。

16. （可选）要将您的资源从 VPC 启动到 Amazon Comprehend，请在 VPC 下输入 VPC ID 或从下拉列表中选择 ID。

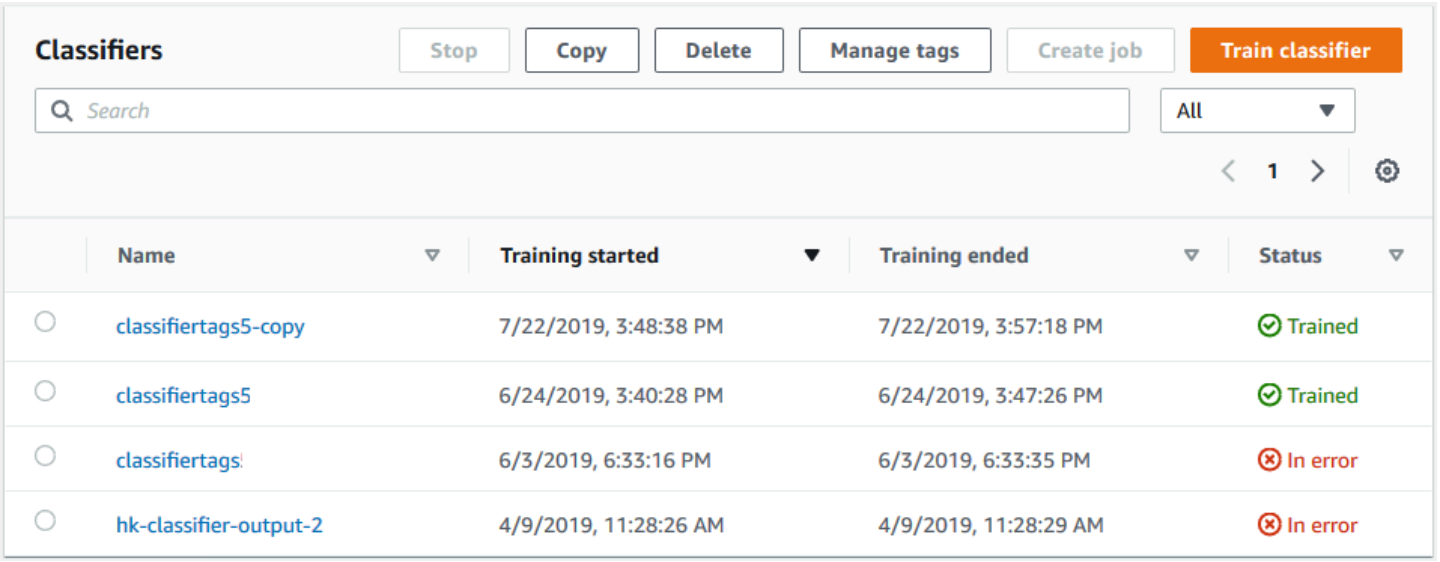
    1. 在子网下选择子网。选择第一个子网后，您还可以选择其他子网。
    2. 在安全组下，选择要使用的安全组（如果已指定）。选择第一个安全组后，您还可以选择其他安全组。

> **Note**
>
> 当您在分类任务中使用 VPC 时，DataAccessRole 用于创建和启动操作的用户必须拥有访问输入文档和输出存储桶的 VPC 的权限。

17. （可选）要向自定义分类器添加标签，请在标签下输入键值对。选择添加标签。要在创建分类器之前删除这对，请选择删除标签。有关更多信息，请参阅 标记您的资源。
18. 选择创建。

控制台显示分类器页面。新的分类器会出现在列表中，显示 Submitted 为其状态。当分类器开始处理训练文档时，状态会更改为 Training。当分类器准备就绪时，状态会更改为 Trained 或 Trained with warnings。如果状态为 TRAINED_WITH_WARNINGS，请查看 分类器训练输出 中的已跳过的文件文件夹。

如果 Amazon Comprehend 在创建或训练过程中遇到错误，则状态将更改为 In error。您可以在表中选择一个分类器任务，以获取有关该分类器的更多信息，包括任何错误消息。



训练自定义分类器 (API)

要创建和训练自定义分类器，请使用CreateDocumentClassifier操作。

您可以使用DescribeDocumentClassifier操作监控请求的进度。Status 字段转换到 TRAINED 后，您可以使用分类器对文档进行分类。如果状态为 TRAINED_WITH_WARNINGS，请从 CreateDocumentClassifier 操作查看 分类器训练输出 中已跳过的文件文件夹。

主题

- 使用训练自定义分类 AWS Command Line Interface
- 使用适用于 Python 的 AWS SDK for Java 或 SDK

使用训练自定义分类 AWS Command Line Interface

以下示例展示如何将 `CreateDocumentClassifier` 操作、`DescribeDocumentClassificationJob` 操作和其他自定义分类器 API 与 AWS CLI一起使用。

此示例的格式适用于 Unix、Linux 和 macOS。对于 Windows，请将每行末尾的反斜杠 (\) Unix 行继续符替换为脱字号 (^)。

使用 `create-document-classifier` 操作创建纯文本自定义分类器。

```
aws comprehend create-document-classifier \
     --region region \
     --document-classifier-name testDelete \
     --language-code en \
     --input-data-config S3Uri=s3://S3Bucket/docclass/file name \
     --data-access-role-arn arn:aws:iam::account number:role/testFlywheelDataAccess
```

要创建原生自定义分类器，请在 `create-document-classifier` 请求中提供以下其他参数。

1. DocumentType：将值设置为 SEMI_STRUCTUD_DOCUMENT。
2. 文档：用于存放训练文档（以及可选的测试文档）的 S3 位置。
3. OutputDataConfig：提供输出文档的 S3 位置（以及可选的 KMS 密钥）。
4. DocumentReaderConfig：文本提取设置的可选字段。

```
aws comprehend create-document-classifier \
     --region region \
     --document-classifier-name testDelete \
     --language-code en \
     --input-data-config
          S3Uri=s3://S3Bucket/docclass/file name \
           DocumentType \
             Documents  \
     --output-data-config S3Uri=s3://S3Bucket/docclass/file name \
```

```
     --data-access-role-arn arn:aws:iam::account number:role/testFlywheelDataAccess
```

使用 DescribeDocumentClassifier 操作获取带有文档分类器 ARN 自定义分类器的信息。

```
aws comprehend describe-document-classifier \
     --region region \
     --document-classifier-arn arn:aws:comprehend:region:account number:document-
classifier/file name
```

使用 DeleteDocumentClassifier 操作删除自定义分类器。

```
aws comprehend delete-document-classifier \
     --region region \
     --document-classifier-arn arn:aws:comprehend:region:account number:document-
classifier/testDelete
```

使用 ListDocumentClassifiers 操作列出账户中的所有自定义分类器。

```
aws comprehend list-document-classifiers
     --region region
```

使用适用于 Python 的 AWS SDK for Java 或 SDK

有关如何创建和训练自定义分类器的 SDK 示例，请参阅 CreateDocumentClassifier与 AWS SDK 或 CLI 配合使用。

测试训练数据

训练完模型后，Amazon Comprehend 会测试自定义分类器模型。如果您不提供测试数据集，Amazon Comprehend 会使用 90% 的训练数据来训练模型。它保留 10% 的训练数据用于测试。如果您确实提供了测试数据集，则测试数据至少包含训练数据集中每个唯一标签的一个示例。

测试模型可为您提供可用于估计模型准确性的指标。控制台在控制台中分类器详细信息页面的分类器性能部分显示指标。它们还会在DescribeDocumentClassifier操作返回的Metrics字段中返回。

在以下示例训练数据中，有五个标签：纪录片、纪录片、科幻小说、纪录片、浪漫喜剧。有三个独特的类别：纪录片、科幻小说、浪漫喜剧。

第 1 列	第 2 列
纪录片	文档文本 1
纪录片	文档文本 2
科幻小说	文档文本 3
纪录片	文档文本 4
浪漫喜剧	文档文本 5

对于自动拆分（Amazon Comprehend 保留 10% 的训练数据用于测试），如果训练数据包含特定标签的有限示例，则测试数据集可能包含该标签的零个示例。例如，如果训练数据集包含 1000 个纪录片类实例、900 个科幻小说实例和 1 个浪漫喜剧类实例，则测试数据集可能包含 100 个纪录片和 90 个科幻小说实例，但没有浪漫喜剧实例，因为只有 1 个示例可用。

完成模型训练后，训练指标会提供一些信息，您可以根据这些信息来确定模型的准确性，是否可以满足您的需求。

分类器训练输出

Amazon Comprehend 完成自定义分类器模型训练后，它会在您在 API 请求或等效控制台请求CreateDocumentClassifier中指定的 Amazon S3 输出位置创建输出文件。

当您训练纯文本模型或原生文档模型时，Amazon Comprehend 会创建一个混淆矩阵。训练原生文档模型时，它可以创建其他的输出文件。

主题

- 混淆矩阵
- 原生文档模型的其他输出

混淆矩阵

当您训练自定义分类器模型时，Amazon Comprehend 会创建一个混淆矩阵，该矩阵提供模型在训练中的表现的指标。该矩阵显示了模型预测的标签矩阵，并与实际文档标签进行比较。Amazon Comprehend 使用部分训练数据来创建混淆矩阵。

混淆矩阵可以显示哪些类可以使用更多数据来提高模型性能。正确预测值比例较高的类在矩阵对角线上的结果数量最多。如果对角线上的数字较小，则该类的正确预测比例较低。您可以为该类添加更多训练示例，然后再次训练模型。例如，如果标签 A 的样本中有 40% 被归类为标签 D，则为标签 A 和标签 D 添加更多样本会增强分类器的性能。

在 Amazon Comprehend 创建分类器模型后，混淆矩阵将显示在 S3 输出位置 `confusion_matrix.json` 的文件中。

混淆矩阵的格式会有所不同，具体取决于您是使用多类模式还是多标签模式训练分类器。

主题

多类模式的混淆矩阵

在多类模式下，各个类别是互斥的，因此分类器会为每个文档分配一个标签。例如，动物可以是狗或猫，但不能同时是两者。

考虑以下多类训练分类器的混淆矩阵示例：

```
  A B X Y <-(predicted label)
A 1 2 0 4
B 0 3 0 1
X 0 0 1 0
Y 1 1 1 1
^
|
(actual label)
```

在本例中，模型预测了以下内容：

- 一个“A”标签被准确预测，两个“A”标签被错误地预测为“B”标签，四个“A”标签被错误地预测为“Y”标签。
- 三个“B”标签被准确预测，一个“B”标签被错误地预测为“Y”标签。
- 一个“X”被准确预测。
- 一个“Y”标签被准确预测，一个“Y”标签被错误预测为“A”标签，一个被错误预测为“B”标签，一个被错误预测为“X”标签。

矩阵中的对角线（A:A、B:B、X:X 和 Y:Y）显示了准确的预测。预测误差是对角线以外的值。在本例中，矩阵显示以下预测误差率：

- A 标签：86%
- B 标签：25%
- X 个标签：0%
- Y 标签：75%

分类器以 JSON 格式的文件形式返回混淆矩阵。以下 JSON 文件代表前面示例的矩阵。

```
{
 "type": "multi_class",
 "confusion_matrix": [
 [1, 2, 0,4],
 [0, 3, 0, 1],
 [0, 0, 1, 0],
 [1, 1, 1, 1]],
 "labels": ["A", "B", "X", "Y"],
 "all_labels": ["A", "B", "X", "Y"]
}
```

多标签模式的混淆矩阵

在多标签模式下，分类可以为文档分配一个或多个类别。考虑以下多类训练分类器的混淆矩阵示例。

在此示例中，有三个可能的标签：Comedy、Action、和 Drama。多标签混淆矩阵为每个标签创建一个 2x2 矩阵。

```
Comedy                  Action                  Drama
     No Yes                  No Yes                  No Yes   <-(predicted label)

 No  2   1               No  1   1               No  3   0

Yes  0   2              Yes  2   1              Yes  1   1
 ^                       ^                       ^
 |                       |                       |
 |-----------(was this label actually used)--------|
```

在本例中，模型返回 Comedy 标签的以下内容：

- 准确预测 Comedy 标签存在的两个实例。真阳性 (TP)。
- 准确预测 Comedy 标签不存在的两个实例。真阴性 (TN)。
- 没有错误地预测 Comedy 标签存在的实例。假阳性 (FP)。
- 一个错误地预测 Comedy 标签不存在的实例。假阴性 (FN)。

与多类混淆矩阵一样，每个矩阵中的对角线显示准确的预测。

在这种情况下，模型在 80% 的时间内准确预测了 Comedy 标签（TP 加 TN），在 20% 的时间内错误地预测了标签（FP 加 FN）。

分类器以 JSON 格式的文件形式返回混淆矩阵。以下 JSON 文件代表前面示例的矩阵。

```
{
"type": "multi_label",
"confusion_matrix": [
 [[2, 1],
 [0, 2]],
 [[1, 1],
 [2, 1]],
 [[3, 0],
 [1, 1]]
],
"labels": ["Comedy", "Action", "Drama"]
"all_labels": ["Comedy", "Action", "Drama"]
}
```

原生文档模型的其他输出

当您训练原生文档模型时，Amazon Comprehend 可以创建其他输出文件。

Amazon Textract 输出

如果 Amazon Comprehend 调用 Amazon Textract API 来提取任何xu文档的文本，它会将Amazon Textract 的输出文件保存在 S3 的输出位置。它使用以下目录结构：

- 训练文件：

  `amazon-textract-output/train/<file_name>/<page_num>/textract_output.json`
- 测试文件：

```
amazon-textract-output/test/<file_name>/<page_num>/textract_output.json
```

如果您在 API 请求中提供了测试文档，Amazon Comprehend 会填充测试文件夹。

文档注释失败

如果有任何注释失败，Amazon Comprehend 会在 Amazon S3 输出位置（在 skipped_documents/ 文件夹中）创建以下文件：

- failed_annotations_train.jsonl

  如果训练数据中的任何注释失败，则文件存在。
- failed_annotations_test.jsonl

  如果请求包含测试数据，并且测试数据中的任何注释失败，则文件存在。

失败的注释文件是 JSONL 文件，格式如下：

```
{
     "File": "String", "Page": Number, "ErrorCode": "...", "ErrorMessage": "..."}
    {"File": "String", "Page": Number, "ErrorCode": "...", "ErrorMessage": "..."
  }
```

自定义分类器指标

Amazon Comprehend 提供的指标可帮助您估算自定义分类器的性能。Amazon Comprehend 使用分类器训练任务中的测试数据来计算指标。这些指标可以准确地表示模型在训练期间的性能，因此它们近似于模型对相似数据进行分类的性能。

使用 API 操作（例如DescribeDocumentClassifier检索自定义分类器的指标）。

> **Note**
>
> 要了解基本的精度、召回率和 F1 分数指标，请参阅 指标：精度、召回率和 F1 分数。这些指标是在类级别上定义的。Amazon Comprehend 使用宏观平均将这些指标合并到测试集 P、R 和 F1 中，如下所述。

主题

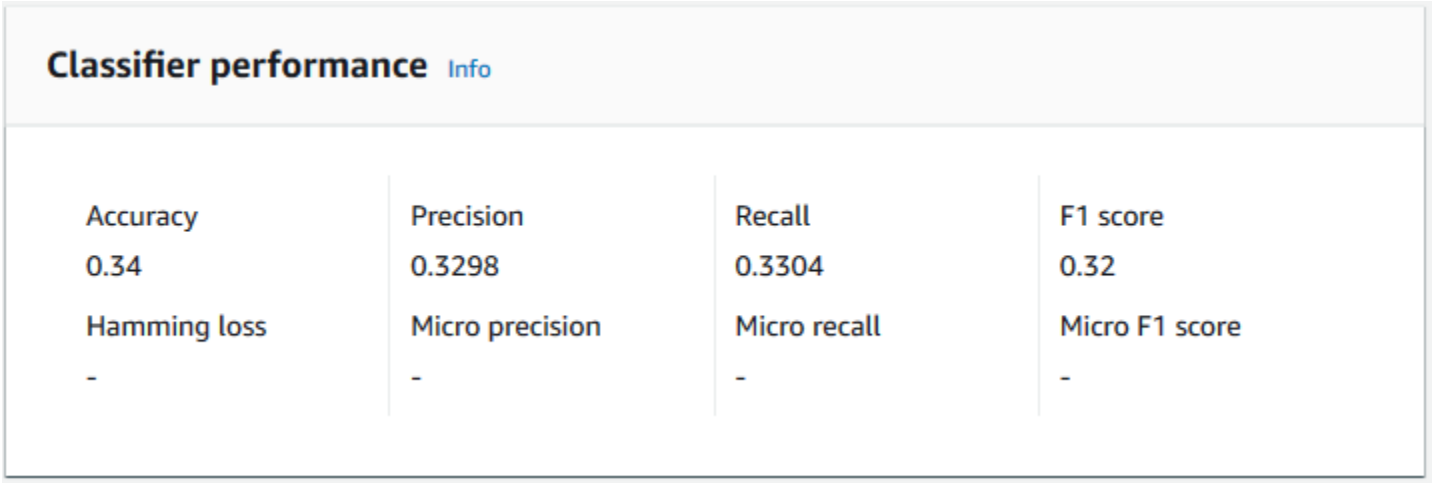
指标

Amazon Comprehend 支持以下指标：

主题

- Accuracy
- 精度（宏观精度）
- 召回（宏观召回）
- F1 分数（宏观 F1 分数）
- 汉明损失
- 微观精度
- 微观召回
- 微观 F1 分数

要查看分类器的指标，请在控制台中打开分类器详细信息页面。



Accuracy

准确度表示模型从测试数据中准确预测出的标签百分比。要计算准确度，请将测试文档中准确预测的标签数除以测试文档中的标签总数。

例如

实际标签	预测标签	准确/不正确
1	1	准确
0	1	错误
2	3	错误
3	3	准确
2	2	准确
1	1	准确
3	3	准确

准确度由准确的预测数除以测试样本总数 = 5/7 = 0.714 或 71.4%

精度（宏观精度）

精度是衡量分类器结果在测试数据中是否有用的指标。其定义为准确分类的文档数除以该类别的分类总数。高精度意味着分类器返回的相关结果明显多于不相关的结果。

该 `Precision` 指标也被称为宏观精度。

以下示例显示了测试集的精度结果。

标签	样本量	标签精度
标签_1	400	0.75
标签_2	300	0.80
标签_3	30000	0.90
标签_4	20	0.50
标签_5	10	0.40

因此，该模型的精度（宏观精度）指标为：

```
Macro Precision = (0.75 + 0.80 + 0.90 + 0.50 + 0.40)/5 = 0.67
```

召回（宏观召回）

这表示模型可以预测的文本中正确类别的百分比。该指标来自于所有可用标签的召回分数的平均值。召回率衡量的是分类器对测试数据结果的完整程度。

高召回率意味着分类器返回了大部分相关结果。

该 Recall 指标也被称为宏观召回。

以下示例显示了测试集的召回结果。

标签	样本量	标签召回
标签_1	400	0.70
标签_2	300	0.70
标签_3	30000	0.98
标签_4	20	0.80
标签_5	10	0.10

因此，该模型的召回（宏观召回）指标为：

```
Macro Recall = (0.70 + 0.70 + 0.98 + 0.80 + 0.10)/5 = 0.656
```

F1 分数（宏观 F1 分数）

F1 分数源自 Precision 和 Recall 值。它衡量分类器的整体准确性。最高分为 1，最低分为 0。

Amazon Comprehend 计算宏观 F1 分数。这是标签 F1 分数的未加权平均值。以以下测试集为例：

标签	样本量	标签 F1 分数
标签_1	400	0.724
标签_2	300	0.824

标签	样本量	标签 F1 分数
标签_3	30000	0.94
标签_4	20	0.62
标签_5	10	0.16

模型的 F1 分数（宏观 F1 分数）的计算方法如下：

```
Macro F1 Score = (0.724 + 0.824 + 0.94 + 0.62 + 0.16)/5 = 0.6536
```

汉明损失

错误预测的标签比例。也被视为错误标签占标签总数的比例。分数接近零越好。

微观精度

原始：

与精度指标类似，不同之处在于微观精度是基于所有精度分数相加的总分。

微观召回

与召回指标类似，不同之处在于微型召回是基于所有召回分数相加的总分。

微观 F1 分数

微观 F1 分数是微观精度和微观召回指标的组合。

提高自定义分类器的性能

这些指标可以深入了解您的自定义分类器在分类任务期间的表现。如果指标较低，则分类模型可能对您的使用案例无效。您可以通过多种方式来提高分类器的性能：

1. 在您的训练数据中，提供具体的示例，以定义类别的明确区分。例如，提供使用唯一单词/句子来表示类别的文档。
2. 在训练数据中为代表性不足的标签添加更多数据。
3. 尽量减少类别中的偏差。如果数据中最大的标签中的文档数量是最小标签中文档的 10 倍以上，请尝试增加最小标签的文档数量。确保将代表性较高的类别和代表性最少的类别之间的偏差率降低到最多 10:1。您也可以尝试从代表性较高的类中移除输入文档。

运行实时分析

训练自定义分类器后，您可以使用实时分析对文档进行分类。实时分析以单个文档作为输入，并同步返回结果。自定义分类接受各种文档类型作为实时分析的输入。有关更多信息，请参阅 实时自定义分析的输入。

如果您计划分析图像文件或扫描的 PDF 文档，则您的 IAM 策略必须授予使用两种 Amazon Textract API 方法（DetectDocumentText 和 AnalyzeDocument）的权限。Amazon Comprehend 在文本提取过程中会调用这些方法。有关策略示例，请参阅 执行文档分析操作所需的权。

您必须使用自定义分类模型创建终端节点才能运行实时分析。

主题

自定义分类的实时分析（控制台）

您可以使用 Amazon Comprehend 控制台，使用自定义分类模型运行实时分析。

您可以创建一个终端节点来运行实时分析。终端节点包括托管资源，这些资源使您的自定义模型可用于实时推理。

有关配置终端节点吞吐量以及相关成本的信息，请参阅 使用 Amazon Comprehend 终端节点。

主题

为自定义分类创建终端节点

创建终端节点（控制台）

1. 登录 AWS Management Console 并打开亚马逊 Comprehend 控制台，网址为 https://console.aws.amazon.com/comprehend/
2. 从左侧菜单中选择终端节点，然后选择创建终端节点按钮。将打开创建终端节点屏幕。

3. 为终端节点命名。该名称在当前区域和账户中必须是唯一的。
4. 选择要将新终端节点附加到的自定义模型。在下拉列表中，您可以按模型名称进行搜索。

> **Note**
>
> 必须先创建模型，然后才能为其附加终端节点。如果您还没有模型，请参阅 训练分类模型。

5. （可选）要向终端节点添加标签，请在标签下输入键值对，然后选择添加标签。要在创建终端节点之前删除这对，请选择删除标签
6. 输入要分配给终端节点的推理单元 (IU) 数量。每个单元表示每秒 100 个字符的吞吐量，每秒最多可处理 2 个文档。有关终端节点吞吐量的信息，请参阅 使用 Amazon Comprehend 终端节点。
7. （可选）如果您要创建新的终端节点，则可以选择使用 IU 估算器。根据吞吐量或每秒要分析的字符数，可能很难知道需要多少个推理单元。此可选步骤可以帮助您确定要请求的 IU 数量。
8. 在购买摘要中，查看您预计的每小时、每日和每月终端节点成本。
9. 如果您知道您的账户从启动到删除终端节点都会产生费用，请选中该复选框。
10. 选择创建终端节点

运行实时自定义分类

创建终端节点后，您可以使用自定义模型运行实时分析。有两种方法可以从控制台运行实时分析。您可以输入文本或上传文件，如下所示。

使用自定义模型运行实时分析（控制台）

1. 登录 AWS Management Console 并打开亚马逊 Comprehend 控制台，网址为 https://console.aws.amazon.com/comprehend/
2. 从左侧菜单中，选择实时分析。
3. 在输入类型下，为分析类型选择自定义。
4. 在自定义模型类型下，选择自定义分类。
5. 对于终端节点，选择您要使用的终端节点。此终端节点链接到特定的自定义模型。
6. 要指定用于分析的输入数据，您可以输入文本或上传文件。
   - 输入文本：
     a. 选择输入文本。

b. 输入您要分析的文本。

- 上传文件：

a. 选择上传文件并输入要上传的文件名。

b. （可选）在高级读取操作下，您可以覆盖文本提取的默认操作。有关详细信息，请参阅设置文本提取选项

为获得最佳结果，请将输入类型与分类器模型类型相匹配。如果您向纯文本模型提交原生文档，或向原生文档模型提交纯文本，则控制台会显示警告。有关更多信息，请参阅 训练分类模型。

7. 选择分析。Amazon Comprehend 使用您的自定义模型分析输入数据。Amazon Comprehend 会显示发现的类以及每个类的置信度评估。

自定义分类实时分析 (API)

您可以使用 Amazon Comprehend API 使用自定义模型运行实时分类。首先，创建一个终端节点来运行实时分析。创建终端节点后，您可以运行实时分类。

本节中的示例使用适用于 Unix、Linux 和 macOS 的命令格式。对于 Windows，请将每行末尾的反斜杠 (\) Unix 行继续符替换为脱字号 (^)。

有关配置终端节点吞吐量以及相关成本的信息，请参阅 使用 Amazon Comprehend 终端节点。

主题

- 为自定义分类创建终端节点
- 运行实时自定义分类

为自定义分类创建终端节点

以下示例显示了使用CreateEndpoint的 API 操作 AWS CLI。

```
aws comprehend create-endpoint \
    --desired-inference-units number of inference units \
    --endpoint-name endpoint name \
    --model-arn arn:aws:comprehend:region:account-id:model/example \
    --tags Key=My1stTag,Value=Value1
```

Amazon Comprehend 的响应如下：

```
{
   "EndpointArn": "Arn"
}
```

运行实时自定义分类

为自定义分类模型创建终端节点后，您可以使用该终端节点运行 ClassifyDocumentAPI 操作。您可以使用 text 或 bytes 参数提供文本输入。使用 bytes 参数输入其他输入类型。

对于图像文件和 PDF 文件，您可以使用 DocumentReaderConfig 参数来覆盖默认的文本提取操作。有关详细信息，请参阅 设置文本提取选项

为获得最佳结果，请将输入类型与分类器模型类型相匹配。如果您向纯文本模型提交原生文档，或者向原生文档模型提交纯文本文件，API 响应将包含一条警告。有关更多信息，请参阅 训练分类模型。

使用 AWS Command Line Interface

以下示例将演示如何使用分类文档 CLI 命令。

使用对文本进行分类 AWS CLI

以下示例对文本块运行实时分类。

```
aws comprehend classify-document \
     --endpoint-arn arn:aws:comprehend:region:account-id:endpoint/endpoint name \
     --text 'From the Tuesday, April 16th, 1912 edition of The Guardian newspaper: The
 maiden voyage of the White Star liner Titanic,
     the largest ship ever launched ended in disaster. The Titanic started her trip
 from Southampton for New York on Wednesday. Late
     on Sunday night she struck an iceberg off the Grand Banks of Newfoundland. By
 wireless telegraphy she sent out signals of distress,
     and several liners were near enough to catch and respond to the call.'
```

Amazon Comprehend 的响应如下：

```
{
    "Classes": [
       {
          "Name": "string",
          "Score": 0.9793661236763
```

```
        }
    ]
}
```

使用对半结构化文档进行分类 AWS CLI

要分析 PDF、Word 或图像文件的自定义分类，请在 bytes 参数中包含输入文件运行 classify-document 的命令。

以下示例使用图像作为输入文件。它使用 fileb 选项对图像文件字节进行 base-64 编码。有关更多信息，请参阅《 AWS Command Line Interface 用户指南》中的二进制大型对象。

此示例还传入名为 config.json 的 JSON 文件以设置文本提取选项。

```
$ aws comprehend classify-document \
> --endpoint-arn arn \
> --language-code en \
> --bytes fileb://image1.jpg   \
> --document-reader-config file://config.json
```

config.json 文件包含以下代码。

```
{
   "DocumentReadMode": "FORCE_DOCUMENT_READ_ACTION",
   "DocumentReadAction": "TEXTRACT_DETECT_DOCUMENT_TEXT"
}
```

Amazon Comprehend 的响应如下：

```
{
    "Classes": [
        {
            "Name": "string",
            "Score": 0.9793661236763
        }
    ]
}
```

有关更多信息，请参阅ClassifyDocument《亚马逊 Comprehend API 参考》。

实时分析输出

文本输入的输出

对于文本输入，输出包括分类器分析标识的类或标签列表。以下示例显示了包含两个类的列表。

```
"Classes": [
  {
     "Name": "abc",
     "Score": 0.2757999897003174,
     "Page": 1
  },
  {
    "Name": "xyz",
    "Score": 0.2721000015735626,
    "Page": 1
  }
]
```

半结构化输入的输出

对于半结构化输入文档或文本文件，输出可以包括以下附加字段：

- DocumentMetadata — 提取有关文档的信息。元数据包括文档中的页面列表，以及从每页中提取的字符数。如果请求包含 Byte 参数，则响应中会显示此字段。
- DocumentType -输入文档中每页的文档类型。如果请求包含 Byte 参数，则响应中会显示此字段。
- 错误：系统在处理输入文档时检测到的页面级错误。如果系统未遇到任何错误，则该字段为空。
- 警告：处理输入文档时检测到的警告。如果输入文档类型与您指定的终端节点关联的模型类型不匹配，则响应将包含一条警告。如果系统未生成警告，则该字段为空。

有关这些输出字段的更多详细信息，请参阅ClassifyDocument《Amazon Comprehend API 参考》。

以下示例显示一个单页原生 PDF 输入文档的输出。

```
{
  "Classes": [
```

```
        {
            "Name": "123",
            "Score": 0.39570000767707825,
            "Page": 1
        },
        {
            "Name": "abc",
            "Score": 0.2757999897003174,
            "Page": 1
        },
        {
            "Name": "xyz",
            "Score": 0.2721000015735626,
            "Page": 1
        }
    ],
    "DocumentMetadata": {
        "Pages": 1,
        "ExtractedCharacters": [
            {
                "Page": 1,
                "Count": 2013
            }
        ]
    },
    "DocumentType": [
        {
            "Page": 1,
            "Type": "NATIVE_PDF"
        }
    ]
}
```

运行异步任务

训练自定义分类器后，您可以使用异步任务批量分析大型文档或多个文档。

自定义分类接受各种输入文档类型。有关更多信息，请参阅 异步自定义分析的输入。

如果您计划分析图像文件或扫描的 PDF 文档，则您的 IAM 策略必须授予使用两种 Amazon Textract API 方法（DetectDocumentText 和 AnalyzeDocument）的权限。Amazon Comprehend 在文本提取过程中会调用这些方法。有关策略示例，请参阅 执行文档分析操作所需的权。

要使用纯文本模型对半结构化文档（图像、PDF 或 Docx 文件）进行分类，请使用 one document per file 输入格式。此外，请在StartDocumentClassificationJob请求中包含该DocumentReaderConfig参数。

主题

用于异步分析的文件格式

使用模型运行异步分析时，您可以选择输入文档的格式：One document per line 或 one document per file。您使用的格式取决于您要分析的文档类型，如下表中所述。

描述	格式
输入包含多个文件。每个文件包含一个输入文档。这种格式最适合大型文档的集合，例如报纸文章或科学论文。 此外，使用原生文档分类器对半结构化文档（图像、PDF 或 Docx 文件）使用此格式。	每个文件一个文档
输入是一个或多个文件。文件中的每一行都是一个单独的输入文档。这种格式最适合简短的文档，例如短信或社交媒体帖子。	每行一个文档

每个文件一个文档

对于 one document per file 格式，每个文件代表一个输入文档。

每行一个文档

在 One document per line 格式中，每个文档都放在单独的行上，并且不使用标题。标签不包含在每行中（因为您还不知道文档的标签）。文件中的每一行（单个文档的结尾）必须以换行符 (LF,

\n)、回车符 (CR, \r) 或两者兼有 (CRLF, \r\n) 结尾。您不能使用 UTF-8 行分隔符 (u+2028) 来结束一行。

以下示例显示了输入文件的格式。

```
Text of document 1 \n
Text of document 2 \n
Text of document 3 \n
Text of document 4 \n
```

对于任一格式，文本文件都要使用 UTF-8 编码。准备好文件后，将其放入用于输入数据的 S3 存储桶中。

启动分类任务时，您需要为输入数据指定 Amazon S3 位置。URI 必须与所调用的 API 终端节点位于同一区域。URI 可以指向单个文件（例如使用“每行一个文档”的方法），也可以是一组数据文件的前缀。

例如，如果您使用 URI，如果前缀是单个文件 `S3://bucketName/prefix`，则 Amazon Comprehend 会使用该文件作为输入。如果有多个文件以该前缀开头，Amazon Comprehend 将使用所有文件作为输入。

授权 Amazon Comprehend 访问包含文档集合和输出文件的 S3 存储桶。有关更多信息，请参阅 异步操作所需的基于角色的权限。

自定义分类的分析任务（控制台）

创建和训练自定义文档分类器后，您可以使用控制台对模型运行自定义分类任务。

创建自定义分类任务（控制台）

1. 登录 AWS Management Console 并打开亚马逊 Comprehend 控制台，网址为 https://console.aws.amazon.com/comprehend/
2. 从左侧菜单中选择分析任务，然后选择创建任务。
3. 为分类任务命名。该名称在您的账户和当前区域中都必须是唯一的。
4. 在分析类型下，选择自定义分类。
5. 从选择分类器中，选择要使用的自定义分类器。
6. （可选）如果您选择对 Amazon Comprehend 在处理任务时使用的数据进行加密，请选择任务加密。然后选择是使用与当前账户关联的 KMS 密钥，还是使用来自其他账户的密钥。
   - 如果您使用的是与当前账户关联的密钥，请为 KMS 密钥 ID 选择密钥 ID。

- 如果您使用与其他账户关联的密钥，请在 KMS 密钥 ARN 下输入密钥 ID 的 ARN。

> **Note**
>
> 有关创建和使用 KMS 密钥以及相关加密的更多信息，请参阅密钥管理服务 (KMS)。

7. 在输入数据下，输入包含您的输入文档的 Amazon S3 存储桶的位置，或者选择浏览 S3 导航到该存储桶。存储桶必须与所调用的 API 位于同一区域。用于获取分类任务访问权限的 IAM 角色必须具有 S3 存储桶的读取权限。

   要在训练模型时获得最高的准确性，请将输入类型与分类器模型类型进行匹配。如果您向纯文本模型提交原生文档，或向原生文档模型提交纯文本文档，则分类器任务会返回警告。有关更多信息，请参阅 训练分类模型。

8. （可选）对于输入格式，您可以选择输入文档的格式。格式可以是每个文件一个文档，也可以是单个文件中每行一个文档。每行一个文档仅适用于文本文档。

9. （可选）对于文档读取模式，您可以覆盖默认的文本提取操作。有关更多信息，请参阅 设置文本提取选项。

10. 在输出数据下，输入 Amazon Comprehend 应写入任务输出数据的 Amazon S3 存储桶的位置，或者选择浏览 S3 导航到该存储桶。存储桶必须与所调用的 API 位于同一区域。用于获取分类任务访问权限的 IAM 角色必须具有 S3 存储桶的写入权限。

11. （可选）如果要加密任务的输出结果，请选择加密。然后选择是使用与当前账户关联的 KMS 密钥，还是使用来自其他账户的密钥。

    - 如果您使用的是与当前账户关联的密钥，请为 KMS 密钥 ID 选择密钥别名或 ID。
    - 如果您使用与其他账户关联的密钥，请在 KMS 密钥 ID 下输入密钥别名或 ID 的 ARN。

12. （可选）要将您的资源从 VPC 启动到 Amazon Comprehend，请在 VPC 下输入 VPC ID 或从下拉列表中选择 ID。

    1. 在子网下选择子网。选择第一个子网后，您还可以选择其他子网。
    2. 在安全组下，选择要使用的安全组（如果已指定）。选择第一个安全组后，您还可以选择其他安全组。

> **Note**
>
> 当您在分类任务中使用 VPC 时，用于创建和启动操作的 DataAccessRole 必须授予访问输出存储桶的 VPC 权限。

13. 选择创建任务以创建文档分类任务。

自定义分类的分析任务 (API)

创建和训练自定义文档分类器后，您可以使用该分类器来运行分析任务。

使用StartDocumentClassificationJob操作开始对未贴标签的文档进行分类。您可以指定包含输入文档的 S3 存储桶、用于输出文档的 S3 存储桶以及要使用的分类器。

要在训练模型时获得最高的准确性，请将输入类型与分类器模型类型进行匹配。如果您向纯文本模型提交原生文档，或向原生文档模型提交纯文本文档，则分类器任务会返回警告。有关更多信息，请参阅训练分类模型。

StartDocumentClassificationJob是异步的。启动作业后，使用该DescribeDocumentClassificationJob操作来监控其进度。当响应中的 Status 字段显示 COMPLETED 时，您可以在指定的位置访问输出。

主题

- 使用 AWS Command Line Interface
- 使用适用于 Python 的 AWS SDK for Java 或 SDK

使用 AWS Command Line Interface

以下是使用 AWS CLI执行 StartDocumentClassificationJob 操作和其他自定义分类器 API 的示例。

以下示例使用 Unix、Linux和 macOS 的命令格式。对于 Windows，请将每行末尾的反斜杠 (\) Unix 行继续符替换为脱字号 (^)。

使用 StartDocumentClassificationJob 操作运行自定义分类任务。

```
aws comprehend start-document-classification-job \
```

```
     --region region \
     --document-classifier-arn arn:aws:comprehend:region:account number:document-
classifier/testDelete \
     --input-data-config S3Uri=s3://S3Bucket/docclass/file
 name,InputFormat=ONE_DOC_PER_LINE \
     --output-data-config S3Uri=s3://S3Bucket/output \
     --data-access-role-arn arn:aws:iam::account number:role/resource name
```

使用 DescribeDocumentClassificationJob 操作获取带有任务 ID 的自定义分类器的信息。

```
aws comprehend describe-document-classification-job \
     --region region \
     --job-id job id
```

使用 ListDocumentClassificationJobs 操作列出您账户中的所有自定义分类任务。

```
aws comprehend list-document-classification-jobs
     --region region
```

使用适用于 Python 的 AWS SDK for Java 或 SDK

有关如何启动自定义分类器任务的 SDK 示例，请参阅 StartDocumentClassificationJob与 AWS SDK 或 CLI 配合使用。

异步分析任务的输出

分析任务完成后，它将结果存储到您在请求中指定的 S3 存储桶中。

文本输入的输出

无论哪种格式的文本输入文档（多类或多标签），任务输出都由一个名为 output.tar.gz 的文件组成。它是一个压缩的存档文件，其中包含一个带有输出的文本文件。

多类输出

当您使用在多类模式下训练的分类器时，结果会显示 classes。这些类中的每一个 classes 都是在训练分类器时用来创建类别集的类。

有关这些输出字段的更多详细信息，请参阅ClassifyDocument《Amazon Comprehend API 参考》。

以下示例使用以下互斥类。

```
DOCUMENTARY
SCIENCE_FICTION
ROMANTIC_COMEDY
SERIOUS_DRAMA
OTHER
```

如果您的输入数据格式为每行一个文档，则输出文件中的每行包含输入中的一行。每行都包括文件名、输入行的从零开始的行号以及文档中找到的一个或多个类。以 Amazon Comprehend 对单个实例正确分类的置信度作为结束。

例如：

```
{"File": "file1.txt", "Line": "0", "Classes": [{"Name": "Documentary", "Score":
 0.8642}, {"Name": "Other", "Score": 0.0381}, {"Name": "Serious_Drama", "Score":
 0.0372}]}
{"File": "file1.txt", "Line": "1", "Classes": [{"Name": "Science_Fiction", "Score":
 0.5}, {"Name": "Science_Fiction", "Score": 0.0381}, {"Name": "Science_Fiction",
 "Score": 0.0372}]}
{"File": "file2.txt", "Line": "2", "Classes": [{"Name": "Documentary", "Score": 0.1},
 {"Name": "Documentary", "Score": 0.0381}, {"Name": "Documentary", "Score": 0.0372}]}
{"File": "file2.txt", "Line": "3", "Classes": [{"Name": "Serious_Drama", "Score":
 0.3141}, {"Name": "Other", "Score": 0.0381}, {"Name": "Other", "Score": 0.0372}]}
```

如果您的输入数据格式为每个文件一个文档，则输出文件包含每个文档一行。每行都有文件名和文档中找到的一个或多个类。以 Amazon Comprehend 对单个实例正确分类的置信度作为结束。

例如：

```
{"File": "file0.txt", "Classes": [{"Name": "Documentary", "Score": 0.8642}, {"Name":
 "Other", "Score": 0.0381}, {"Name": "Serious_Drama", "Score": 0.0372}]}
{"File": "file1.txt", "Classes": [{"Name": "Science_Fiction", "Score": 0.5}, {"Name":
 "Science_Fiction", "Score": 0.0381}, {"Name": "Science_Fiction", "Score": 0.0372}]}
{"File": "file2.txt", "Classes": [{"Name": "Documentary", "Score": 0.1}, {"Name":
 "Documentary", "Score": 0.0381}, {"Name": "Domentary", "Score": 0.0372}]}
{"File": "file3.txt", "Classes": [{"Name": "Serious_Drama", "Score": 0.3141}, {"Name":
 "Other", "Score": 0.0381}, {"Name": "Other", "Score": 0.0372}]}
```

多标签输出

当您使用在多标签模式下训练的分类器时，结果会显示 `labels`。这些标签中的每一个 `labels` 都是在训练分类器时用来创建类别集的标签。

以下示例使用这些独特的标签。

```
SCIENCE_FICTION
ACTION
DRAMA
COMEDY
ROMANCE
```

如果您的输入数据格式为每行一个文档，则输出文件中的每行包含输入中的一行。每行都包括文件名、输入行的从零开始的行号以及文档中找到的一个或多个类。以 Amazon Comprehend 对单个实例正确分类的置信度作为结束。

例如：

```
{"File": "file1.txt", "Line": "0", "Labels": [{"Name": "Action", "Score": 0.8642},
 {"Name": "Drama", "Score": 0.650}, {"Name": "Science Fiction", "Score": 0.0372}]}
{"File": "file1.txt", "Line": "1", "Labels": [{"Name": "Comedy", "Score": 0.5},
 {"Name": "Action", "Score": 0.0381}, {"Name": "Drama", "Score": 0.0372}]}
{"File": "file1.txt", "Line": "2", "Labels": [{"Name": "Action", "Score": 0.9934},
 {"Name": "Drama", "Score": 0.0381}, {"Name": "Action", "Score": 0.0372}]}
{"File": "file1.txt", "Line": "3", "Labels": [{"Name": "Romance", "Score": 0.9845},
 {"Name": "Comedy", "Score": 0.8756}, {"Name": "Drama", "Score": 0.7723}, {"Name":
 "Science_Fiction", "Score": 0.6157}]}
```

如果您的输入数据格式为每个文件一个文档，则输出文件包含每个文档一行。每行都有文件名和文档中找到的一个或多个类。以 Amazon Comprehend 对单个实例正确分类的置信度作为结束。

例如：

```
{"File": "file0.txt", "Labels": [{"Name": "Action", "Score": 0.8642}, {"Name": "Drama",
 "Score": 0.650}, {"Name": "Science Fiction", "Score": 0.0372}]}
{"File": "file1.txt", "Labels": [{"Name": "Comedy", "Score": 0.5}, {"Name": "Action",
 "Score": 0.0381}, {"Name": "Drama", "Score": 0.0372}]}
{"File": "file2.txt", "Labels": [{"Name": "Action", "Score": 0.9934}, {"Name": "Drama",
 "Score": 0.0381}, {"Name": "Action", "Score": 0.0372}]}
{"File": "file3.txt", "Labels": [{"Name": "Romance", "Score": 0.9845}, {"Name":
 "Comedy", "Score": 0.8756}, {"Name": "Drama", "Score": 0.7723}, {"Name":
 "Science_Fiction", "Score": 0.6157}]}
```

半结构化输入文档的输出

对于半结构化输入文档，输出可以包括以下附加字段：

- DocumentMetadata — 提取有关文档的信息。元数据包括文档中的页面列表，以及从每页中提取的字符数。如果请求包含 Byte 参数，则响应中会显示此字段。
- DocumentType -输入文档中每页的文档类型。如果请求包含 Byte 参数，则响应中会显示此字段。
- 错误：系统在处理输入文档时检测到的页面级错误。如果系统未遇到任何错误，则该字段为空。

有关这些输出字段的更多详细信息，请参阅ClassifyDocument《Amazon Comprehend API 参考》。

以下示例显示了两页扫描的 PDF 文件的输出。

```
[{ #First page output
    "Classes": [
        {
            "Name": "__label__2 ",
            "Score": 0.9993996620178223
        },
        {
            "Name": "__label__3 ",
            "Score": 0.0004330444789957255
        }
    ],
    "DocumentMetadata": {
        "PageNumber": 1,
        "Pages": 2
    },
    "DocumentType": "ScannedPDF",
    "File": "file.pdf",
    "Version": "VERSION_NUMBER"
},
#Second page output
{
    "Classes": [
        {
            "Name": "__label__2 ",
            "Score": 0.9993996620178223
        },
        {
            "Name": "__label__3 ",
            "Score": 0.0004330444789957255
        }
    ],
    "DocumentMetadata": {
```

```
        "PageNumber": 2,
        "Pages": 2
    },
    "DocumentType": "ScannedPDF",
    "File": "file.pdf",
    "Version": "VERSION_NUMBER"
}]
```

自定义实体识别

自定义实体识别可帮助您识别不在预设通用实体类型中的特定新实体类型，从而扩展了 Amazon Comprehend 的功能。这意味着您可以分析文档并提取符合您特定需求的实体，例如产品代码或业务特定实体。

自己构建精确的自定义实体识别器可能是一个复杂的过程，需要准备大量手动注释的训练文档，并选择正确的算法和参数进行模型训练。Amazon Comprehend 通过提供自动注释和模型开发来创建自定义实体识别模型，从而帮助降低复杂性。

与使用字符串匹配或正则表达式从文档中提取实体相比，创建自定义实体识别模型是一种更有效的方法。例如，要提取文档中的 ENGINEER 姓名，就很难枚举所有可能的名称。此外，如果没有上下文，很难区分 ENGINEER 姓名和 ANALYST 姓名。自定义实体识别模型可以了解这些名称可能出现的上下文。此外，字符串匹配不会检测到有错别字或遵循新命名约定的实体，而使用自定义模型可以做到这一点。

您可以通过两种方法来创建自定义模型：

1. 注释：提供包含带注释的实体的数据集，用于模型训练。
2. 实体列表（仅限纯文本）：提供实体列表及其类型标签（例如，PRODUCT_CODES 以及一组包含用于模型训练的这些实体的无注释文档）。

当您使用带注释的 PDF 文件创建自定义实体识别器时，您可以使用具有多种输入文件格式的识别器：纯文本、图像文件（JPG、PNG、TIFF）、PDF 文件和 Word 文档，无需预处理或拼合文档。Amazon Comprehend 不支持对图像文件或 Word 文档进行注释。

> **Note**
>
> 使用带注释的 PDF 文件的自定义实体识别器仅支持英文文档。

您一次最多可以在 25 个自定义实体上训练模型。有关更多详细信息，请参阅指南和配额页面。

训练完模型后，您可以使用该模型进行实时实体检测和实体检测作业。

主题

- 准备实体识别器训练数据

- 训练自定义实体识别器模型
- 运行实时自定义识别器分析
- 运行分析作业以识别自定义实体

准备实体识别器训练数据

要训练成功的自定义实体识别模型，向模型训练器提供高质量的数据作为输入是非常重要的。如果没有好的数据，模型就无法学会如何正确识别实体。

您可以选择以下两种方式之一向 Amazon Comprehend 提供数据，以便训练自定义实体识别模型：

- 实体列表：列出特定实体，这样 Amazon Comprehend 就可以通过训练来识别您的自定义实体。注意：实体列表只能用于纯文本文档。
- 注释：在多个文档中提供您的实体的位置，这样 Amazon Comprehend 就可以对实体及其上下文进行培训。要创建用于分析图像文件、PDF 或 Word 文档的模型，必须使用 PDF 注释训练识别器。

在这两种情况下，Amazon Comprehend 都会了解文档的类型和实体出现的背景，并构建一个识别器，该识别器可以在您分析文档时进行泛化以检测新实体。

创建自定义模型（或训练新版本）时，可以提供测试数据集。如果您不提供测试数据，Amazon Comprehend 将保留 10% 的输入文档用于测试模型。Amazon Comprehend 使用剩余的文档对模型进行训练。

如果您为注释训练集提供测试数据集，则测试数据必须包含创建请求中指定的每种实体类型的至少一个注释。

主题

- 何时使用注释与实体列表
- 实体列表（仅限纯文本）
- 注释

何时使用注释与实体列表

创建注释比创建实体列表需要更多的工作，但是生成的模型可以更加精确。使用实体列表更快，工作量也更少，但结果不够精细，也更不准确。这是因为注释为 Amazon Comprehend 提供了更多背景信

息，供其在训练模型时使用。如果没有这种背景，Amazon Comprehend 在尝试识别实体时会出现更多的误报。

在某些情况下，避免使用注释带来的更高费用和工作负载更具商业意义。例如，John Johnson 这个名字对你的搜索很重要，但它是否是确切的个人并不重要。或者，使用实体列表时的指标足以为您提供所需的识别器结果。在这种情况下，改用实体列表可能是更有效的选择。

我们建议在以下情况下使用注释模式：

- 如果您计划对图像文件、PDF 或 Word 文档进行推断。在这种情况下，您可以使用带注释的 PDF 文件训练模型，然后使用该模型为图像文件、PDF 和 Word 文档运行推理作业。
- 当实体的含义可能模棱两可且取决于上下文时。例如，Amazon一词可以指巴西的河流，也可以指在线零售商 Amazon.com。当您构建自定义实体识别器来识别Amazon等商业实体时，应使用注释而不是实体列表，因为此方法能够更好地使用上下文来查找实体。
- 当您愿意设置获取批注的流程时，这可能需要付出一些努力。

我们建议在以下情况下使用实体列表：

- 当你已经有了实体列表或者相对容易地撰写一份全面的实体列表时。如果您使用实体列表，则该列表应该是完整的，或者至少涵盖您提供的训练文件中可能出现的大多数有效实体。
- 对于首次使用的用户，通常建议使用实体列表，因为这比构造注释所需的精力更少。但是，请务必注意，经过训练的模型可能不如您使用注释时那样准确。

实体列表（仅限纯文本）

要使用实体列表训练模型，您需要提供两条信息：实体名称及其对应的自定义实体类型的列表，以及您希望实体出现在其中的未注释文档的集合。

当您提供实体列表时，Amazon Comprehend 会使用智能算法来检测文档中出现的实体，以此作为训练自定义实体识别器模型的基础。

对于实体列表，请在实体列表中为每种实体类型提供至少 25 个实体匹配项。

用于自定义实体识别的实体列表，需要一个逗号分隔值 (CSV) 文件，文件包含以下几列：

- 文本：条目示例的文本与随附的文档语料库中显示的完全相同。
- 类型：客户定义的实体类型。实体类型必须使用大写的、下划线分隔的字符串，例如 MANAGER 或 SENIOR_MANAGER。每个模型最多可以训练 25 种实体类型。

文件 documents.txt 包含四行：

```
Jo Brown is an engineer in the high tech industry.
John Doe has been a engineer for 14 years.
Emilio Johnson is a judge on the Washington Supreme Court.
Our latest new employee, Jane Smith, has been a manager in the industry for 4 years.
```

包含实体列表的 CSV 文件包含以下几行：

```
Text, Type
Jo Brown, ENGINEER
John Doe, ENGINEER
Jane Smith, MANAGER
```

> **Note**
>
> 在实体列表中，Emilio Johnson 的条目不存在，因为它不包含 ENGINEER 或 MANAGER 实体。

创建您的数据文件

请务必将您的实体列表放在正确配置的 CSV 文件中，这样您的实体列表文件出现问题的几率就会降至最低。要手动配置 CSV 文件，必须满足以下条件：

- 必须明确指定 UTF-8 编码，即使在大多数情况下将其用作默认编码。
- 它必须包含列名：Type 和 Text。

我们强烈建议您以编程方式生成 CSV 输入文件，以避免出现潜在问题。

以下示例使用 Python 为上面显示的注释生成 CSV：

```
import csv
with open("./entitylist/entitylist.csv", "w", encoding="utf-8") as csv_file:
    csv_writer = csv.writer(csv_file)
    csv_writer.writerow(["Text", "Type"])
    csv_writer.writerow(["Jo Brown", " ENGINEER"])
    csv_writer.writerow(["John Doe", " ENGINEER"])
```

```
    csv_writer.writerow(["Jane Smith", " MANAGER"])
```

最佳实践

要在使用实体列表时获得最佳结果，需要考虑很多因素，包括：

- 列表中实体的顺序对模型训练没有影响。
- 使用实体列表项目，这些项目应涵盖未注释文档语料库中提及的 80%-100% 的正面实体示例。
- 通过删除常用单词和短语，避免使用与文档语料库中非实体相匹配的实体示例。即使是少量不正确的匹配也会严重影响生成的模型的准确性。例如，实体列表中类似的单词将导致大量匹配项，而这些匹配项不太可能成为您要查找的实体，因此会严重影响您的准确性。
- 输入数据不应包含重复项。重复样本的存在可能会导致测试集污染，从而对训练过程、模型指标和模型行为产生负面影响。
- 提供尽可能与真实用例非常相似的文档。不要将玩具数据或合成数据用于生产系统。输入数据应尽可能多样化，以避免过度拟合，并帮助底层模型更好地概括真实示例。
- 实体列表区分大小写，目前不支持正则表达式。但是，即使实体与实体列表中提供的大小写不完全匹配，经过训练后的模型通常仍然可以识别实体。
- 如果您的实体是另一个实体的子字符串（例如“Smith”和“Jane Smith”），请在实体列表中同时提供这两个实体。

其他建议可以在 提高自定义实体识别器的性能 找到

注释

注释通过将您的自定义实体类型与培训文档中出现的位置相关联，在上下文中标注实体。

通过将注释与文档一起提交，可以提高模型的准确性。使用注释，您不仅可以提供要查找的实体的位置，还可以为要查找的自定义实体提供更准确的上下文。

例如，如果您正在搜索实体类型为 JUDGE 的 John Johnson 这个名字，那么提供注释可能有助于模型了解您要查找的人是一名法官。如果它能够使用上下文，那么 Amazon Comprehend 就不会找到名叫 John Johnson 的人是律师或证人。在不提供注释的情况下，Amazon Comprehend 将创建自己的注释版本，但在仅包括法官方面效果不佳。提供自己的注释可能有助于获得更好的结果，并生成能够在提取自定义实体时更好地利用上下文的模型。

主题

- 最小注释数量

- 注释最佳实践
- 纯文本注释文件
- PDF 注释文件
- 注释 PDF 文件

最小注释数量

训练模型所需的最小输入文档和注释数量取决于注释的类型。

PDF 注释

要创建用于分析图像文件、PDF 或 Word 文档的模型，请使用 PDF 注释训练识别器。对于 PDF 注释，请为每个实体提供至少 250 个输入文档和至少 100 个注释。

如果您提供测试数据集，则测试数据必须包含创建请求中指定的每种实体类型的至少一个注释。

纯文本注释

要创建用于分析文本文档的模型，您可以使用纯文本注释来训练识别器。

对于纯文本注释，请为每个实体提供至少三个带注释的输入文档和至少 25 个注释。如果您提供的注释总数少于 50 个，Amazon Comprehend 将保留超过 10% 的输入文档用于测试模型（除非您在训练请求中提供了测试数据集）。别忘了，最小文档语料库大小为 5 KB。

如果您的输入仅包含少量训练文档，则可能会遇到错误，即训练输入数据包含提及其中一个实体的文档太少。再次提交作业并附上提及该实体的其他文件。

如果您提供测试数据集，则测试数据必须包含创建请求中指定的每种实体类型的至少一个注释。

有关如何使用小型数据集对模型进行基准测试的示例，请参阅在 AWS 博客网站上 Amazon Comprehend 宣布降低自定义实体识别的注释限制。

注释最佳实践

要在使用注释时获得最佳结果，需要考虑很多因素，包括：

- 请谨慎地注释您的数据，并确认您是否对实体的每个提及进行了注释。不精确的注释可能会导致结果不佳。
- 输入数据不应包含重复项，例如您要进行注释的 PDF 的副本。重复样本的存在可能会导致测试集污染，并可能对训练过程、模型指标和模型行为产生负面影响。

- 确保您的所有文档都带有注释，并且没有注释的文档是由于缺乏合法实体而不是疏忽造成的。例如，如果你有一份写着“J Doe 当工程师已有 14 年了”的文档，那么你还应该为“J Doe”和“John Doe”提供注释。如果不这样做，会使模型混淆，并可能导致模型无法将“J Doe”识别为 ENGINEER。这应该在同一文档中和跨文档中保持一致。
- 通常，注释越多，结果越好。
- 您可以使用最少数量的文档和注释来训练模型，但是增加数据通常可以改进模型。我们建议将带注释的数据量增加 10%，以提高模型的准确性。您可以对测试数据集进行推理，该数据集保持不变，并且可以通过不同的模型版本进行测试。然后，您可以比较后续模型版本的指标。
- 提供尽可能与真实用例非常相似的文档。应避免使用重复模式的合成数据。输入数据应尽可能多样化，以避免过度拟合，并帮助底层模型更好地概括真实示例。
- 重要的是，文档的字数应多种多样。例如，如果训练数据中的所有文档都很短，则生成的模型可能难以预测较长文档中的实体。
- 尝试为训练提供与实际检测自定义实体（推理时间）时预期使用的相同的数据分布。例如，在推理时，如果您希望向我们发送的文档中没有实体，那么这也应该是您的训练文档集的一部分。

有关其他建议，请参阅提高自定义实体识别器性能。

纯文本注释文件

对于纯文本注释，您可以创建包含注释列表的逗号分隔值 (CSV) 文件。如果您的训练文件输入格式为每行一个文档，则 CSV 文件必须包含以下列。

文件	行	开始偏移量	结束偏移量	Type
包含文档的文件的名称。例如，如果其中一个文档文件位于 `s3://my-S3-bucket/test-files/documents.txt` ，则该 `File` 列中的值将为 `documents.txt` 。文件名	包含实体的行号。如果您的输入格式为每个文件一个文档，请省略此列。	输入文本中的字符偏移量（相对于行的开头），显示了实体的开始位置。第一个字符位于位置 0。	输入文本中的字符偏移量，用于显示实体的结束位置。	客户定义的实体类型。实体类型必须是大写的、用下划线分隔的字符串。我们建议使用描述性实体类型，例如 `MANAGER`、`SENIOR_MANAGER` 、或 `PRODUCT_CODE` 。每个模型

文件	行	开始偏移量	结束偏移量	Type
中必须包含文件扩展名（在本例中为“.txt”）。				最多可以训练 25 种实体类型。

如果您的训练文件输入格式为每个文件一个文档，则省略行号列，并且开始偏移量和结束偏移量值是实体相对于文档开头的偏移量。

以下示例适用于每行一个文档。文件 documents.txt 包含四行（第 0、1、2 和 3 行）：

```
Diego Ramirez is an engineer in the high tech industry.
Emilio Johnson has been an engineer for 14 years.
J Doe is a judge on the Washington Supreme Court.
Our latest new employee, Mateo Jackson, has been a manager in the industry for 4 years.
```

带有注释列表的 CSV 文件如下所示：

```
File, Line, Begin Offset, End Offset, Type
documents.txt, 0, 0, 13, ENGINEER
documents.txt, 1, 0, 14, ENGINEER
documents.txt, 3, 25, 38, MANAGER
```

> **Note**
>
> 在注释文件中，包含实体的行号从第 0 行开始。在此示例中，CSV 文件不包含第 2 行的条目，因为 documents.txt 的第 2 行中没有实体。

创建您的数据文件

请务必将您的注释放在正确配置的 CSV 文件中，以降低出错的风险。要手动配置 CSV 文件，必须满足以下条件：

- 必须明确指定 UTF-8 编码，即使在大多数情况下将其用作默认编码。
- 第一行包含列标题：File、Line（可选）、Begin Offset、End Offset、Type。

我们强烈建议您以编程方式生成 CSV 输入文件，以避免出现潜在问题。

以下示例使用 Python 为前面显示的注释生成 CSV：

```
import csv
with open("./annotations/annotations.csv", "w", encoding="utf-8") as csv_file:
    csv_writer = csv.writer(csv_file)
    csv_writer.writerow(["File", "Line", "Begin Offset", "End Offset", "Type"])
    csv_writer.writerow(["documents.txt", 0, 0, 11, "ENGINEER"])
    csv_writer.writerow(["documents.txt", 1, 0, 5, "ENGINEER"])
    csv_writer.writerow(["documents.txt", 3, 25, 30, "MANAGER"])
```

PDF 注释文件

对于 PDF 批注，您可以使用 SageMaker Ground Truth 在增强的清单文件中创建带标签的数据集。Ground Truth 是一项数据标注服务，可帮助您（或您雇用的人力）为机器学习模型构建训练数据集。Amazon Comprehend 接受增强的清单文件作为自定义模型的训练数据。您可以在使用 Amazon Comprehend 控制台或 API 操作创建自定义实体识别器时提供这些文件。CreateEntityRecognizer

您可以使用 Ground Truth 内置作业类型“命名实体识别”来创建标注作业，让工作人员识别文本中的实体。要了解更多信息，请参阅《Amazon SageMaker 开发者指南》中的命名实体识别。要了解有关 Amazon G SageMaker round Truth 的更多信息，请参阅使用 Amazon G SageMaker round Truth 为数据添加标签。

> **Note**
>
> 使用 Ground Truth，您可以定义重叠的标签（与多个标签关联的文本）。但是，Amazon Comprehend 实体识别不支持重叠的标签。

增强清单文件采用 JSON 行格式。文件中的每一行都是一个完整的 JSON 对象，其中包含一个训练文档及其关联的标签。以下示例是一个增强的清单文件，用于训练实体识别器来检测文本中提及的个人的职业：

```
{"source":"Diego Ramirez is an engineer in the high tech
 industry.","NamedEntityRecognitionDemo":{"annotations":{"entities":
[{"endOffset":13,"startOffset":0,"label":"ENGINEER"}],"labels":
[{"label":"ENGINEER"}]}},"NamedEntityRecognitionDemo-metadata":
{"entities":[{"confidence":0.92}],"job-name":"labeling-job/
namedentityrecognitiondemo","type":"groundtruth/text-span","creation-
date":"2020-05-14T21:45:27.175903","human-annotated":"yes"}}
{"source":"J Doe is a judge on the Washington Supreme
 Court.","NamedEntityRecognitionDemo":{"annotations":{"entities":
```

```
[{"endOffset":5,"startOffset":0,"label":"JUDGE"}],"labels":
[{"label":"JUDGE"}]}},"NamedEntityRecognitionDemo-metadata":
{"entities":[{"confidence":0.72}],"job-name":"labeling-job/
namedentityrecognitiondemo","type":"groundtruth/text-span","creation-
date":"2020-05-14T21:45:27.174910","human-annotated":"yes"}}
{"source":"Our latest new employee, Mateo Jackson, has been a manager in
 the industry for 4 years.","NamedEntityRecognitionDemo":{"annotations":
{"entities":[{"endOffset":38,"startOffset":26,"label":"MANAGER"}],"labels":
[{"label":"MANAGER"}]}},"NamedEntityRecognitionDemo-metadata":
{"entities":[{"confidence":0.91}],"job-name":"labeling-job/
namedentityrecognitiondemo","type":"groundtruth/text-span","creation-
date":"2020-05-14T21:45:27.174035","human-annotated":"yes"}}
```

此 JSON 行文件中的每一行都是一个完整的 JSON 对象，其中的属性包括文档文本、注释和 Ground Truth 中的其他元数据。以下示例是增强清单文件中的单个 JSON 对象，该对象已格式化以提高可读性：

```
{
  "source": "Diego Ramirez is an engineer in the high tech industry.",
  "NamedEntityRecognitionDemo": {
    "annotations": {
      "entities": [
        {
          "endOffset": 13,
          "startOffset": 0,
          "label": "ENGINEER"
        }
      ],
      "labels": [
        {
          "label": "ENGINEER"
        }
      ]
    }
  },
  "NamedEntityRecognitionDemo-metadata": {
    "entities": [
      {
        "confidence": 0.92
      }
    ],
    "job-name": "labeling-job/namedentityrecognitiondemo",
    "type": "groundtruth/text-span",
```

```
    "creation-date": "2020-05-14T21:45:27.175903",
    "human-annotated": "yes"
  }
}
```

在此示例中，source 属性提供训练文档的文本，NamedEntityRecognitionDemo 属性为文本中的实体提供注释。NamedEntityRecognitionDemo 属性的名称是任意的，在 Ground Truth 中定义标注作业时，您可以提供自己选择的名称。

在此示例中，NamedEntityRecognitionDemo 属性是标签属性名称，它是提供 Ground Truth 工作人员分配给训练数据的标签的属性。当您向 Amazon Comprehend 提供训练数据时，您必须指定一个或多个标签属性名称。您指定的属性名称数量取决于您的增强清单文件是单个标注作业的输出还是链式标注作业的输出。

如果您的文件是单个标注作业的输出，请指定 Ground Truth 中创建作业时使用的单个标签属性名称。

如果您的文件是链式标注作业的输出，请为链中的一个或多个作业指定标签属性名称。每个标签属性名称都提供来自单个作业的注释。您最多可以为链式标注作业生成的增强清单文件指定 5 个此类属性。

在增强的清单文件中，标签属性名称通常紧随 source 键。如果该文件是链式作业的输出，则会有多个标签属性名称。当您向 Amazon Comprehend 提供训练数据时，请仅提供包含与您的模型相关的注释的属性。请勿指定以“-元数据”结尾的属性。

有关链式标签任务的更多信息以及它们产生的输出示例，请参阅《Amazon SageMaker 开发者指南》中的 “链接标签任务”。

注释 PDF 文件

在 G SageMaker round Truth 中为训练 PDF 添加注释之前，请先完成以下先决条件：

- 安装 python3.8.x
- 安装 jq
- 安装 AWS CLI

  如果你使用的是 us-east-1 区域，则可以跳过安装 AWS CLI，因为它已经安装在你的 Python 环境中。在本例中，您可以创建一个虚拟环境，以便在 AWS Cloud9 中使用 Python 3.8。
- 配置您的AWS 凭证
- 创建一支私人 G SageMaker round Truth 员工队伍来支持注释

  请务必记录您选择的工作团队名称记录在新的私有人力中，因为在安装过程中会用到它。

主题

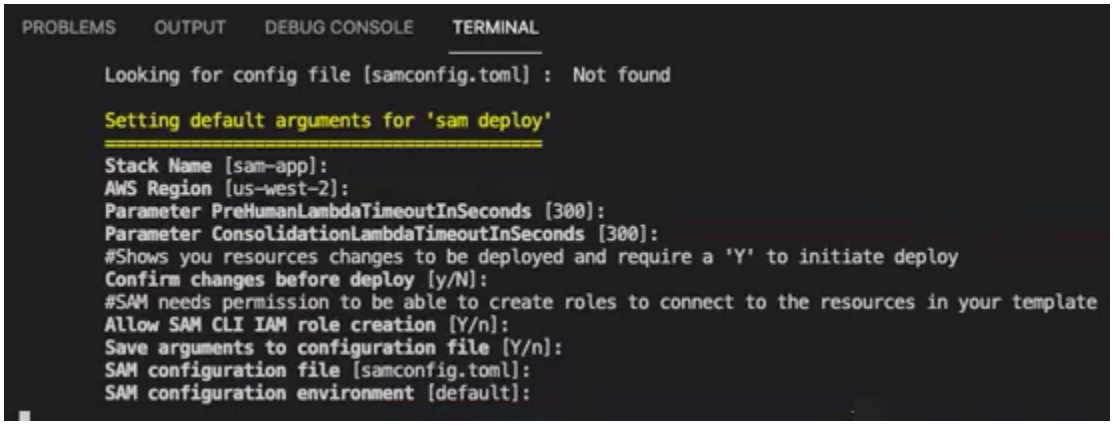
- 设置环境
- 将 PDF 上传到 S3 存储桶
- 创建注释作业
- 使用 Ground Truth 进行 SageMaker 注释

设置环境

1. 如果使用 Windows，请安装 Cygwin；如果使用的是 Linux 或 Mac，请跳过此步骤。
2. 从中下载注释工件 GitHub。解压缩该文件。
3. 在终端窗口中，导航到解压后的文件夹 (amazon-comprehend-semi-structured-documents-annotation-tools-main)。
4. 这个文件夹包括一个选项 `Makefiles`，您可以运行它来安装依赖项、设置 Python virtualenv 和部署所需的资源。查看自述文件以做出选择。
5. 推荐的选项使用单个命令将所有依赖项安装到 virtualenv 中，根据模板构建 AWS CloudFormation 堆栈，然后 AWS 账户 通过交互式指导将堆栈部署到您的。运行以下命令：

   ```
   make ready-and-deploy-guided
   ```

   此命令提供了一组配置选项。请确保你的正确 AWS 区域 无误。对于所有其他字段，您可以接受默认值或填写自定义值。如果您修改了 AWS CloudFormation 堆栈名称，请在接下来的步骤中根据需要将其写下来。

   CloudFormation 堆栈创建和管理注解工具所需的 AWS lambda、AWS IAM 角色和 AWS S3 存储桶。

   您可以在 CloudFormation 控制台的堆栈详细信息页面中查看这些资源中的每一个资源。

6. 该命令会提示您开始部署。 CloudFormation 在指定区域创建所有资源。

当 CloudFormation 堆栈状态转换为 “创建完成” 时，资源就可以使用了。

将 PDF 上传到 S3 存储桶

在设置部分，您部署了一个 CloudFormation 堆栈，该堆栈用于创建名为 comprehend-semi-structured-documents-$ {AWS::Region}-${AWS::AccountId} 的 S3 存储桶。现在，您可以将源 PDF 文档上传到此存储桶中。

> **Note**
>
> 此存储桶包含您的标注作业所需的数据。Lambda 执行角色策略向 Lambda 函数授予访问此存储桶的权限。
> 您可以使用 “S3Bucket” 密钥在CloudFormation 堆栈详细信息中找到 SemiStructuredDocumentsS3 存储桶名称。

1. 在 S3 存储桶中，创建一个新的文件夹。将这个新文件夹命名为src。
2. 将您的 PDF 源文件添加到src文件夹。在后面的步骤中，将对这些文件进行注释以训练识别器。
3. （可选）以下是可用于将源文档从本地目录上传到 S3 存储桶的 AWS CLI 示例：

   ```
   aws s3 cp --recursive local-path-to-your-source-docs s3://deploy-guided/src/
   ```

   或者，使用您的地区和账户 ID：

   ```
   aws s3 cp --recursive local-path-to-your-source-docs s3://deploy-guided-Region-AccountID/src/
   ```
4. 你现在有了一名私人 G SageMaker round Truth 工作人员，并且已经将源文件上传到 S3 存储桶 deplo y-guided/src/；你已经准备好开始添加注释了。

创建注释作业

bin目录中的 comprehend-ssie-annotation-tool-cli.py 脚本是一个简单的包装器命令，它简化了 G SageMaker round Truth 标签作业的创建。python 脚本从 S3 存储桶中读取源文档，并创建相应的单页清单文件，每行包含一个源文档。然后，该脚本会创建一个标签作业，该作业需要清单文件作为输入。

python 脚本使用您在设置部分配置的 S3 存储桶和 CloudFormation 堆栈。脚本所需的输入参数包括：

- input-s3-path：您上传到 S3 存储桶的源文档的 S3 Uri。例如：`s3://deploy-guided/src/`。您还可将您的地区和账户 ID 添加到此路径。例如：`s3://deploy-guided-Region-AccountID/src/`。
- cfn-name： CloudFormation 堆栈名称。如果您使用堆栈名称的默认值，则您的 cfn-name 为 sam-app。
- work-team-name: 你在 G SageMaker round Truth 中建立私人员工队伍时创建的员工队伍名称。
- job-name-prefix: G SageMaker round Truth 标签作业的前缀。请注意，此字段有 29 个字符的限制。此值后面会附加一个时间戳。例如：`my-job-name-20210902T232116`。
- entity-types：您要在标注作业中使用的实体，请用逗号分隔。此列表必须包括您要在训练数据集中注释的所有实体。Ground Truth 标注作业仅显示这些实体，供注释者标注 PDF 文档中的内容。

要查看脚本支持的其他参数，请使用 -h 选项显示帮助内容。

- 使用上面列表中所述的输入参数运行以下脚本。

```
python bin/comprehend-ssie-annotation-tool-cli.py \
--input-s3-path s3://deploy-guided-Region-AccountID/src/ \
--cfn-name sam-app \
--work-team-name my-work-team-name \
--region us-east-1 \
--job-name-prefix my-job-name-20210902T232116 \
--entity-types "EntityA, EntityB, EntityC" \
--annotator-metadata "key=info,value=sample,key=Due Date,value=12/12/2021"
```

该脚本生成以下输出：

```
Downloaded files to temp local directory /tmp/a1dc0c47-0f8c-42eb-9033-74a988ccc5aa
Deleted downloaded temp files from /tmp/a1dc0c47-0f8c-42eb-9033-74a988ccc5aa
Uploaded input manifest file to s3://comprehend-semi-structured-documents-
us-west-2-123456789012/input-manifest/my-job-name-20220203-labeling-
job-20220203T183118.manifest
```

```
Uploaded schema file to s3://comprehend-semi-structured-documents-us-
west-2-123456789012/comprehend-semi-structured-docs-ui-template/my-job-
name-20220203-labeling-job-20220203T183118/ui-template/schema.json
Uploaded template UI to s3://comprehend-semi-structured-documents-us-
west-2-123456789012/comprehend-semi-structured-docs-ui-template/my-job-
name-20220203-labeling-job-20220203T183118/ui-template/template-2021-04-15.liquid
Sagemaker GroundTruth Labeling Job submitted: arn:aws:sagemaker:us-
west-2:123456789012:labeling-job/my-job-name-20220203-labeling-job-20220203t183118
(amazon-comprehend-semi-structured-documents-annotation-tools-main)
 user@3c063014d632 amazon-comprehend-semi-structured-documents-annotation-tools-
main %
```

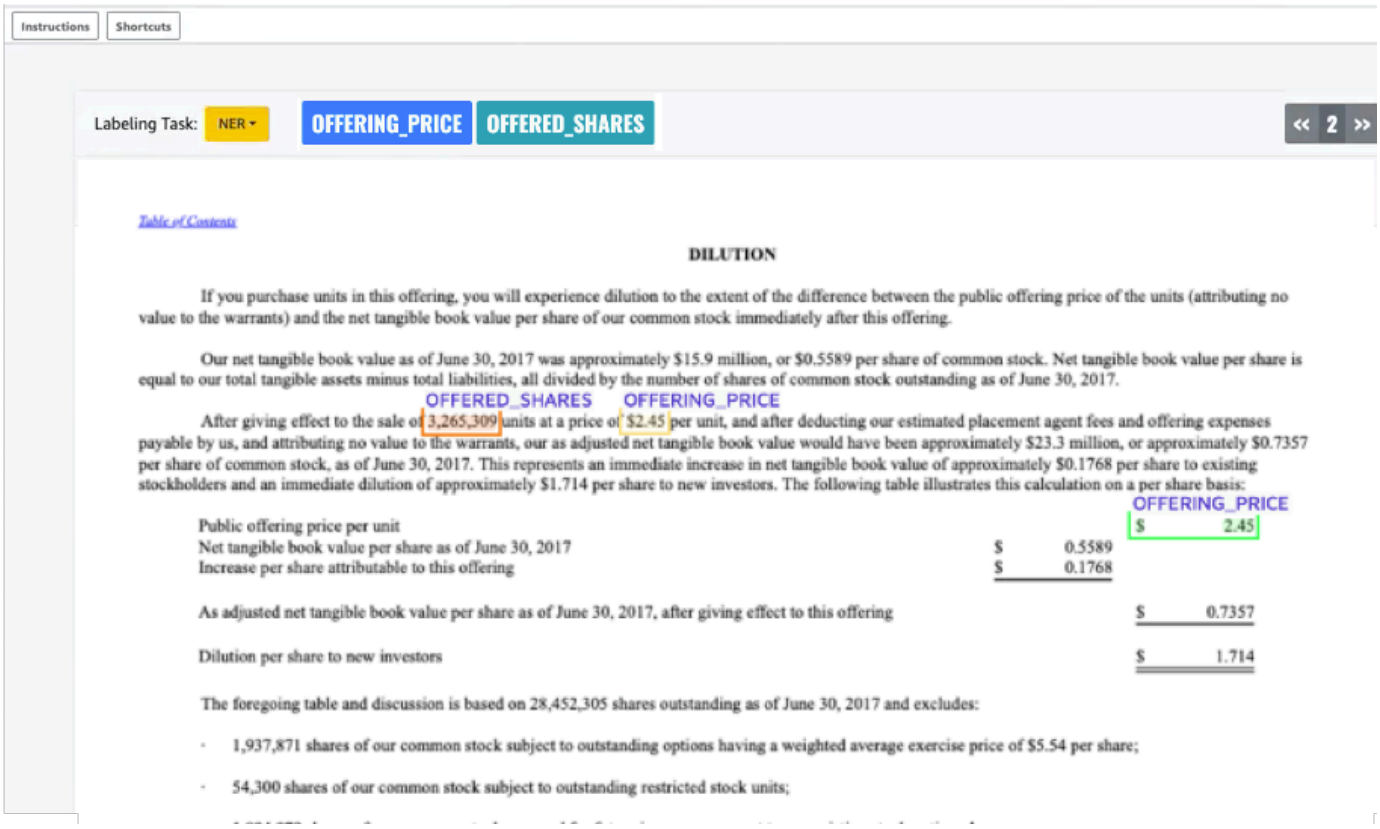
使用 Ground Truth 进行 SageMaker 注释

现在，您已经配置了所需的资源并创建了标注作业，您可以登录标注门户并为 PDF 添加注释。

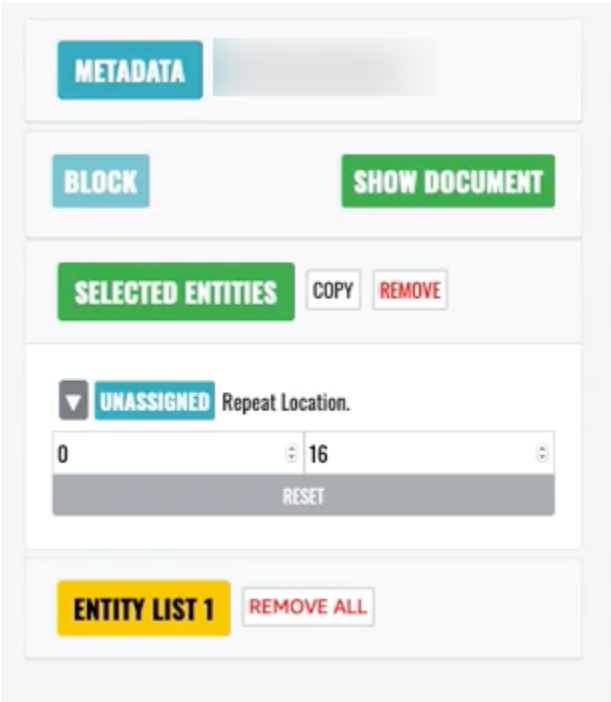
1. 使用 Chrome 或 Firefox 网络浏览器登录SageMaker 控制台。
2. 选择标注人力，然后选择私有。
3. 在私有人力摘要下，选择您使用私有人力创建的标注门户登录网址。使用适当的凭证登录。

   如果您没有看到列出的任何作业，请不要担心，更新可能需要一段时间，具体取决于您上传用于注释的文件数量。
4. 选择您的作业，然后在右上角，选择开始工作，以打开注释屏幕。

   您将在注释屏幕中看到其中一个文档处于打开状态，并在其上方看到您在设置过程中提供的实体类型。在实体类型的右边，有一个箭头，您可以用它来浏览你的文档。



为打开的文档添加注释。您还可以删除、撤消或自动标记每个文档上的注释；这些选项可在注释工具的右侧面板中找到。



要使用自动标记，请为其中一个实体的实例添加注释；然后，该特定单词的所有其他实例都将自动使用该实体类型进行注释。

完成后，选择右下角的提交，然后使用导航箭头移至下一个文档。重复此操作，直到为所有 PDF 添加了注释。

为所有训练文档添加注释后，您可以在 Amazon S3 存储桶中的以下位置找到 JSON 格式的注释：

```
/output/your labeling job name/annotations/
```

输出文件夹还包含一个输出清单文件，其中列出了您的训练文档中的所有注释。您可以在以下位置找到您的输出清单文件。

```
/output/your labeling job name/manifests/
```

训练自定义实体识别器模型

自定义实体识别器仅识别您在训练模型时包含的实体类型。它不会自动包括预设的实体类型。如果您还想识别预设的实体类型，例如位置、日期或人员，则需要为这些实体提供其他训练数据。

当您使用带注释的 PDF 文件创建自定义实体识别器时，您可以使用具有多种输入文件格式的识别器：纯文本、图像文件（JPG、PNG、TIFF）、PDF 文件和 Word 文档，无需预处理或拼合文档。Amazon Comprehend 不支持对图像文件或 Word 文档进行注释。

> **Note**
>
> 使用带注释的 PDF 文件的自定义实体识别器仅支持英文文档。

创建自定义实体识别器后，您可以使用DescribeEntityRecognizer操作监控请求的进度。当 `Status` 字段为 `TRAINED` 时，识别器模型即可用于自定义实体识别。

主题

- 训练自定义识别器（控制台）
- 训练自定义实体识别器 (API)
- 自定义实体识别器指标

训练自定义识别器（控制台）

您可以使用 Amazon Comprehend 控制台创建自定义实体识别器。本节说明了如何创建和训练自定义实体识别器。

使用控制台创建自定义实体识别器 - CSV 格式

要创建自定义实体识别器，请先提供一个用于训练模型的数据集。该数据集包括以下内容之一：一组带注释的文档或一组实体及其类型标签的列表，以及一组包含这些实体的文档。有关更多信息，请参阅自定义实体识别。

使用 CSV 文件训练自定义实体识别器

1. 登录 AWS Management Console 并打开亚马逊 Comprehend 控制台，网址为 https://console.aws.amazon.com/comprehend/
2. 从左侧菜单中选择自定义，然后选择自定义实体识别。
3. 选择创建新模型。
4. 为识别器命名。该名称在区域和账户中必须是唯一的。
5. 选择语言。
6. 在自定义实体类型下，输入您希望识别器在数据集中找到的自定义标签。

   实体类型必须大写，如果由多个单词组成，则用下划线分隔单词。
7. 选择添加类型。
8. 如果要添加其他实体类型，请输入该类型，然后选择添加类型。如果要删除已添加的实体类型，请选择删除类型，然后选择要从列表中删除的实体类型。最多可以列出 25 种实体类型。
9. 要对训练作业加密，请选择识别器加密，然后选择是使用与当前账户关联的 KMS 密钥，还是使用来自其他账户的 KMS 密钥。

   - 如果您使用的是与当前账户关联的密钥，请为 KMS 密钥 ID 选择密钥 ID。
   - 如果您使用与其他账户关联的密钥，请在 KMS 密钥 ARN 中输入密钥 ID 的 ARN。

> **Note**
>
> 有关创建和使用 KMS 密钥以及相关加密的更多信息，请参阅 AWS Key Management Service。

10. 在数据规范下，选择训练文档的格式：

    - CSV 文件：补充您的训练文档的 CSV 文件。CSV 文件包含有关您的训练模型将检测到的自定义实体的信息。所需的文件格式取决于您提供的是注释还是实体列表。
    - 增强清单 — 由 Amazon G SageMaker round Truth 生成的带标签的数据集。该文件采用 JSON 行格式。每一行都是一个完整的 JSON 对象，其中包含一个训练文档及其标签。每个标签都对训练文档中的一个命名实体进行注释。您最多可以提供 5 个增强的清单文件。

    有关可用格式的更多信息以及示例，请参阅 训练自定义实体识别器模型。

11. 在训练类型下，选择要使用的训练类型：

    - 使用注释与训练文档
    - 使用实体列表与训练文档

    如果选择注释，请在 Amazon S3 中输入注释文件的 URL。您也可以导航到 Amazon S3 中注释文件所在的存储桶或文件夹，然后选择浏览 S3。

    如果选择实体列表，请在 Amazon S3 中输入实体列表的 URL。您也可以导航到 Amazon S3 中实体列表所在的存储桶或文件夹，然后选择浏览 S3。

12. 在 Amazon S3 中输入包含训练文档的输入数据集的网址。您也可以导航到 Amazon S3 中训练文档所在的存储桶或文件夹，然后选择选择文件夹。

13. 在测试数据集下，选择您想要如何评估训练模型的性能-您可以对注释和实体列表训练类型执行此操作。

    - 自动分割：自动分割会自动选择您提供的训练数据的 10% 用作测试数据
    - （可选）客户提供：当您选择客户提供的时，您可以准确指定要使用的测试数据。

14. 如果您选择客户提供的测试数据集，请在 Amazon S3 中输入注释文件的 URL。您也可以导航到 Amazon S3 中注释文件所在的存储桶或文件夹，然后选择选择文件夹。

15. 在选择 IAM 角色部分中，选择一个现有 IAM 角色，或者创建一个新的 IAM 角色。

    - 选择现有 IAM 角色：如果您已经拥有有权访问输入和输出 Amazon S3 存储桶的 IAM 角色，请选择此选项。
    - 创建新的 IAM 角色：如果您要创建一个新的 IAM 角色，该角色具有适当的权限，让 Amazon Comprehend 可以访问输入和输出存储桶，请选择此选项。

> **Note**
>
> 如果输入文档已加密，则所使用的 IAM 角色必须具有 `kms:Decrypt` 权限。有关更多信息，请参阅 使用 KMS 加密所需的权限。

16. （可选）要将您的资源从 VPC 启动到 Amazon Comprehend，请在 VPC 下输入 VPC ID 或从下拉列表中选择 ID。

    1. 在子网下选择子网。选择第一个子网后，您还可以选择其他子网。
    2. 在安全组下，选择要使用的安全组（如果已指定）。选择第一个安全组后，您还可以选择其他安全组。

> **Note**
>
> 当您在自定义实体识别任务中使用 VPC 时，`DataAccessRole` 用于创建和启动操作的用户必须拥有访问输入文档和输出存储桶的 VPC 的权限。

17. （可选）若要向自定义实体识别器添加标签，请在标签下输入键值对。选择添加标签。要在创建识别器之前删除这对，请选择删除标签。
18. 选择训练。

然后，新的识别器将出现在列表中，显示其状态。它将首先显示为 `Submitted`。然后，它将显示 `Training` 正在处理训练文档的分类器、`Trained` 准备就绪的分类器以及存在 `In error` 错误的分类器。您可以单击作业以获取有关识别器的更多信息，包括任何错误消息。

使用控制台创建自定义实体识别器-增强清单

使用纯文本、PDF 或 Word 文档训练自定义实体识别器

1. 登录 AWS Management Console 并打开 Amazon Comprehend 控制台。
2. 从左侧菜单中选择自定义，然后选择自定义实体识别。
3. 选择训练识别器。
4. 为识别器命名。该名称在区域和账户中必须是唯一的。
5. 选择语言。注意：如果您正在训练 PDF 或 Word 文档，则支持的语言为英语。
6. 在自定义实体类型下，输入您希望识别器在数据集中找到的自定义标签。

实体类型必须大写，如果由多个单词组成，则用下划线分隔单词。

7. 选择添加类型。
8. 如果要添加其他实体类型，请输入该类型，然后选择添加类型。如果要删除已添加的实体类型，请选择删除类型，然后选择要从列表中删除的实体类型。最多可以列出 25 种实体类型。
9. 要对训练作业加密，请选择识别器加密，然后选择是使用与当前账户关联的 KMS 密钥，还是使用来自其他账户的 KMS 密钥。

   - 如果您使用的是与当前账户关联的密钥，请为 KMS 密钥 ID 选择密钥 ID。
   - 如果您使用与其他账户关联的密钥，请在 KMS 密钥 ARN 中输入密钥 ID 的 ARN。

   > **Note**
   >
   > 有关创建和使用 KMS 密钥以及相关加密的更多信息，请参阅 AWS Key Management Service。

10. 在训练数据下，选择增强清单作为数据格式：

    - 增强清单 — 是由 Amazon G SageMaker round Truth 生成的带标签的数据集。该文件采用 JSON 行格式。文件中的每一行都是一个完整的 JSON 对象，其中包含一个训练文档及其标签。每个标签都对训练文档中的一个命名实体进行注释。您最多可以提供 5 个增强的清单文件。如果您使用 PDF 文档作为训练数据，则必须选择增强清单。您最多可以提供 5 个增强的清单文件。每个文件最多可命名 5 个属性作为训练数据。

    有关可用格式的更多信息以及示例，请参阅 训练自定义实体识别器模型。

11. 选择训练模型类型。

    如果您选择了纯文本文档，请在输入位置下输入 Amazon Tr SageMakerGround uth 增强清单文件的 Amazon S3URL。您也可以导航到 Amazon S3 中增强清单文件所在的存储桶或文件夹，然后选择选择文件夹。

12. 在属性名称下，输入包含注释的属性的名称。如果文件包含来自多个链式标注任务的注释，请为每个作业添加一个属性。在这种情况下，每个属性都包含标注作业中的一组注释。注意：您最多可以为每个文件提供 5 个属性名称。
13. 选择添加。

14. 如果你在 “输入位置” 下选择了 PDF、Word 文档，请输入 Amazon G SageMaker round Truth 增强清单文件的 Amazon S3URL。您也可以导航到 Amazon S3 中增强清单文件所在的存储桶或文件夹，然后选择选择文件夹。
15. 输入注释数据文件的 S3 前缀。这些是您标记的 PDF 文档。
16. 输入源文档的 S3 前缀。这些是您提供给 Ground Truth 进行标注作业的原始 PDF 文档（数据对象）。
17. 输入包含注释的属性名称。注意：您最多可以为每个文件提供 5 个属性名称。文件中任何您未指定的属性都将被忽略。
18. 在 IAM 角色部分，选择一个现有 IAM 角色，或者创建一个新的 IAM 角色。
    - 选择现有 IAM 角色：如果您已经拥有有权访问输入和输出 Amazon S3 存储桶的 IAM 角色，请选择此选项。
    - 创建新的 IAM 角色：如果您要创建一个新的 IAM 角色，该角色具有适当的权限，让 Amazon Comprehend 可以访问输入和输出存储桶，请选择此选项。

    > **Note**
    >
    > 如果输入文档已加密，则所使用的 IAM 角色必须具有 `kms:Decrypt` 权限。有关更多信息，请参阅 使用 KMS 加密所需的权限。

19. （可选）要将您的资源从 VPC 启动到 Amazon Comprehend，请在 VPC 下输入 VPC ID 或从下拉列表中选择 ID。
    1. 在子网下选择子网。选择第一个子网后，您还可以选择其他子网。
    2. 在安全组下，选择要使用的安全组（如果已指定）。选择第一个安全组后，您还可以选择其他安全组。

    > **Note**
    >
    > 当您在自定义实体识别任务中使用 VPC 时，`DataAccessRole` 用于创建和启动操作的用户必须拥有访问输入文档和输出存储桶的 VPC 的权限。

20. （可选）若要向自定义实体识别器添加标签，请在标签下输入键值对。选择添加标签。要在创建识别器之前删除这对，请选择删除标签。
21. 选择训练。

然后，新的识别器将出现在列表中，显示其状态。它将首先显示为 Submitted。然后，它将显示 Training 正在处理训练文档的分类器、Trained 准备就绪的分类器以及存在 In error 错误的分类器。您可以单击作业以获取有关识别器的更多信息，包括任何错误消息。

训练自定义实体识别器 (API)

要创建和训练自定义实体识别模型，请使用 Amazon Comprehend API CreateEntityRecognizer操作

主题

- 使用 AWS Command Line Interface 训练自定义实体识别器
- 使用 AWS SDK for Java 训练自定义实体识别器
- 使用 Python (Boto3) 训练自定义实体识别器

使用 AWS Command Line Interface 训练自定义实体识别器

以下示例演示了如何将 CreateEntityRecognizer 操作和其他相关的 API 与 AWS CLI 结合使用。

此示例的格式适用于 Unix、Linux 和 macOS。对于 Windows，请将每行末尾的反斜杠 (\) Unix 行继续符替换为脱字号 (^)。

使用 create-entity-recognizer CLI 命令创建自定义实体识别器。有关 input-data-config 参数的信息，请参阅《亚马逊 Comprehend API 参考》CreateEntityRecognizer中的。

```
aws comprehend create-entity-recognizer \
     --language-code en \
     --recognizer-name test-6 \
     --data-access-role-arn "arn:aws:iam::account number:role/service-role/
AmazonComprehendServiceRole-role" \
     --input-data-config "EntityTypes=[{Type=PERSON}],Documents={S3Uri=s3://Bucket
 Name/Bucket Path/documents},
                 Annotations={S3Uri=s3://Bucket Name/Bucket Path/annotations}" \
     --region region
```

使用 list-entity-recognizers CLI 命令列出一个区域中的所有实体识别器。

```
aws comprehend list-entity-recognizers \
     --region region
```

使用 describe-entity-recognizer CLI 命令检查自定义实体识别器的作业状态。

```
aws comprehend describe-entity-recognizer \
     --entity-recognizer-arn arn:aws:comprehend:region:account number:entity-
recognizer/test-6 \
     --region region
```

使用 AWS SDK for Java 训练自定义实体识别器

此示例使用 Java 创建自定义实体识别器并训练模型

有关使用 Java 的 Amazon Comprehend 示例，请参阅 Amazon Comprehend Java 示例。

使用 Python (Boto3) 训练自定义实体识别器

实例化 Boto3 SDK：

```
import boto3
import uuid
comprehend = boto3.client("comprehend", region_name="region")
```

创建实体识别器：

```
response = comprehend.create_entity_recognizer(
    RecognizerName="Recognizer-Name-Goes-Here-{}".format(str(uuid.uuid4())),
    LanguageCode="en",
    DataAccessRoleArn="Role ARN",
    InputDataConfig={
        "EntityTypes": [
            {
                "Type": "ENTITY_TYPE"
            }
        ],
        "Documents": {
            "S3Uri": "s3://Bucket Name/Bucket Path/documents"
        },
        "Annotations": {
            "S3Uri": "s3://Bucket Name/Bucket Path/annotations"
        }
    }
)
recognizer_arn = response["EntityRecognizerArn"]
```

列出所有识别器：

```
response = comprehend.list_entity_recognizers()
```

等待识别器达到“已训练”状态：

```
while True:
    response = comprehend.describe_entity_recognizer(
        EntityRecognizerArn=recognizer_arn
    )

    status = response["EntityRecognizerProperties"]["Status"]
    if "IN_ERROR" == status:
        sys.exit(1)
    if "TRAINED" == status:
        break

    time.sleep(10)
```

自定义实体识别器指标

Amazon Comprehend 为您提供指标，帮助您估算实体识别器在您的作业中的工作效果。它们基于对识别器模型的训练，因此，虽然它们可以准确地表示模型在训练过程中的性能，但它们只是实体发现期间 API 性能的近似值。

每当从经过训练的实体识别器返回元数据时，都会返回指标。

Amazon Comprehend 支持一次在多达 25 个实体上训练模型。当训练有素的实体识别器返回指标时，将根据识别器整体（全局指标）和每个实体（实体指标）计算分数。

有三个指标可用，既有全局指标，也有实体指标：

- 精度

  这表示系统生成的实体中被正确识别和正确标记的比例。这显示了模型的实体识别有多少次是真正准确的识别。它是识别总数的百分比。

  换句话说，精度基于真阳性 (tp) 和假阳性 (fp)，其计算方式为精度 = tp/(tp + fp)。

  例如，如果模型预测文档中存在两个实体的示例，而实际上只有一个，则结果为一个真阳性和一个假阳性。在这种情况下，精度 = 1/(1 + 1)。精度为 50%，因为模型识别的两个实体中有一个是正确的。

- 召回率

  这表示文档中存在的实体被系统正确识别和标记的比例。从数学上讲，这是根据正确识别真阳性 (tp) 和未能识别假阴性 (fn) 的总数来定义的。

  计算方法为召回率 = tp/(tp + fn)。例如，如果模型正确识别了一个实体，但错过了该实体存在的另外两个实例，则结果为一个真阳性和两个假阴性。在这种情况下，召回 = 1/(1 + 2)。召回率为 33.33%，因为在可能的三个例子中，有一个实体是正确的。

- F1 分数

  这是精度和召回指标的组合，用于衡量自定义实体识别模型的整体准确性。F1 分数是精度和召回率指标的调和平均值：F1 = 2 * 精度 * 召回率/（精度 + 召回率）。

  > **Note**
  >
  > 直观上，调和平均值比简单平均值或其他均值更能惩罚极值（例如：`precision` = 0，`recall` = 1 可以通过预测所有可能的跨度来轻而易举地实现。在这里，简单平均值为 0.5，但 F1 会将其惩罚为0）。

  在上面的示例中，`precision` = 50% 和 `recall` = 33.33%，因此 F1 = 2 * 0.5 * 0.3333/(0.5 + 0.3333)。F1 分数为 0.3975，或 39.75%。

全局和单个实体指标

在分析以下句子时，可以看到全局实体指标和单个实体指标之间的关系，这些实体要么是地点，要么是人

```
John Washington and his friend Smith live in San Francisco, work in San Diego, and own
    a house in Seattle.
```

在我们的示例中，模型做出了以下预测。

```
John Washington = Person
Smith = Place
San Francisco = Place
```

```
San Diego = Place
Seattle = Person
```

但是，预测应该如下。

```
John Washington = Person
Smith = Person
San Francisco = Place
San Diego = Place
Seattle = Place
```

这方面的单个实体指标将是：

```
entity:  Person
  True positive (TP) = 1 (because John Washington is correctly predicted to be a
    Person).
  False positive (FP) = 1 (because Seattle is incorrectly predicted to be a Person,
    but is actually a Place).
  False negative (FN) = 1 (because Smith is incorrectly predicted to be a Place, but
    is actually a Person).
  Precision = 1 / (1 + 1) = 0.5 or 50%
  Recall = 1 / (1+1) = 0.5 or 50%
  F1 Score = 2 * 0.5 * 0.5 / (0.5 + 0.5) = 0.5 or 50%

entity:  Place
  TP = 2 (because San Francisco and San Diego are each correctly predicted to be a
    Place).
  FP = 1 (because Smith is incorrectly predicted to be a Place, but is actually a
    Person).
  FN = 1 (because Seattle is incorrectly predicted to be a Person, but is actually a
    Place).
  Precision = 2 / (2+1) = 0.6667 or 66.67%
  Recall = 2 / (2+1) = 0.6667 or 66.67%
  F1 Score = 2 * 0.6667 * 0.6667 / (0.6667 + 0.6667) = 0.6667 or  66.67%
```

这方面的全局指标将是：

全局：

```
Global:
  TP = 3 (because John Washington, San Francisco and San Diego are predicted
 correctly.
```

```
    This is also the sum of all individual entity TP).
  FP = 2 (because Seattle is predicted as Person and Smith is predicted as Place. This
    is the sum of all individual entity FP).
  FN = 2 (because Seattle is predicted as Person and Smith is predicted as Place. This
    is the sum of all individual FN).
  Global Precision = 3 / (3+2) = 0.6 or 60%
    (Global Precision = Global TP / (Global TP + Global FP))
  Global Recall = 3 / (3+2) = 0.6 or 60%
    (Global Recall = Global TP / (Global TP + Global FN))
  Global F1Score = 2 * 0.6 * 0.6 / (0.6 + 0.6) = 0.6 or 60%
    (Global F1Score = 2 * Global Precision *  Global Recall / (Global Precision +
    Global Recall))
```

提高自定义实体识别器的性能

这些指标可让您深入了解使用训练后的模型识别实体时的准确性。如果指标低于您的预期，您可以使用以下几个选项来改善这些指标：

1. 根据您使用的是 注释 或 实体列表（仅限纯文本），请务必遵循相应文档中的指南，以提高数据质量。如果您在改进数据并重新训练模型后观察到更好的指标，则可以不断迭代和提高数据质量，以实现更好的模型性能。
2. 如果您使用的是实体列表，请考虑改用注释。手动注释通常可以改善结果。
3. 如果您确定不存在数据质量问题，但指标仍然过低，请提交支持请求。

运行实时自定义识别器分析

实时分析对于在小文档到达时进行处理的应用程序非常有用。例如，您可以在社交媒体帖子、支持票证或客户评论中检测自定义实体。

> **开始之前**
>
> 在检测自定义实体之前，您需要使用自定义实体识别模型（也称为识别器）。有关这些模型的更多信息，请参阅 the section called “训练识别器模型”。

使用纯文本注释训练的识别器仅支持纯文本文档的实体检测。使用 PDF 文档注释训练的识别器支持纯文本文档、图像、PDF 文件和 Word 文档的实体检测。有关输入文件的信息，请参阅 实时自定义分析的输入。

如果您计划分析图像文件或扫描的 PDF 文档，则您的 IAM 策略必须授予使用两种 Amazon Textract API 方法（DetectDocumentText 和 AnalyzeDocument）的权限。Amazon Comprehend 在文本提取过程中会调用这些方法。有关策略示例，请参阅 执行文档分析操作所需的权。

主题

用于自定义实体识别的实时分析（控制台）

您可以使用 Amazon Comprehend 控制台通过自定义模型运行实时运行。首先，创建一个终端节点来运行实时分析。创建终端节点后，您可以运行实时分析。

有关配置终端节点吞吐量以及相关成本的信息，请参阅 使用 Amazon Comprehend 终端节点。

主题

为自定义实体检测创建终端节点

创建终端节点（控制台）

1. 登录 AWS Management Console 并打开 Amazon Comprehend 控制台，网址：https://console.aws.amazon.com/comprehend/
2. 从左侧菜单中选择终端节点，然后选择创建终端节点按钮。将打开创建终端节点屏幕。
3. 为终端节点命名。该名称在当前区域和账户中必须是唯一的。
4. 选择要将新终端节点附加到的自定义模型。在下拉列表中，您可以按模型名称进行搜索。

> **Note**
>
> 必须先创建模型，然后才能为其附加终端节点。如果您还没有模型，请参阅 训练自定义实体识别器模型。

5. （可选）要向终端节点添加标签，请在标签下输入键值对，然后选择添加标签。要在创建终端节点之前删除这对，请选择删除标签。
6. 输入要分配给终端节点的推理单元 (IU) 数量。每个单元表示每秒 100 个字符的吞吐量，每秒最多可处理 2 个文档。有关终端节点吞吐量的更多信息，请参阅 使用 Amazon Comprehend 终端节点。
7. （可选）如果您要创建新的终端节点，则可以选择使用 IU 估算器。估算器可以帮助您确定要请求的 IU 数量。推理单元的数量取决于吞吐量或每秒要分析的字符数。
8. 在购买摘要中，查看您预计的每小时、每日和每月终端节点成本。
9. 如果您知道您的账户会从终端节点启动到删除终端节点期间都会产生费用，请选中该复选框。
10. 选择创建端点。

运行实时自定义实体检测

为自定义实体识别器模型创建终端节点后，您可以运行实时分析来检测单个文档中的实体。

完成以下步骤以使用 Amazon Comprehend 控制台检测文本中的自定义实体。

1. 登录 AWS Management Console 并打开 Amazon Comprehend 控制台，网址：https://console.aws.amazon.com/comprehend/
2. 从左侧菜单中，选择实时分析。
3. 在输入文本部分的分析类型中，选择自定义。
4. 在选择终端节点中，选择与要使用的实体检测模型关联的终端节点。
5. 要指定用于分析的输入数据，您可以输入文本或上传文件。
   - 输入文本：
     a. 选择输入文本。
     b. 输入您要分析的文本。
   - 上传文件：
     a. 选择上传文件并输入要上传的文件名。
     b. （可选）在高级读取操作下，您可以覆盖文本提取的默认操作。有关更多信息，请参阅设置文本提取选项。
6. 选择分析。控制台显示分析结果以及置信度评估。

用于自定义实体识别的实时分析 (API)

您可以使用 Amazon Comprehend API 使用自定义模型运行实时分析。首先，创建一个终端节点来运行实时分析。创建终端节点后，您可以运行实时分析。

有关配置终端节点吞吐量以及相关成本的信息，请参阅 使用 Amazon Comprehend 终端节点。

主题

为自定义实体检测创建终端节点

有关与终端节点相关的成本的信息，请参阅 使用 Amazon Comprehend 终端节点。

使用 AWS CLI 创建终端节点

要使用 AWS CLI 创建终端节点，请使用 `create-endpoint` 命令：

```
$ aws comprehend create-endpoint \
> --desired-inference-units number of inference units \
> --endpoint-name endpoint name \
> --model-arn arn:aws:comprehend:region:account-id:model/example \
> --tags Key=Key,Value=Value
```

如果您的命令成功，Amazon Comprehend 将使用终端节点 ARN 进行响应：

```
{
   "EndpointArn": "Arn"
}
```

有关该命令、其参数配置以及其输出的更多信息，请参阅 AWS CLI 命令参考中的 `create-endpoint`。

运行实时自定义实体检测

为自定义实体识别器模型创建端点后，您可以使用该端点运行 DetectEntitiesAPI 操作。您可以使用 `text` 或 `bytes` 参数提供文本输入。使用 `bytes` 参数输入其他输入类型。

对于图像文件和 PDF 文件，您可以使用 `DocumentReaderConfig` 参数来覆盖默认的文本提取操作。有关更多信息，请参阅 设置文本提取选项。

使用 AWS CLI 检测文本中的实体

要检测文本中的自定义实体，请使用 `text` 参数中的输入文本运行 `detect-entities` 命令。

Example ：使用 CLI 检测输入文本中的实体

```
$ aws comprehend detect-entities \
> --endpoint-arn arn \
> --language-code en \
> --text  "Andy Jassy is the CEO of Amazon."
```

如果您的命令成功，Amazon Comprehend 将使用分析进行响应。对于 Amazon Comprehend 检测到的每个实体，它都会提供实体类型、文本、位置和置信度分数。

使用 AWS CLI 检测半结构化文档中的实体

要 PDF、Word 或图像文件中的自定义实体，请使用 `bytes` 参数中的输入文本运行 `detect-entities` 命令。

Example ：使用 CLI 检测图像文件中的实体

此示例说明如何使用 `fileb` 选项对图像字节进行 base64 编码，从而传入图像文件。有关更多信息，请参阅《AWS Command Line Interface 用户指南》中的二进制大型对象。

此示例还传入名为 `config.json` 的 JSON 文件以设置文本提取选项。

```
$ aws comprehend detect-entities \
> --endpoint-arn arn \
> --language-code en \
> --bytes fileb://image1.jpg   \
> --document-reader-config file://config.json
```

config.json 文件包含以下代码。

```
{
   "DocumentReadMode": "FORCE_DOCUMENT_READ_ACTION",
   "DocumentReadAction": "TEXTRACT_DETECT_DOCUMENT_TEXT"
}
```

有关命令语法的更多信息，请参阅《Amazon Comprehend API 参考》DetectEntities中的。

实时分析输出

文本输入的输出

如果使用 Text 参数输入文本，输出将由分析检测到的实体数组组成。以下示例显示了检测到两个 JUDGE 实体的分析。

```
{
        "Entities":
        [
            {
                "BeginOffset": 0,
                "EndOffset": 22,
                "Score": 0.9763959646224976,
                "Text": "John Johnson",
                "Type": "JUDGE"
            },
            {
                "BeginOffset": 11,
                "EndOffset": 15,
                "Score": 0.9615424871444702,
                "Text": "Thomas Kincaid",
                "Type": "JUDGE"
            }
        ]
    }
```

半结构化输入的输出

对于半结构化输入文档或文本文件，输出可以包括以下附加字段：

- DocumentMetadata — 提取有关文档的信息。元数据包括文档中的页面列表，以及从每页中提取的字符数。如果请求包含 Byte 参数，则响应中会显示此字段。
- DocumentType -输入文档中每页的文档类型。此字段出现在包含该 Byte 参数的请求的响应中。
- 块：有关输入文档中每个文本块的信息。块是嵌套的。页面块包含每行文本的块，其中每个单词都包含一个块。此字段出现在包含该 Byte 参数的请求的响应中。
- BlockReferences — 对该实体的每个区块的引用。此字段出现在包含该 Byte 参数的请求的响应中。文本文件中不存在该字段。

- 错误：系统在处理输入文档时检测到的页面级错误。如果系统未遇到任何错误，则该字段为空。

有关这些输出字段的描述，请参阅DetectEntities《Amazon Comprehend API 参考》。有关布局元素的更多信息，请参阅《Amazon Textract 开发人员指南》中的 Amazon Textract 分析对象。

以下示例显示了单页扫描的 PDF 输入文档的输出。


```
{
    "Entities": [{
        "Score": 0.9984670877456665,
        "Type": "DATE-TIME",
        "Text": "September 4,",
        "BlockReferences": [{
            "BlockId": "42dcaaee-c484-4b5d-9e3f-ae0be928b3e1",
            "BeginOffset": 0,
            "EndOffset": 12,
            "ChildBlocks": [{
                    "ChildBlockId": "6e9cbb43-f8be-4da0-9a4b-ff9a6c350a14",
                    "BeginOffset": 0,
                    "EndOffset": 9
                },
                {
                    "ChildBlockId": "599e0d53-ae9f-491b-a762-459b22c79ff5",
                    "BeginOffset": 0,
                    "EndOffset": 2
                },
                {
                    "ChildBlockId": "599e0d53-ae9f-491b-a762-459b22c79ff5",
                    "BeginOffset": 0,
                    "EndOffset": 2
                }
            ]
        }]
    }],
    "DocumentMetadata": {
        "Pages": 1,
        "ExtractedCharacters": [{
            "Page": 1,
            "Count": 609
        }]
    },
    "DocumentType": [{
        "Page": 1,
```

```
        "Type": "SCANNED_PDF"
    }],
    "Blocks": [{
        "Id": "ee82edf3-28de-4d63-8883-40e2e4938ccb",
        "BlockType": "LINE",
        "Text": "Your Band",
        "Page": 1,
        "Geometry": {
            "BoundingBox": {
                "Height": 0.024125460535287857,
                "Left": 0.11745482683181763,
                "Top": 0.06821706146001816,
                "Width": 0.12074867635965347
            },
            "Polygon": [{
                    "X": 0.11745482683181763,
                    "Y": 0.06821706146001816
                },
                {
                    "X": 0.2382034957408905,
                    "Y": 0.06821706146001816
                },
                {
                    "X": 0.2382034957408905,
                    "Y": 0.09234252572059631
                },
                {
                    "X": 0.11745482683181763,
                    "Y": 0.09234252572059631
                }
            ]
        },
        "Relationships": [{
            "Ids": [
                "b105c561-c8d9-485a-a728-7a5b1a308935",
                "60ecb119-3173-4de2-8c5d-de182a5f86a5"
            ],
            "Type": "CHILD"
        }]
    }]
}
```


以下示例显示了用于分析原生 PDF 文档的输出。

Example PDF 文档自定义实体识别分析的输出示例


```
{
        "Blocks":
        [
            {
                "BlockType": "LINE",
                "Geometry":
                {
                    "BoundingBox":
                    {
                        "Height": 0.012575757575757575,
                        "Left": 0.0,
                        "Top": 0.0015063131313131314,
                        "Width": 0.02262091503267974
                    },
                    "Polygon":
                    [
                        {
                            "X": 0.0,
                            "Y": 0.0015063131313131314
                        },
                        {
                            "X": 0.02262091503267974,
                            "Y": 0.0015063131313131314
                        },
                        {
                            "X": 0.02262091503267974,
                            "Y": 0.014082070707070706
                        },
                        {
                            "X": 0.0,
                            "Y": 0.014082070707070706
                        }
                    ]
                },
                "Id": "4330efed-6334-4fc4-ba48-e050afa95c8d",
                "Page": 1,
                "Relationships":
                [
                    {
                        "ids":
                        [
                            "f343ce48-583d-4abe-b84b-a232e266450f"
```

```
                    ],
                    "type": "CHILD"
                }
            ],
            "Text": "S-3"
        },
        {
            "BlockType": "WORD",
            "Geometry":
            {
                "BoundingBox":
                {
                    "Height": 0.012575757575757575,
                    "Left": 0.0,
                    "Top": 0.0015063131313131314,
                    "Width": 0.02262091503267974
                },
                "Polygon":
                [
                    {
                        "X": 0.0,
                        "Y": 0.0015063131313131314
                    },
                    {
                        "X": 0.02262091503267974,
                        "Y": 0.0015063131313131314
                    },
                    {
                        "X": 0.02262091503267974,
                        "Y": 0.014082070707070706
                    },
                    {
                        "X": 0.0,
                        "Y": 0.014082070707070706
                    }
                ]
            },
            "Id": "f343ce48-583d-4abe-b84b-a232e266450f",
            "Page": 1,
            "Relationships":
            [],
            "Text": "S-3"
        }
    ],
```

```
        "DocumentMetadata":
        {
            "PageNumber": 1,
            "Pages": 1
        },
        "DocumentType": "NativePDF",
        "Entities":
        [
            {
                "BlockReferences":
                [
                    {
                        "BeginOffset": 25,
                        "BlockId": "4330efed-6334-4fc4-ba48-e050afa95c8d",
                        "ChildBlocks":
                        [
                            {
                                "BeginOffset": 1,
                                "ChildBlockId": "cbba5534-ac69-4bc4-beef-306c659f70a6",
                                "EndOffset": 6
                            }
                        ],
                        "EndOffset": 30
                    }
                ],
                "Score": 0.9998825926329088,
                "Text": "0.001",
                "Type": "OFFERING_PRICE"
            },
            {
                "BlockReferences":
                [
                    {
                        "BeginOffset": 41,
                        "BlockId": "f343ce48-583d-4abe-b84b-a232e266450f",
                        "ChildBlocks":
                        [
                            {
                                "BeginOffset": 0,
                                "ChildBlockId": "292a2e26-21f0-401b-a2bf-03aa4c47f787",
                                "EndOffset": 9
                            }
                        ],
                        "EndOffset": 50
```

```
                }
            ],
            "Score": 0.9809727537330395,
            "Text": "6,097,560",
            "Type": "OFFERED_SHARES"
        }
    ],
    "File": "example.pdf",
    "Version": "2021-04-30"
}
```

运行分析作业以识别自定义实体

您可以运行异步分析作业来检测一组或多个文档中的自定义实体。

> **开始之前**
>
> 在检测自定义实体之前，您需要使用自定义实体识别模型（也称为识别器）。有关这些模型的更多信息，请参阅 the section called “训练识别器模型”。

使用纯文本注释训练的识别器仅支持纯文本文档的实体检测。使用 PDF 文档注释训练的识别器支持纯文本文档、图像、PDF 文件和 Word 文档的实体检测。对于文本文件以外的文件，Amazon Comprehend 会在运行分析之前执行文本提取。有关输入文件的信息，请参阅 异步自定义分析的输入。

如果您计划分析图像文件或扫描的 PDF 文档，则您的 IAM 策略必须授予使用两种 Amazon Textract API 方法（DetectDocumentText 和 AnalyzeDocument）的权限。Amazon Comprehend 在文本提取过程中会调用这些方法。有关策略示例，请参阅 执行文档分析操作所需的权。

要运行异步分析作业，请执行以下总体步骤：

1. 将这些文档存储在 Amazon S3 存储桶中。
2. 使用 API 或控制台启动分析作业。
3. 监控分析作业的进度。
4. 作业运行完成后，从启动作业时指定的 S3 存储桶中检索分析结果。

主题

启动自定义实体检测作业（控制台）

您可以使用控制台启动和监控异步分析作业，以进行自定义实体识别。

启动异步分析作业

1. 登录 AWS Management Console 并打开 Amazon Comprehend 控制台，网址：https://console.aws.amazon.com/comprehend/
2. 从左侧菜单中选择分析作业，然后选择创建作业。
3. 为分类作业命名。该名称在您的账户和当前区域中都必须是唯一的。
4. 在分析类型下，选择自定义实体识别。
5. 从识别器模型中，选择要使用的自定义实体识别器。
6. 从版本中，选择要使用的识别器版本。
7. （可选）如果您选择对 Amazon Comprehend 在处理作业时使用的数据进行加密，请选择作业加密。然后选择是使用与当前账户关联的 KMS 密钥，还是使用来自其他账户的密钥。
   - 如果您使用的是与当前账户关联的密钥，请为 KMS 密钥 ID 选择密钥 ID。
   - 如果您使用与其他账户关联的密钥，请在 KMS 密钥 ARN 下输入密钥 ID 的 ARN。

   > **Note**
   >
   > 有关创建和使用 KMS 密钥以及相关加密的更多信息，请参阅密钥管理服务 (KMS)。

8. 在输入数据下，输入包含您的输入文档的 Amazon S3 存储桶的位置，或者选择浏览 S3 导航到该存储桶。存储桶必须与所调用的 API 位于同一区域。用于获取分析作业访问权限的 IAM 角色必须具有 S3 存储桶的读取权限。
9. （可选）对于输入格式，您可以选择输入文档的格式。格式可以是每个文件一个文档，也可以是单个文件中每行一个文档。每行一个文档仅适用于文本文档。
10. （可选）对于文档读取模式，您可以覆盖默认的文本提取操作。有关更多信息，请参阅 设置文本提取选项。

11. 在输出数据下，输入 Amazon Comprehend 应写入作业输出数据的 Amazon S3 存储桶的位置，或者选择浏览 S3 导航到该存储桶。存储桶必须与所调用的 API 位于同一区域。用于获取分类作业访问权限的 IAM 角色必须具有 S3 存储桶的写入权限。
12. （可选）如果要加密作业的输出结果，请选择加密。然后选择是使用与当前账户关联的 KMS 密钥，还是使用来自其他账户的密钥。
    - 如果您使用的是与当前账户关联的密钥，请为 KMS 密钥 ID 选择密钥别名或 ID。
    - 如果您使用与其他账户关联的密钥，请在 KMS 密钥 ID 下输入密钥别名或 ID 的 ARN。
13. （可选）要将您的资源从 VPC 启动到 Amazon Comprehend，请在 VPC 下输入 VPC ID 或从下拉列表中选择 ID。
    1. 在子网下选择子网。选择第一个子网后，您还可以选择其他子网。
    2. 在安全组下，选择要使用的安全组（如果已指定）。选择第一个安全组后，您还可以选择其他安全组。

    > **Note**
    >
    > 当您在分析作业中使用 VPC 时，用于创建和启动操作的 `DataAccessRole` 必须具有访问输出存储桶的 VPC 权限。

14. 选择创建作业以创建实体识别作业。

启动自定义实体检测作业 (API)

您可以使用 API 启动和监控异步分析作业，以进行自定义实体识别。

要使用该StartEntitiesDetectionJob操作启动自定义实体检测任务，请提供 EntityRecognizerArn，即训练模型的 Amazon 资源名称 (ARN)。你可以在对操作的响应中找到此 ARN。CreateEntityRecognizer

主题

- 使用检测自定义实体 AWS Command Line Interface
- 使用 AWS SDK for Java检测自定义实体
- 使用检测自定义实体 AWS SDK for Python (Boto3)
- 覆盖 PDF 文件的 API 操作

使用检测自定义实体 AWS Command Line Interface

以下示例适用 Unix、Linux 和 macOS 环境。对于 Windows，请将每行末尾的反斜杠 (\) Unix 行继续符替换为脱字号 (^)。要检测文档集中的自定义实体，请使用以下请求语法：

```
aws comprehend start-entities-detection-job \
     --entity-recognizer-arn "arn:aws:comprehend:region:account number:entity-
recognizer/test-6" \
     --job-name infer-1 \
     --data-access-role-arn "arn:aws:iam::account number:role/service-role/
AmazonComprehendServiceRole-role" \
     --language-code en \
     --input-data-config "S3Uri=s3://Bucket Name/Bucket Path" \
     --output-data-config "S3Uri=s3://Bucket Name/Bucket Path/" \
     --region region
```

Amazon Comprehend 使用 `JobID` 和 `JobStatus` 作为响应，并将返回您在请求中指定的 S3 存储桶中作业的输出。

使用 AWS SDK for Java检测自定义实体

有关使用 Java 的 Amazon Comprehend 示例，请参阅 Amazon Comprehend Java 示例。

使用检测自定义实体 AWS SDK for Python (Boto3)

此示例创建自定义实体识别器，训练模型，然后使用 AWS SDK for Python (Boto3)在实体识别器作业中运行该模型。

实例化适用于 Python 的 SDK。

```
import boto3
import uuid
comprehend = boto3.client("comprehend", region_name="region")
```

创建实体识别器：

```
response = comprehend.create_entity_recognizer(
    RecognizerName="Recognizer-Name-Goes-Here-{}".format(str(uuid.uuid4())),
    LanguageCode="en",
    DataAccessRoleArn="Role ARN",
    InputDataConfig={
        "EntityTypes": [
```

```
            {
                "Type": "ENTITY_TYPE"
            }
        ],
        "Documents": {
            "S3Uri": "s3://Bucket Name/Bucket Path/documents"
        },
        "Annotations": {
            "S3Uri": "s3://Bucket Name/Bucket Path/annotations"
        }
    }
)
recognizer_arn = response["EntityRecognizerArn"]
```

列出所有识别器：

```
response = comprehend.list_entity_recognizers()
```

等待实体识别器达到“已训练”状态：

```
while True:
    response = comprehend.describe_entity_recognizer(
        EntityRecognizerArn=recognizer_arn
    )

    status = response["EntityRecognizerProperties"]["Status"]
    if "IN_ERROR" == status:
        sys.exit(1)
    if "TRAINED" == status:
        break

    time.sleep(10)
```

启动自定义实体检测作业：

```
response = comprehend.start_entities_detection_job(
    EntityRecognizerArn=recognizer_arn,
    JobName="Detection-Job-Name-{}".format(str(uuid.uuid4())),
    LanguageCode="en",
    DataAccessRoleArn="Role ARN",
    InputDataConfig={
        "InputFormat": "ONE_DOC_PER_LINE",
```

```
        "S3Uri": "s3://Bucket Name/Bucket Path/documents"
    },
    OutputDataConfig={
        "S3Uri": "s3://Bucket Name/Bucket Path/output"
    }
)
```

覆盖 PDF 文件的 API 操作

对于图像文件和 PDF 文件，您可以使用 InputDataConfig 中的 DocumentReaderConfig 参数覆盖默认的提取操作。

以下示例定义了一个名为 myInputData config.json 的 JSON 文件来设置这些InputDataConfig值。它将 DocumentReadConfig 设置为对所有 PDF 文件使用 Amazon Textract DetectDocumentText API。

Example

```
"InputDataConfig": {
  "S3Uri": s3://Bucket Name/Bucket Path",
  "InputFormat": "ONE_DOC_PER_FILE",
  "DocumentReaderConfig": {
      "DocumentReadAction": "TEXTRACT_DETECT_DOCUMENT_TEXT",
      "DocumentReadMode": "FORCE_DOCUMENT_READ_ACTION"
  }
}
```

在StartEntitiesDetectionJob操作中，将 myInputData config.json 文件指定为参数：InputDataConfig

```
  --input-data-config file://myInputDataConfig.json
```

有关 DocumentReaderConfig 参数的更多信息，请参阅 设置文本提取选项。

异步分析作业的输出

分析作业完成后，它将结果存储到您在请求中指定的 S3 存储桶中。

文本输入的输出

对于文本输入文件，输出由每个输入文档的实体列表组成。

以下示例显示了名为 50_docs 的输入文件中两个文档的输出，使用每行一个文档的格式。


```
{
        "File": "50_docs",
        "Line": 0,
        "Entities":
        [
            {
                "BeginOffset": 0,
                "EndOffset": 22,
                "Score": 0.9763959646224976,
                "Text": "John Johnson",
                "Type": "JUDGE"
            }
        ]
    }
    {
        "File": "50_docs",
        "Line": 1,
        "Entities":
        [
            {
                "BeginOffset": 11,
                "EndOffset": 15,
                "Score": 0.9615424871444702,
                "Text": "Thomas Kincaid",
                "Type": "JUDGE"
            }
        ]
    }
```


半结构化输入的输出

对于半结构化输入文档，输出可以包括以下附加字段：

- DocumentMetadata — 提取有关文档的信息。元数据包括文档中的页面列表，以及从每页中提取的字符数。如果请求包含 Byte 参数，则响应中会显示此字段。
- DocumentType -输入文档中每页的文档类型。此字段出现在包含该 Byte 参数的请求的响应中。
- 块：有关输入文档中每个文本块的信息。块可以嵌套在块内。页面块包含每行文本的块，其中每个单词都包含一个块。此字段出现在包含该 Byte 参数的请求的响应中。

- BlockReferences — 对该实体的每个区块的引用。此字段出现在包含该 Byte 参数的请求的响应中。文本文件中不存在该字段。
- 错误：系统在处理输入文档时检测到的页面级错误。如果系统未遇到任何错误，则该字段为空。

有关这些输出字段的更多详细信息，请参阅亚马逊 Comprehend API 参考DetectEntities中的

以下示例显示一个单页原生 PDF 输入文档的输出。

Example PDF 文档自定义实体识别分析的输出示例


```
{
        "Blocks":
        [
            {
                "BlockType": "LINE",
                "Geometry":
                {
                    "BoundingBox":
                    {
                        "Height": 0.012575757575757575,
                        "Left": 0.0,
                        "Top": 0.0015063131313131314,
                        "Width": 0.02262091503267974
                    },
                    "Polygon":
                    [
                        {
                            "X": 0.0,
                            "Y": 0.0015063131313131314
                        },
                        {
                            "X": 0.02262091503267974,
                            "Y": 0.0015063131313131314
                        },
                        {
                            "X": 0.02262091503267974,
                            "Y": 0.014082070707070706
                        },
                        {
                            "X": 0.0,
                            "Y": 0.014082070707070706
                        }
```

```
                    ]
                },
                "Id": "4330efed-6334-4fc4-ba48-e050afa95c8d",
                "Page": 1,
                "Relationships":
                [
                    {
                        "ids":
                        [
                            "f343ce48-583d-4abe-b84b-a232e266450f"
                        ],
                        "type": "CHILD"
                    }
                ],
                "Text": "S-3"
            },
            {
                "BlockType": "WORD",
                "Geometry":
                {
                    "BoundingBox":
                    {
                        "Height": 0.012575757575757575,
                        "Left": 0.0,
                        "Top": 0.0015063131313131314,
                        "Width": 0.02262091503267974
                    },
                    "Polygon":
                    [
                        {
                            "X": 0.0,
                            "Y": 0.0015063131313131314
                        },
                        {
                            "X": 0.02262091503267974,
                            "Y": 0.0015063131313131314
                        },
                        {
                            "X": 0.02262091503267974,
                            "Y": 0.014082070707070706
                        },
                        {
                            "X": 0.0,
                            "Y": 0.014082070707070706
```

```
                    }
                ]
            },
            "Id": "f343ce48-583d-4abe-b84b-a232e266450f",
            "Page": 1,
            "Relationships":
            [],
            "Text": "S-3"
        }
    ],
    "DocumentMetadata":
    {
        "PageNumber": 1,
        "Pages": 1
    },
    "DocumentType": "NativePDF",
    "Entities":
    [
        {
            "BlockReferences":
            [
                {
                    "BeginOffset": 25,
                    "BlockId": "4330efed-6334-4fc4-ba48-e050afa95c8d",
                    "ChildBlocks":
                    [
                        {
                            "BeginOffset": 1,
                            "ChildBlockId": "cbba5534-ac69-4bc4-beef-306c659f70a6",
                            "EndOffset": 6
                        }
                    ],
                    "EndOffset": 30
                }
            ],
            "Score": 0.9998825926329088,
            "Text": "0.001",
            "Type": "OFFERING_PRICE"
        },
        {
            "BlockReferences":
            [
                {
                    "BeginOffset": 41,
```

```
                    "BlockId": "f343ce48-583d-4abe-b84b-a232e266450f",
                    "ChildBlocks":
                    [
                        {
                            "BeginOffset": 0,
                            "ChildBlockId": "292a2e26-21f0-401b-a2bf-03aa4c47f787",
                            "EndOffset": 9
                        }
                    ],
                    "EndOffset": 50
                }
            ],
            "Score": 0.9809727537330395,
            "Text": "6,097,560",
            "Type": "OFFERED_SHARES"
        }
    ],
    "File": "example.pdf",
    "Version": "2021-04-30"
}
```

创建和管理自定义模型

Amazon Comprehend 包含内置的 NLP（自然语言处理）模型，可用于分析见解或主题建模。您可以使用 Amazon Comprehend 创建用于实体识别和文档分类的自定义模型。

您可以使用模型版本控制来跟踪模型的历史记录。创建和训练新的模型版本时，可以对训练数据集进行更改。Amazon Comprehend 在模型详情页面上显示每个型号版本的详细信息（包括模型性能）。随着时间的推移，您可以看到对训练数据集进行更改时模型性能的变化。

您可以使用 Amazon Comprehend 控制台或 API 创建模型版本。作为替代方案，Amazon Comprehend 提供 飞轮 以简化与训练和评估新的自定义模型版本相关的作业。

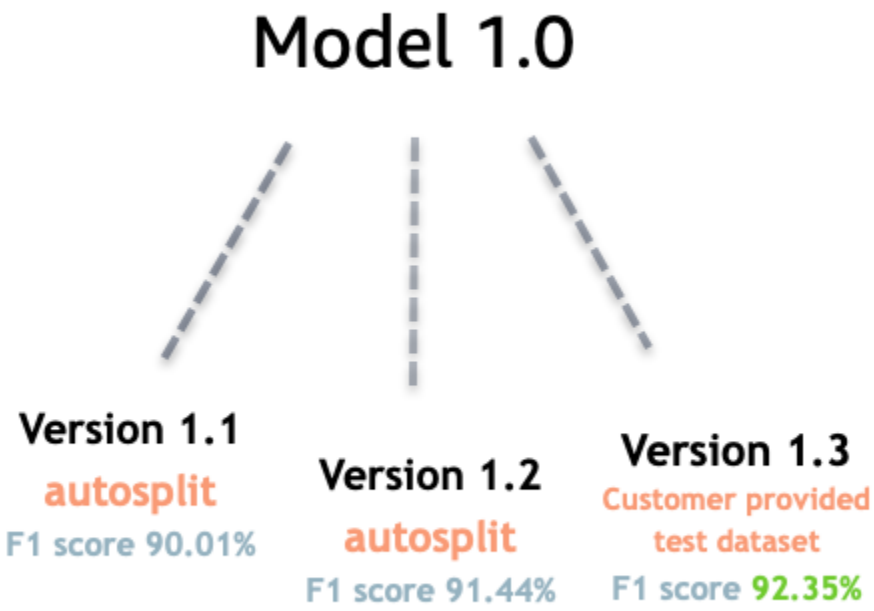
创建自定义模型后，您可以允许其他 AWS 账户 导入您的模型副本，从而与其他用户共享该模型。

主题

使用 Amazon Comprehend 进行模型版本控制

人工智能和机器学习 (AI/ML) 都是关于快速实验的。借助 Amazon Comprehend，您可以训练和构建模型，用于深入了解数据。通过模型版本控制，您可以在提供更多或不同的数据集时,跟踪与模型运行结果相关的建模历史和分数。您可以对自定义分类模型或自定义实体识别模型使用版本控制。随着时间的推移，查看您的不同版本，您可以深入了解它们的成功表现，并获得关于您用来达到成功状态的参数的见解。

训练现有自定义分类器模型或实体识别模型的新版本时，您所需要做的就是从模型详细信息页面创建一个新版本，然后为您填充所有详细信息。新版本将与您之前的模型同名（我们称之为版本 ID），尽管您在创建时会为其指定一个唯一的版本名称。向模型添加新版本时，您可以从模型详细信息页面的一个视图中查看所有先前版本及其详细信息。通过版本控制，您可以看到对训练数据集进行更改时模型性能的变化。



创建新的自定义分类器版本（控制台）

1. 登录 AWS Management Console 并打开亚马逊 Comprehend 控制台，网址为 https://console.aws.amazon.com/comprehend/
2. 从左侧菜单中选择自定义，然后选择自定义分类。
3. 从分类器列表中，选择要向其创建新版本的自定义模型的名称。显示自定义模型的详细信息页面。
4. 在右上角，选择创建新模型。屏幕将打开，其中包含来自父自定义分类模型的预填充详细信息。
5. 在版本名称下为新版本添加一个唯一的名称。
6. 在版本详细信息下，您可以更改与新模型关联的标签语言和数量。
7. 在数据规范部分下，配置如何向新版本提供数据，确保提供完整数据，包括以前模型中的文档和新文档。您可以更改分类器模式（单标签或多标签）、数据格式（CSV 文件、增强清单）、训练数据集和测试数据集（自动拆分或自定义测试数据配置）。
8. （可选）更新输出数据的 S3 位置
9. 在访问权限下，创建或使用现有的 IAM 角色。
10. （可选）更新您的 VPC 设置
11. （可选）向新版本添加标签，以帮助跟踪详细信息。

有关创建自定义分类器的更多信息，请参阅创建自定义分类器

创建新的自定义实体识别器版本（控制台）

1. 登录 AWS Management Console 并打开亚马逊 Comprehend 控制台，网址为 https://console.aws.amazon.com/comprehend/
2. 从左侧菜单中选择自定义，然后选择自定义实体识别。
3. 从识别器模型列表中，选择要向其创建新版本的识别器的名称。显示详细信息页面。
4. 在右上角，选择训练新版本。屏幕将打开，其中包含来自父实体识别器的预填充详细信息。
5. 在版本名称下为新版本添加一个唯一的名称。
6. 在自定义实体类型下，添加您希望识别器在数据集中识别的自定义标签或标注，然后选择添加类型。从您提供的注释或实体列表中选择自定义实体类型。然后，识别器将在运行作业时使用所有包含的实体类型来识别数据集中的实体。如果每个实体类型使用多个单词，则必须使用大写字母并用下划线分隔。最多允许 25 种类型。
7. （可选）选择识别器加密可在处理作业时对存储卷中的数据进行加密。
8. 在训练数据部分下，指定注释和数据格式详细信息（CSV 文件、增强清单、单标签或多标签）、数据格式（CSV、增强清单）、您的训练数据集和测试数据集（自动拆分或您的自定义测试数据配置）。
9. （可选）更新输出数据的 S3 位置
10. 在访问权限下，创建或使用现有的 IAM 角色。
11. （可选）更新您的 VPC 设置
12. （可选）向新版本添加标签，以帮助跟踪详细信息。

要了解有关自定义实体识别器的更多信息，请参阅自定义实体识别和使用控制台创建自定义实体识别器。

在 AWS 账户 之间复制自定义模型

Amazon Comprehend 用户可以通过两步过程，在 AWS 账户 之间复制经过训练的自定义模型。首先，一个 AWS 账户（账户 A）中的用户共享其账户中的自定义模型。然后，另一个 AWS 账户（账户 B）中的用户将模型导入他们的账户中。账户 B 用户无需训练模型，也不需要复制（或访问）原始训练数据或测试数据。

要在账户 A 中共享自定义模型，用户需要将一个 AWS Identity and Access Management (IAM) policy 附加到模型版本。该策略授权账户 B 中的实体（例如用户或角色）将模型版本导入其 AWS 账户 中的 Amazon Comprehend。账户 B 用户必须将模型导入到与原始模型相同的 AWS 区域 中。

要将模型导入账户 B 中，则此账户的用户需要向 Amazon Comprehend 提供必要的详细信息，例如模型的 Amazon 资源名称（ARN)。通过导入模型，该用户可以在其 AWS 账户 中创建一个新的自定义模型，该模型将复制他们导入的模型。该模型经过全面训练，可以用于推理工作，例如文档分类或命名实体识别。

在以下情况下，复制自定义模型很有用：

- 您所属的组织使用多个 AWS 账户。例如，您的组织可能在每个开发阶段（例如构建、阶段、测试和部署）都拥有一个 AWS 账户。或者，它可能具有用于业务功能的不同的 AWS 账户，例如数据科学和工程。
- 您的组织与其他组织（例如 AWS 合作伙伴）合作，后者在 Amazon Comprehend 中训练自定义模型，并将其作为客户提供给您。

在这样的场景中，您可以快速将经过训练的自定义实体识别器或文档分类器从一个 AWS 账户 复制到另一个账户中。用这种方式复制模型比另一种方法更容易，在另一种方法中，您可以在 AWS 账户 之间复制训练数据以训练重复的模型。

主题

与其他人共享自定义模型 AWS 账户

借助 Amazon Comprehend，您可以与其他人共享您的自定义模型，这样他们就可以将您的模型导入他们的 AWS 账户。当用户导入您的一个自定义模型时，他们会在自己的账户中创建一个新的自定义模型。他们的新模型与您共享的模型相同。

要共享自定义模型，您需要为其附加一个授权他人导入该模型的策略。然后，您可以向这些用户提供他们需要的详细信息。

> **Note**
>
> 当其他用户导入您共享的自定义模型时，他们必须使用包含您的模型的相同 AWS 区域，例如。美国东部（弗吉尼亚州北部）。

主题

开始之前

在共享模型之前，您必须在 AWS 账户 中的 Amazon Comprehend 中拥有经过训练的自定义分类器或自定义实体识别器。有关训练自定义模型的更多信息，请参阅自定义分类 或 自定义实体识别。

所需的权限

IAM policy 语句

在向自定义模型添加基于资源的策略之前，您需要在 AWS Identity and Access Management (IAM) 中获得权限。您的用户、组或角色必须附加策略，这样您才能创建、获取和删除模型策略，如以下示例所示。

Example 用于管理自定义模型的基于资源的策略的 IAM policy

```
{
  "Effect": "Allow",
  "Action": [
    "comprehend:PutResourcePolicy",
    "comprehend:DeleteResourcePolicy",
    "comprehend:DescribeResourcePolicy"
  ],
  "Resource": "arn:aws:comprehend:us-west-2:111122223333:document-classifier/foo/
version/*"
}
```

有关创建 IAM policy 的更多信息，请参阅《IAM 用户指南》中的创建 IAM policy。有关附加 IAM policy 的信息，请参阅《IAM 用户指南》中的添加和删除 IAM 身份权限。

AWS KMS 密钥策略语句

如果您要共享加密模型，则可能需要为 AWS KMS 添加权限。此要求取决于您在 Amazon Comprehend 中用于加密模型的 KMS 密钥的类型。

AWS 拥有的密钥 由 AWS 服务拥有和管理。如果您使用 AWS 拥有的密钥，则无需为 AWS KMS 添加权限，并且可以跳过本节。

客户自主管理型密钥是在您的 AWS 账户 中创建、拥有和管理的密钥。若要使用客户自主管理型密钥，您必须向 KMS 密钥策略添加一条语句。

该策略语句授权一个或多个实体（例如用户或账户）执行解密模型所需的 AWS KMS 操作。

您可以使用条件键来帮助防止混淆代理问题。有关更多信息，请参阅 the section called “防止跨服务混淆座席”。

在策略中使用以下条件密钥来验证访问您的 KMS 密钥的实体。当用户导入模型时，AWS KMS 会检查源模型版本的 ARN 是否符合条件。如果您未在策略中包含条件，则指定的委托人可以使用您的 KMS 密钥解密任何模型版本：

- a@@ ws: SourceArn — 将此条件键与`kms:GenerateDataKey`和`kms:Decrypt`操作一起使用。
- kms: EncryptionContext — 将此条件密钥与`kms:GenerateDataKey`、`kms:Decrypt`、和`kms:CreateGrant`操作配合使用。

在以下示例中，策略授权 AWS 账户 444455556666 使用 AWS 账户 111122223333 所拥有的指定分类器模型的版本 1。

Example 用于访问特定分类器模型版本的 KMS 密钥策略

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
          "AWS":
                "arn:aws:iam::444455556666:root"
      },
      "Action": [
```

```
            "kms:Decrypt",
            "kms:GenerateDataKey"
        ],
        "Resource": "*",
        "Condition": {
            "StringEquals": {
                "aws:SourceArn":
                  "arn:aws:comprehend:us-west-2:111122223333:document-
classifier/classifierName/version/1"
            }
        }
    },
    {
        "Effect": "Allow",
        "Principal": {
            "AWS":  "arn:aws:iam::444455556666:root"
        },
        "Action": "kms:CreateGrant",
        "Resource": "*",
        "Condition": {
          "StringEquals": {
             "kms:EncryptionContext:aws:comprehend:arn":
                "arn:aws:comprehend:us-west-2:111122223333:document-
classifier/classifierName/version/1"
          }
        }
    }
  ]
}
```

以下示例策略授权用户ExampleUser 通过 Amazon C AWS 账户 444455556666 om ExampleRolep AWS 账户 123456789012 rehend 服务访问此 KMS 密钥。

Example 允许访问 Amazon Comprehend 服务的 KMS 密钥策略（替代方案 1）。

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
        "Effect": "Allow",
        "Principal": {
            "AWS": [
                "arn:aws:iam::444455556666:user/ExampleUser",
```

```
                "arn:aws:iam::123456789012:role/ExampleRole"
              ]
        },
        "Action": [
            "kms:Decrypt",
            "kms:GenerateDataKey"
        ],
        "Resource": "*",
        "Condition": {
            "StringLike": {
                "aws:SourceArn": "arn:aws:comprehend:*"

            }
        }
    },
    {
      "Effect": "Allow",
      "Principal": {
         "AWS": [
                "arn:aws:iam::444455556666:user/ExampleUser",
                "arn:aws:iam::123456789012:role/ExampleRole"
              ]
      },
      "Action": "kms:CreateGrant",
      "Resource": "*",
      "Condition": {
          "StringLike": {
              "kms:EncryptionContext:aws:comprehend:arn": "arn:aws:comprehend:*"
          }
      }
    }
  ]
}
```

以下示例策略授权 AWS 账户 444455556666 通过 Amazon Comprehend 服务访问此 KMS 密钥，使用前一个示例的替代语法。

Example 允许访问 Amazon Comprehend 服务的 KMS 密钥策略（替代方案 2）。

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

```
      "Effect": "Allow",
      "Principal": {
          "AWS": "arn:aws:iam::444455556666:root"
      },
      "Action": [
          "kms:Decrypt",
          "kms:GenerateDataKey",
          "kms:CreateGrant"
      ],
      "Resource": "*",
      "Condition": {
          "StringLike": {
              "kms:EncryptionContext:aws:comprehend:arn": "arn:aws:comprehend:*"
          }
      }
    }
  ]
}
```

有关更多信息，请参阅《AWS Key Management Service 开发人员指南》中的在 AWS KMS 中使用密钥策略。

自定义模型的基于资源的策略

在其他 AWS 账户 的 Amazon Comprehend 用户可以从您的 AWS 账户导入自定义模型之前，您必须授权他们这样做。要对它们进行授权，请将基于资源的策略添加到您要共享的模型版本中。在基于资源的策略是附加到 AWS 中的资源的 IAM policy。

当您将资源策略附加到自定义模型版本时，该策略会授权用户、组或角色对模型版本执行 `comprehend:ImportModel` 操作。

Example 自定义模型版本的基于资源的策略

此示例在 `Principal` 属性中指定了授权实体。资源“*”是指您附加策略的特定模型版本。

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": "comprehend:ImportModel",
      "Resource": "*",
      "Principal": {
```

```
        "AWS": [
                "arn:aws:iam::111122223333:root",
                "arn:aws:iam::444455556666:user/ExampleUser",
                "arn:aws:iam::123456789012:role/ExampleRole"
          ]
       }
     }
  ]
}
```

对于您附加到自定义模型的策略，`comprehend:ImportModel` 是 Amazon Comprehend 支持的唯一操作。

有关基于资源的策略的更多信息，请参见《IAM 用户指南》中的基于身份的策略和基于资源的策略。

步骤 1：向自定义模型添加基于资源的策略

您可以使用 AWS Management Console、AWS CLI 或 Amazon Comprehend API 添加基于资源的策略。

AWS Management Console

您可以在 AWS Management Console 中使用 Amazon Comprehend。

添加基于资源的策略

1. 登录 AWS Management Console 并打开 Amazon Comprehend 控制台，网址：https://console.aws.amazon.com/comprehend/
2. 在左侧导航菜单中的自定义下，请选择包含您的自定义模型的页面：
   a. 如果您要共享自定义文档分类器，请选择自定义分类。
   b. 如果您要共享自定义实体识别器，请选择自定义实体识别。
3. 在模型列表中，选择模型名称以打开其详细信息页面。
4. 在版本下，请选择要共享的模型版本的名称。
5. 在版本详情页面上，选择标签、VPC 和策略选项卡。
6. 在基于资源的策略部分中，选择编辑。
7. 在编辑基于资源的策略页面上，执行以下操作：
   a. 在策略名称中，输入一个可以帮助您在创建策略后识别该策略的名称。
   b. 在授权下，指定以下一个或多个实体以授权它们导入您的模型：

字段	定义和示例
服务主体	可以访问此模型版本的服务的服务主体标识符。例如： comprehend.amazonaws.com
AWS 账户 ID	可以访问此模型版本的 AWS 账户。授权属于该账户的所有用户。例如： 111122223333，123456789012
IAM 实体	可以访问此模型版本的用户或角色的 ARN。例如： arn: aws: iam:: 111122223333: user/，arn: aws: iam:: 444455556666: role/ ExampleUser ExampleRole

8. 在共享下，您可以复制模型版本的 ARN，以帮助您与将要导入模型的人员共享。当有人从不同的 AWS 账户 导入自定义模型时，需要模型版本 ARN。
9. 选择保存。Amazon Comprehend 创建基于资源的策略并将其附加到您的模型。

AWS CLI

要使用向自定义模型添加基于资源的策略AWS CLI，请使用PutResourcePolicy命令。 命令使用以下参数：

- `resource-arn`：自定义模型的 ARN，包括模型版本。
- `resource-policy`：一个 JSON 文件，用于定义要附加到您的自定义模型的基于资源的策略。

  您也可以将策略作为内联 JSON 字符串提供。要为您的策略提供有效的 JSON，请用双引号将属性名和值括起来。如果 JSON 正文也用双引号括起来，则可以转义策略内的双引号。
- `policy-revision-id`：Amazon Comprehend 分配给您正在更新的策略的修订 ID。如果您正在创建没有先前版本的新策略，请不要使用此参数。Amazon Comprehend 会为您创建修订版 ID。

Example 使用 **`put-resource-policy`** 命令向自定义模型添加基于资源的策略

此示例在名为 policyFile.json 的 JSON 文件中定义了一个策略，并将该策略与模型关联起来。该模型是名为 mycf1 的分类器的 v2 版本。

```
$ aws comprehend put-resource-policy \
> --resource-arn arn:aws:comprehend:us-west-2:111122223333:document-classifier/mycf1/version/v2 \
> --resource-policy file://policyFile.json \
> --policy-revision-id revision-id
```

资源策略的 JSON 文件包含以下内容：

- 操作：策略授权指定委托人使用 comprehend:ImportModel。
- 资源：自定义模型的 ARN。资源“*”是指您在 put-resource-policy 命令中指定的模型版本。
- 委托人：该策略授权来自 AWS 账户 444455556666 的用户 jane 和来自 AWS 账户 123456789012 的所有用户。

```
{
"Version":"2012-10-17",
 "Statement":[
    {"Sid":"ResourcePolicyForImportModel",
     "Effect":"Allow",
     "Action":["comprehend:ImportModel"],
     "Resource":"*",
     "Principal":
         {"AWS":
            ["arn:aws:iam::444455556666:user/jane",
             "123456789012"]
         }
   }
 ]
}
```

Amazon Comprehend API

要使用 Amazon Comprehend API 向自定义模型添加基于资源的策略，请使用 API 操作。PutResourcePolicy

您还可以在创建模型的 API 请求中向自定义模型添加策略。为此，请在提交CreateDocumentClassifier或CreateEntityRecognizer请求时提供 ModelPolicy 参数的策略 JSON。

步骤 2：提供其他人需要导入的详细信息

现在，您已将基于资源的策略添加到您的自定义模型中，您已授权其他 Amazon Comprehend 用户将您的模型导入到他们的 AWS 账户 中。但是，在他们可以导入之前，您必须向他们提供以下详细信息：

- 模块版本的 Amazon 资源名称（ARN)。
- 包含模型的 AWS 区域。任何导入您的模型的人都必须使用相同的 AWS 区域。
- 模型是否已加密，如果已加密，则您使用的 AWS KMS 密钥类型：AWS 拥有的密钥 或客户自主管理型密钥。
- 如果您的模型使用客户自主管理型密钥加密，您必须提供 KMS 密钥的 ARN。任何导入您的模型的人都必须在其 AWS 账户 中包含 IAM 服务角色中的 ARN。此角色授权 Amazon Comprehend 在导入期间使用 KMS 密钥来解密模型。

有关其他用户如何导入您的模型的更多信息，请参阅 从另一个 AWS 账户 导入自定义模型。

从另一个 AWS 账户 导入自定义模型

在 Amazon Comprehend 中，您可以导入另一个 AWS 账户 中的自定义模型。当您导入模型时，您会在您的账户中创建一个新的自定义模型。您的新自定义模型是您导入的模型的经过全面训练的副本。

主题

开始之前

在您从另一个 AWS 账户 导入自定义模型之前，请确保与您共享该模型的人员执行以下操作：

- 授权您进行导入。此授权是在附加到模型版本的基于资源的策略中授予的。有关更多信息，请参阅 自定义模型的基于资源的策略。
- 为您提供以下信息：
  - 模块版本的 Amazon 资源名称（ARN)。
  - 包含模型的 AWS 区域。导入时必须使用相同的 AWS 区域。
  - 模型是否使用 AWS KMS 密钥加密，如果是，则使用密钥的类型。

如果模型已加密，则可能需要采取其他步骤，具体取决于所使用的 KMS 密钥类型：

- AWS 拥有的密钥：此类型的 KMS 密钥由 AWS 拥有和管理。如果模型使用 AWS 拥有的密钥 加密，则无需执行其他步骤。
- 客户自主管理型密钥：此类型的 KMS 密钥由AWS 客户在其 AWS 账户 中创建、拥有和管理。如果模型使用客户自主管理型密钥加密，则共享模型的人员必须：
  - 授权您解密模型。此授权是在客户自主管理型密钥的 KMS 密钥策略中授予的。有关更多信息，请参阅 AWS KMS 密钥策略语句。
  - 提供客户自主管理型密钥的标识符。创建 IAM 服务角色时使用此 ARN。此角色授权 Amazon Comprehend 使用 KMS 密钥解密模型。

所需的权限

在导入自定义模型之前，您或您的管理员必须在 AWS Identity and Access Management (IAM) 中授权所需的操作。作为 Amazon Comprehend 用户，您必须获得 IAM policy 语句的导入授权。如果在导入过程中需要加密或解密，则必须授权 Amazon Comprehend 使用必要的 AWS KMS 密钥。

IAM policy 语句

您的用户、组或角色必须附加允许 `ImportModel` 操作的策略，如下例中所示。

Example 导入自定义模型的 IAM policy

```
{
  "Effect": "Allow",
  "Action": [
    "comprehend:ImportModel"
  ],
  "Resource": "arn:aws:comprehend:us-west-2:111122223333:document-classifier/foo/
version/*"
}
```

有关创建 IAM policy 的更多信息，请参阅《IAM 用户指南》中的创建 IAM policy。有关附加 IAM policy 的信息，请参阅《IAM 用户指南》中的添加和删除 IAM 身份权限。

用于 AWS KMS 加密的 IAM 服务角色

导入自定义模型时，您必须授权 Amazon Comprehend 在以下任一情况下使用 AWS KMS 密钥：

- 您正在导入使用 AWS KMS 中客户自主管理型密钥加密的自定义模型。在这种情况下，Amazon Comprehend 需要访问 KMS 密钥，以便在导入期间解密模型。
- 您想对通过导入创建的新自定义模型进行加密，并且使用客户自主管理型密钥。在这种情况下，Amazon Comprehend 需要访问您的 KMS 密钥，以便对新模型进行加密。

要授权 Amazon Comprehend 使用这些 AWS KMS 密钥，您需要创建一个 IAM 服务角色。这种类型的 IAM 角色允许 AWS 服务代表您访问其他服务中的资源。有关创建服务角色的更多信息，请参阅《IAM 用户指南》中的创建角色以将权限委托给 AWS 服务。

如果您使用 Amazon Comprehend 控制台进行导入，则可以让 Amazon Comprehend 为您创建服务角色。否则，您必须在导入之前在 IAM 中创建服务角色。

IAM 服务角色必须具有权限策略和信任策略，如下例中所示。

Example 权限策略

以下权限策略允许 Amazon Comprehend 用于加密和解密自定义模型的 AWS KMS 操作。它授予对两个 KMS 密钥的访问权限：

- 其中有一个 KMS 密钥位于包含要导入的模型的 AWS 账户 中。它被用来加密模型，Amazon Comprehend 在导入过程中使用它来解密模型。
- 另一个 KMS 密钥位于导入模型的 AWS 账户 中。Amazon Comprehend 使用此密钥对通过导入创建的新自定义模型进行加密。

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
        "Effect": "Allow",
        "Action": [
            "kms:CreateGrant"
        ],
        "Resource": [
            "arn:aws:kms:us-west-2:111122223333:key/key-id",
            "arn:aws:kms:us-west-2:444455556666:key/key-id"
        ]
    },
    {
        "Effect": "Allow",
        "Action": [
```

```
            "kms:Decrypt",
            "kms:GenerateDatakey"
        ],
        "Resource": [
            "arn:aws:kms:us-west-2:111122223333:key/key-id",
            "arn:aws:kms:us-west-2:444455556666:key/key-id"
        ],
        "Condition": {
            "StringEquals": {
                "kms:ViaService": [
                    "s3.us-west-2.amazonaws.com"
                ]
            }
        }
    }
  ]
}
```

Example 信任策略

以下信任策略允许 Amazon Comprehend 担任此角色并获得其权限。它允许 comprehend.amazonaws.com 服务主体执行 sts:AssumeRole 操作。为了帮助避免混淆代理，您可以使用一个或多个全局条件上下文键来限制权限的范围。对于 aws:SourceAccount，请指定正在导入模型的用户的账户 ID。

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "Service": "comprehend.amazonaws.com"
      },
      "Action": "sts:AssumeRole",
      "Condition": {
        "StringEquals": {
          "aws:SourceAccount": "444455556666"
        }
      }
    }
  ]
}
```

导入自定义模型

您可以使用 AWS Management Console、AWS CLI 或 Amazon Comprehend API 导入自定义模型。

AWS Management Console

您可以在 AWS Management Console 中使用 Amazon Comprehend。

导入自定义模型

1. 登录 AWS Management Console 并打开 Amazon Comprehend 控制台，网址：https://console.aws.amazon.com/comprehend/
2. 在左侧导航菜单的自定义下，选择要导入的模型类型的页面：
   a. 如果要导入自定义文档分类器，请选择自定义分类。
   b. 如果要导入自定义实体识别器，请选择自定义实体识别。
3. 选择导入版本。
4. 在导入模板版本页面上，输入以下详细信息：
   - 模型版本 ARN：要导入的模型版本的 ARN。
   - 模型名称：导入时创建的新模型的自定义名称。
   - 版本名称：导入时创建的新模型版本的自定义名称。
5. 对于模型加密，请选择用于加密导入时创建的新自定义模型的 KMS 密钥类型：
   - 使用 AWS 自有密钥：Amazon Comprehend 使用由 AWS 代表您创建、管理和使用的 AWS Key Management Service (AWS KMS) 中的密钥来加密您的模型。
   - 选择其他 AWS KMS 密钥（高级）：Amazon Comprehend 使用您在 AWS KMS 中管理的客户自主管理型密钥对您的模型进行加密。

     如果您选择此选项，请选择您的 AWS 账户 中的 KMS 密钥，或者通过选择创建 AWS KMS 密钥来创建新的密钥。
6. 在服务访问权限部分，授予 Amazon Comprehend 访问其所需的任何 AWS KMS 密钥的权限，以便：
   - 解密您导入的自定义模型。
   - 对您通过导入创建的新自定义模型进行加密。

   您可以使用允许 Amazon Comprehend 使用 KMS 密钥的 IAM 服务角色授予访问权限。

对于服务角色，请执行下列操作之一：

- 如果您有要使用的现有服务角色，请选择使用现有 IAM 角色。然后，在角色名称下将其选中。
- 如果您希望 Amazon Comprehend 为您创建角色，请选择创建 IAM 角色。

7. 如果您选择让 Amazon Comprehend 为您创建角色，请执行以下操作：

   a. 对于角色名称，请输入有助于您稍后识别此角色的角色名称后缀。

   b. 对于源 KMS 密钥 ARN，请输入用于加密您导入的模型的 KMS 密钥 ARN。Amazon Comprehend 在导入过程中使用此密钥对模型进行解密。

8. （可选）在标签部分，您可以向通过导入创建的新自定义模型添加标签。有关为自定义模型添加标签的更多信息，请参阅 标记新的资源。

9. 选择确认。

AWS CLI

可以通过利用 AWS CLI 运行命令来使用 Amazon Comprehend。

Example 导入模型命令

要导入自定义模型，请使用 `import-model` 命令：

```
$ aws comprehend import-model \
> --source-model arn:aws:comprehend:us-west-2:111122223333:document-classifier/foo/
version/bar \
> --model-name importedDocumentClassifier \
> --version-name versionOne \
> --data-access-role-arn arn:aws:iam::444455556666:role/comprehendAccessRole \
> --model-kms-key-id kms-key-id
```

本示例使用以下参数：

- `source-model`：要导入的自定义模型的 ARN。
- `model-name`：导入时创建的新模型的自定义名称。
- `version-name`：导入时创建的新模型版本的自定义名称。
- `data-access-role-arn`：IAM 服务角色的 ARN，它允许 Amazon Comprehend 使用必要的 AWS KMS 密钥来加密或解密自定义模型。

- `model-kms-key-id`：Amazon Comprehend 用于加密您通过此导入创建的自定义模型的 KMS 密钥的 ARN 或 ID。此密钥必须位于您的 AWS 账户 中的 AWS KMS 中。

Amazon Comprehend API

要使用 Amazon Comprehend API 导入自定义模型，请使用 API 操作。ImportModel

飞轮

Amazon Comprehend 飞轮简化了随时间改进自定义模型的过程。您可以使用飞轮来编排与训练和评估新的自定义模型版本相关的任务。飞轮支持纯文本自定义模型，用于自定义分类和自定义实体识别。

主题

飞轮概览

飞轮是一种 Amazon Comprehend 资源，用于编排自定义模型新版本的训练和评估。您可以创建一个飞轮以使用现有经过训练的模型，或者 Amazon Comprehend 可以为飞轮创建并训练一个新模型。将飞轮与纯文本自定义模型一起用于自定义分类或自定义实体识别。

您可以使用 Amazon Comprehend 控制台或 API 配置和管理飞轮。您也可以使用 AWS CloudFormation 配置飞轮。

当您创建飞轮时，Amazon Comprehend 会在您的账户中创建一个数据湖。数据湖存储和管理所有飞轮数据，例如模型所有版本的训练数据和测试数据。

您可以将活动模型版本设置为要用于推理作业或 Amazon Comprehend 终端节点的飞轮模型版本。最初，飞轮包含模型的一个版本。随着时间的推移，在训练新模型版本时，您会选择性能最好的版本作为活动模型版本。当用户指定飞轮 ARN 来运行推理作业时，Amazon Comprehend 会使用飞轮的活动模型版本运行该作业。

您定期获取模型的新标注数据（训练数据或测试数据）。您可以通过创建一个或多个数据集来为飞轮提供新数据。数据集包含用于训练或测试与飞轮关联的自定义模型的输入数据。Amazon Comprehend 将输入数据上传到飞轮的数据湖。

要将新数据集合并到您的自定义模型中，您需要创建并运行飞轮迭代。飞轮迭代是一种使用新数据集来评估活动模型版本和训练新模型版本的工作流程。根据现有模型版本和新模型版本的指标，您可以决定是否将新模型版本提升为活动版本。

您可以使用飞轮活动模型版本来运行自定义分析（实时或异步作业）。要使用飞轮模型进行实时分析，必须为飞轮创建终端节点。

使用飞轮不收取任何额外费用。但是，当您运行飞轮迭代时，训练新模型版本和存储模型数据会产生标准费用。有关定价的详细信息，请参阅 Amazon Comprehend 定价。

主题

飞轮数据集

要向飞轮添加新的标注数据，您需要创建一个数据集。您可以将每个数据集配置为训练数据或测试数据。您可以将数据集与特定的飞轮和自定义模型相关联。

创建数据集后，Amazon Comprehend 会将数据上传到飞轮的数据湖。有关更多信息，请参阅 飞轮数据湖。

飞轮创建

当您创建飞轮时，可以将飞轮与现有的训练模型相关联，或者飞轮可以创建新模型。

当您使用现有模型创建飞轮时，需要指定活动模型版本。Amazon Comprehend 将模型的训练数据和测试数据复制到飞轮的数据湖中。确保模型训练和测试数据与您创建模型时位于相同的 Amazon S3 位置。

要为新模型创建飞轮，请在创建飞轮时为训练数据提供数据集（以及用于测试数据的可选数据集）。当您运行飞轮来创建第一个飞轮迭代时，飞轮会训练新模型。

训练自定义模型时，您可以指定要识别的自定义标签（自定义分类）或自定义实体（自定义实体识别）的列表。请注意有关自定义标签/实体的以下要点：

- 当您为新模型创建飞轮时，您在创建飞轮时提供的标签/实体列表是飞轮的最终列表。

- 当您根据现有模型创建飞轮时，与该模型关联的标签/实体列表将成为飞轮的最终列表。
- 如果您将新数据集与飞轮相关联，并且该数据集包含其他标签/实体，则 Amazon Comprehend 会忽略新的标签/实体。
- 您可以使用 API 操作查看飞轮的标签/实体列表。DescribeFlywheel

> **Note**
>
> 对于自定义分类，Amazon Comprehend 会在飞轮状态变为“活动”后填充标签列表。等到飞轮处于活动状态后再调用 DescribeFlywheel API 操作。

飞轮状态

飞轮在以下状态之间切换：

- 正在创建：Amazon Comprehend 正在创建飞轮资源。您可以对飞轮执行读取操作，例如 `DescribeFlywheel`。
- 激活：飞轮处于活动状态。您可以确定飞轮迭代是否正在进行并查看迭代的状态。您可以对飞轮执行读取操作以及诸如 `DeleteFlywheel` 和 `UpdateFlywheel` 之类的操作。
- 更新：Amazon Comprehend 正在更新飞轮。您可以对飞轮执行读取操作。
- 正在删除：Amazon Comprehend 正在删除飞轮。您可以对飞轮执行读取操作。
- 失败：飞轮创建操作失败。

在 Amazon Comprehend 删除飞轮后，您仍保留对飞轮数据湖中所有模型数据的访问权限。Amazon Comprehend 会删除管理飞轮资源所需的所有内部元数据。Amazon Comprehend 还会删除与该飞轮相关的数据集（模型数据保存在数据湖中）。

飞轮迭代

当您获取飞轮模型的新训练或测试数据时，您将创建一个或多个新数据集以将新数据上传到飞轮的数据湖。

您可以运行飞轮来创建新的迭代。飞轮迭代使用新数据评估当前活动模型版本，并将结果存储在数据湖中。飞轮还会创建和训练新的模型版本。

如果新模型表现出比当前活动模型版本更好的性能，则可以将新模型版本升级为活动模型版本。您可以使用控制台或 UpdateFlywheelAPI 操作来更新现役模型版本。

飞轮数据湖

当您创建飞轮时，Amazon Comprehend 会在您的账户中创建一个数据湖，用于存储所有的飞轮数据，例如模型版本所需的输入和输出数据。

Amazon Comprehend 会在您创建飞轮时指定的 Amazon S3 位置创建数据湖。您可以将位置指定为 Amazon S3 存储桶或Amazon S3 存储桶中的新文件夹。

数据湖文件夹结构

当 Amazon Comprehend 创建数据湖时，它会在 Amazon S3 位置设置以下文件夹结构。

> **Warning**
>
> Amazon Comprehend 可管理数据湖文件夹的组织和内容。请务必使用 Amazon Comprehend API 操作来修改数据湖文件夹，否则您的飞轮可能无法正常运行。

```
Document Pool
Annotations Pool
Staging
Model Datasets
  (data for each version of the model)
  VersionID-1
    Training
    Test
    ModelStats
  VersionID-2
    Training
    Test
    ModelStats
```

要查看模型版本的训练评估，请执行以下步骤：

1. 在数据湖的根级别打开名为模型数据集的文件夹。此文件夹包含每个模型版本的子文件夹。
2. 打开相关模型版本的文件夹。
3. 打开名为的文件夹ModelStats以查看模型的统计数据。

数据湖管理

Amazon Comprehend 代表您执行以下任务来管理数据湖：

- 定义数据湖的文件夹结构，并将数据集导入到相应的文件夹中。
- 管理训练模型所需的输入文档（例如文本文件和注释文件）。
- 管理与模型的每个版本相关的训练和评估输出数据。
- 管理数据湖中存储文件的加密。

Amazon Comprehend 执行数据湖的所有数据创建和更新操作。您保留对数据湖中数据的完全访问权限。例如：

- 您可以完全访问数据湖的内容。
- 删除飞轮后，数据湖仍然可用。
- 您可以为包含数据湖的 Amazon S3 存储桶配置访问日志。
- 您可以为数据提供加密密钥。您在创建飞轮时，将指定这些内容。

我们建议您遵循以下最佳实操：

- 不要手动将自己的文件夹或文件添加到数据湖中。请勿修改或删除数据湖中的任何文件。
- 请务必使用 Amazon Comprehend 创建和更新操作来添加或修改数据湖中的数据。例如，用于 `CreateDataset` 提供训练或测试数据以及 `StartFlywheelIteration` 为模型版本生成评估数据。
- 数据湖结构可能会随着时间的推移而演变。不要创建明确依赖数据湖结构的下游脚本或程序。
- 在为飞轮提供数据湖位置时，我们建议为与所有飞轮相关的数据创建一个通用前缀，或者为每个飞轮使用不同的前缀。我们不建议使用一个飞轮的完整数据湖路径作为另一个飞轮的前缀。

IAM policy 和权限

您需要配置以下策略和权限以使用飞轮：

- the section called “配置 IAM 用户权限” 供用户访问飞轮操作。
- （可选）the section called “为 AWS KMS 配置密钥的权限” 用于数据湖。
- the section called “创建数据访问角色” 授权 Amazon Comprehend 访问数据湖。

配置 IAM 用户权限

要使用飞轮功能，请向您的 AWS Identity and Access Management (IAM) 身份（用户、组和角色）添加相应的权限策略。

以下示例显示了创建数据集、创建和管理飞轮以及运行飞轮的权限策略。

Example 管理飞轮的 IAM policy

```
{
  "Effect": "Allow",
  "Action": [
    "comprehend:CreateFlywheel",
    "comprehend:DeleteFlywheel",
    "comprehend:UpdateFlywheel",
    "comprehend:ListFlywheels",
    "comprehend:DescribeFlywheel",
    "comprehend:CreateDataset",
    "comprehend:DescribeDataset",
    "comprehend:ListDatasets",
    "comprehend:StartFlywheelIteration",
    "comprehend:DescribeFlywheelIteration",
    "comprehend:ListFlywheelIterationHistory"
  ],
  "Resource": "*"
}
```

有关为 Amazon Comprehend 创建 IAM policy 的信息，请参阅 Amazon Comprehend 如何与 IAM 配合使用。

为 AWS KMS 配置密钥的权限

如果您在数据湖中使用 AWS KMS 密钥来管理数据，请设置所需的权限。有关信息，请参阅 使用 KMS 加密所需的权限。

创建数据访问角色

您可以在 IAM 中为 Amazon Comprehend 创建数据访问角色以访问数据湖中的飞轮数据。如果您使用控制台创建飞轮，则系统可以选择为此创建新角色。有关更多信息，请参阅 异步操作所需的基于角色的权限。

使用控制台配置飞轮

您可以使用 Amazon Comprehend 控制台创建、更新和删除飞轮。

当您创建飞轮时，Amazon Comprehend 会创建一个数据湖来存储飞轮所需的所有数据，例如每个版本模型的训练数据和测试数据。

当您删除飞轮时，Amazon Comprehend 不会删除与该飞轮关联的数据湖或模型。

在创建新的飞轮之前，请查看 飞轮创建 部分中的信息。

主题

创建飞轮

创建飞轮时，必填的配置字段取决于飞轮是用于现有自定义模型还是新模型。

创建飞轮

1. 登录 AWS Management Console 并打开 Amazon Comprehend 控制台。
2. 从左侧菜单中，选择飞轮。
3. 从飞轮列表中，选择创建一个飞轮。
4. 在飞轮名称下，输入飞轮的名称。
5. （可选）要为现有模型创建飞轮，请在活动模型版本下配置字段。
    a. 从模型下拉列表中，选择一个模型
    b. 从版本下拉列表中，选择模型版本。
6. （可选）要为飞轮创建新的分类器模型，请在自定义模型类型下选择自定义分类，并按以下步骤配置参数。
    a. 在语言下，选择模型的语言。
    b. 在分类器模式下，选择单标签模式或多标签模式。
    c. 在自定义标签下，输入一个或多个用于训练模型的自定义标签。每个标签必须与输入训练数据中的一个类相匹配。

7. （可选）要为飞轮创建新的实体识别模型，请在自定义模型类型下选择自定义实体识别，并按以下步骤配置参数。

   a. 在语言下，选择模型的语言。

   b. 在自定义实体类型下，输入最多 25 个用于训练模型的自定义实体。每个标签必须与输入训练数据中的实体类型相匹配。

      要创建多个标签，请多次执行以下步骤。

      i. 输入自定义标签。标签必须全部为大写。使用下划线作为标签中单词之间的分隔符。

      ii. 选择添加类型。

      要删除已添加的其中一个标签，请选择标签名称右侧的 X。

8. 配置您的卷加密、模型加密和数据湖加密选项。对于每个密钥，请选择是使用 AWS 自有的 KMS 密钥还是您有权使用的密钥。

   - 如果您使用的是 AWS 自有的 KMS 密钥，则没有其他参数。
   - 如果您使用的是其他现有密钥，请在 KMS 密钥 ARN 中输入密钥 ID 的 ARN。
   - 如果您想创建新密钥，请选择创建 KMS AWS 密钥。

   有关创建和使用 KMS 密钥以及相关加密的更多信息，请参阅 AWS Key Management Service。

   a. 配置卷加密密钥。在处理您的任务时，Amazon Comprehend 使用此密钥对存储卷中的数据进行加密。请选择使用 AWS 自有的 KMS 密钥还是您有权使用的密钥。

   b. 配置模型加密密钥。Amazon Comprehend 使用此密钥来加密此模型版本的模型数据。

9. 配置数据湖位置。有关更多信息，请参阅 数据湖管理。

10. （可选）配置数据湖加密密钥。Amazon Comprehend 使用此密钥对数据湖中的所有文件进行加密。

11. （可选）配置 VPC 设置。在 VPC 下输入 VPC ID 或从下拉列表中选择 ID。

    1. 在子网下选择子网。选择第一个子网后，您还可以选择其他子网。
    2. 在安全组下，选择要使用的安全组（如果已指定）。选择第一个安全组后，您还可以选择其他安全组。

12. 配置服务访问权限。

    1. 如果您选择使用现有 IAM 角色，请在下拉列表中选择角色名称。

2. 如果您选择创建 IAM 角色，Amazon Comprehend 会创建一个新角色。控制台显示 Amazon Comprehend 为该角色配置的权限。在角色名称下，请为角色输入一个名称。

13. （可选）配置标签设置。要添加标签，请在标签下输入键值对。选择添加标签。要在创建飞轮之前删除这对，请选择删除标签。有关更多信息，请参阅 标记您的资源。
14. 选择创建。

更新飞轮

只有在创建飞轮时，才能配置飞轮名称、数据湖位置、模型类型和模型配置。

更新飞轮时，如果模型类型和配置选项与当前模型相同，则可以指定其他模型。您可以配置新的活动模型版本。您还可以更新加密详细信息、服务访问权限和 VPC 设置。

更新飞轮

1. 登录 AWS Management Console 并打开 Amazon Comprehend 控制台。
2. 从左侧菜单中，选择飞轮。
3. 从飞轮列表中，选择要更新的飞轮。
4. 在活动模型版本下，从模型下拉列表中选择模型并选择模型版本。

   该表单会填充模型类型和模型配置。
5. （可选）配置卷加密和模型加密设置。
6. （可选）配置数据湖加密设置。
7. 配置服务访问权限。
8. （可选）配置 VPC 设置。
9. （可选）配置标签设置。
10. 选择保存。

删除飞轮

删除飞轮

1. 登录 AWS Management Console 并打开 Amazon Comprehend 控制台。
2. 从左侧菜单中，选择飞轮。
3. 从飞轮列表中，选择要删除的飞轮。

4. 选择删除。

使用 API 配置飞轮

您可以使用 Amazon Comprehend API 创建、更新和删除飞轮。

当您创建飞轮时，Amazon Comprehend 会创建一个数据湖来存储飞轮所需的所有数据，例如每个版本模型的训练数据和测试数据。

当您删除飞轮时，Amazon Comprehend 不会删除与该飞轮关联的数据湖或模型。

如果飞轮正在运行迭代或创建数据集，则飞轮删除操作将失败。

在创建新的飞轮之前，请查看 飞轮创建 部分中的信息。

为现有模型创建飞轮

使用CreateFlywheel操作为现有模型创建飞轮。

Example

```
aws comprehend create-flywheel  \
    --flywheel-name "myFlywheel2"  \
    --active-model-arn  "modelArn"  \
    --data-access-role-arn   arn:aws::iam::111122223333:role/testFlywheelDataAccess \
    --data-lake-s3-uri": "https://s3-bucket-endpoint"   \
```

如果操作成功，则响应将包含飞轮 ARN。

```
{
  "FlywheelArn": "arn:aws::comprehend:aws-region:111122223333:flywheel/name",
  "ActiveModelArn": "modelArn"
}
```

为新模型创建飞轮

使用CreateFlywheel操作为新的自定义分类模型创建飞轮。

Example

```
aws comprehend create-flywheel \
```

```
    --flywheel-name "myFlywheel2" \
    --data-access-role-arn  arn:aws::iam::111122223333:role/testFlywheelDataAccess \
    --model-type "DOCUMENT_CLASSIFIER" \
    --data-lake-s3-uri  "s3Uri"  \
    --task-config  file://taskConfig.json
```

taskConfig.json 文件包含以下内容。

```
{
    "LanguageCode": "en",
    "DocumentClassificationConfig": {
        "Mode": "MULTI_LABEL",
        "Labels": ["optimism", "anger"]
    }
}
```

API 响应正文包含以下内容。

```
{
  "FlywheelArn": "arn:aws::comprehend:aws-region:111122223333:flywheel/name",
  "ActiveModelArn": "modelArn"
}
```

描述飞轮

使用 Amazon Compreh DescribeFlywheelend 操作检索有关飞轮的配置信息。

```
aws comprehend describe-flywheel \
    --flywheel-arn  "flywheelArn"
```

API 响应正文包含以下内容。

```
{
  "FlywheelProperties": {
      "FlywheelArn": "arn:aws::comprehend:aws-region:111122223333:flywheel/
myTestFlywheel",
      "DataAccessRoleArn": "arn:aws::iam::111122223333:role/Admin",
      "TaskConfig": {
          "LanguageCode": "en",
          "DocumentClassificationConfig": {
              "Mode": "MULTI_LABEL"
```

```
          }
      },
      "DataLakeS3Uri": "s3://my-test-datalake/flywheelbasictest/myTestFlywheel/
schemaVersion=1/20220801T014326Z",
      "Status": "ACTIVE",
      "ModelType":  "DOCUMENT_CLASSIFIER",
      "CreationTime": 1659318206.102,
      "LastModifiedTime": 1659318249.05
  }
}
```

更新飞轮

使用该UpdateFlywheel操作更新飞轮的可修改配置值。

某些配置字段是带有子字段的 JSON 结构。要更新一个或多个子字段，请为所有子字段提供值（对于请求中缺少的任何子字段，Amazon Comprehend 会将该值设置为空）。

如果您在 `UpdateFlywheel` 请求中省略了顶级参数，则 Amazon Comprehend 不会更改飞轮中该参数或其任何子字段的值。

要在飞轮上添加或删除标签，请使用TagResource和UntagResource操作。

您还可以通过设置 `ActiveModelArn` 参数来升级模型版本，如以下示例中所示。

```
aws comprehend update-flywheel \
    --region aws-region \
    --flywheel-arn  "flywheelArn" \
    --active-model-arn  "modelArn" \
```

API 响应正文包含以下内容。

```
{
  "FlywheelArn": "arn:aws::comprehend:aws-region:111122223333:flywheel/name",
  "ActiveModelArn": "modelArn"
}
```

删除飞轮

使用 Amazon Compreh DeleteFlywheelend 操作删除飞轮。

```
aws comprehend delete-flywheel \
```

```
    --flywheel-arn  "flywheelArn"
```

成功的 API 响应包含一个空的响应信息正文

列出飞轮

使用 Amazon Com ListFlywheelsprehend 操作检索当前区域的飞轮列表。

```
aws comprehend list-flywheel \
    --region aws-region \
    --endpoint-url  "uri"
```

API 响应正文包含以下内容。

```
{
    "FlywheelSummaryList": [
        {
            "FlywheelArn": "arn:aws::comprehend:aws-region:111122223333:flywheel/
myTestFlywheel",
            "DataLakeS3Uri": "s3://my-test-datalake/flywheelbasictest/myTestFlywheel/
schemaVersion=1/20220801T014326Z",
            "Status": "ACTIVE",
            ""ModelType":  "DOCUMENT_CLASSIFIER",
            "CreationTime": 1659318206.102,
            "LastModifiedTime": 1659318249.05
        }
    ]
}
```

配置数据集

要将带标签的训练或测试数据添加到飞轮中，请使用 Amazon Comprehend 控制台或 API 创建数据集。

您可以将每个数据集配置为训练数据或测试数据。您可以将数据集与特定的飞轮和自定义模型相关联。创建数据集时，Amazon Comprehend 会将数据上传到飞轮的数据湖。有关训练数据文件格式的详细信息，请参阅 准备分类器训练数据 或 准备实体识别器训练数据。

当您删除飞轮时，Amazon Comprehend 会删除数据集。上传的数据在数据湖中仍然可用。

创建数据集（控制台）

创建数据集

1. 登录 AWS Management Console 并打开 Amazon Comprehend 控制台。
2. 从左侧菜单中选择飞轮，然后选择要在其中添加数据的飞轮。
3. 选择数据集选项卡。
4. 在训练数据集或测试数据集表中，选择创建数据集。
5. 在数据集详细信息下，输入数据集的名称和可选描述。
6. 在数据规范下，选择数据格式和数据集类型配置字段。
7. （可选）在输入格式下，选择输入文档的格式。
8. 在 S3 上的注释位置下，输入注释文件的 Amazon S3 位置。
9. 在 S3 上的训练数据位置下，输入文档文件的 Amazon S3 位置。
10. 选择创建。

创建数据集 (API)

您可以使用该CreateDataset操作来创建数据集。

Example

```
aws comprehend create-dataset \
    --flywheel-arn "myFlywheel2" \
    --dataset-name "my-training-dataset"
    --dataset-type "TRAIN"
    --description "my training dataset"
    --cli-input-json file://inputConfig.json
}
```

inputConfig.json 文件包含以下代码。

```
{
    "DataFormat": "COMPREHEND_CSV",
    "DocumentClassifierInputDataConfig": {
        "S3Uri": "s3://my-comprehend-datasets/multilabel_train.csv"
    }
}
```

要在数据集中添加或移除标签，请使用TagResource和UntagResource操作。

描述数据集

使用 Amazon Compreh DescribeDatasetend 操作检索有关飞轮的配置信息。

```
aws comprehend describe-dataset \
    --dataset-arn  "datasetARN"
```

响应包含以下内容。

```
{
   "DatasetProperties": {
      "DatasetArn": "arn:aws::comprehend:aws-region:111122223333:flywheel/
myTestFlywheel/dataset/train-dataset",
      "DatasetName": "train-dataset",
      "DatasetType": "TRAIN",
      "DatasetS3Uri": "s3://my-test-datalake/flywheelbasictest/myTestFlywheel/
schemaVersion=1/20220801T014326Z/datasets/train-dataset/20220801T194844Z",
      "Description": "Good Dataset",
      "Status": "COMPLETED",
      "NumberOfDocuments": 90,
      "CreationTime": 1659383324.297
  }
}
```

飞轮迭代

使用飞轮迭代帮助您创建和管理新的模型版本。

主题

- 迭代工作流程
- 管理迭代（控制台）
- 管理迭代 (API)

迭代工作流程

飞轮从经过训练的模型版本开始，或者使用初始数据集来训练模型版本。

随着时间的推移，当您获得新的标记数据时，您可以训练新的模型版本以提高飞轮模型的性能。当您运行飞轮时，它会创建一个新的迭代，用于训练和评估新的模型版本。如果新模型版本的性能优于现有的活动模型版本，则可以对其进行升级。

飞轮迭代工作流程包括以下步骤：

1. 您可以为新标记的数据创建数据集。
2. 您可以运行飞轮来创建新的迭代。迭代遵循以下步骤来训练和评估新的模型版本：
   a. 使用新数据评估活动模型版本。
   b. 使用新数据训练新的模型版本。
   c. 将评估和训练结果存储在数据湖中。
   d. 返回两个模型的 F1 分数。
3. 迭代完成后，您可以比较现有活动模型和新模型的 F1 分数。
4. 如果新模型版本具有卓越的性能，则将其提升为活动模型版本。您可以使用控制台或 API 来提升新的模型版本。

管理迭代（控制台）

您可以使用控制台开始新的迭代并查询正在进行的迭代的状态。您还可以查看已完成迭代的结果。

开始飞轮迭代（控制台）

在开始新的迭代之前，请创建一个或多个新的训练或测试数据集。请参阅 配置数据集。

开始飞轮迭代（控制台）

1. 登录 AWS Management Console 并打开 Amazon Comprehend 控制台。
2. 从左侧菜单中，选择飞轮。
3. 从飞轮列表中，选择一个飞轮。
4. 选择运行飞轮。

分析迭代结果（控制台）

运行飞轮迭代后，控制台将结果显示在飞轮迭代列表中。

提升新模型版本（控制台）

在控制台的模型详细信息页面上，您可以将新的模型版本升级为活动模型版本。

将飞轮模型版本升级为活动模型版本（控制台）

1. 登录 AWS Management Console 并打开 Amazon Comprehend 控制台。
2. 从左侧菜单中，选择飞轮。
3. 从飞轮列表中，选择一个飞轮。
4. 从飞轮详细信息页面列表中，从飞轮迭代列表中选择要升级的版本。
5. 选择激活模型。

管理迭代 (API)

您可以使用 Amazon Comprehend API 开始新的迭代并查询正在进行的迭代的状态。您还可以查看已完成迭代的结果。

开始飞轮迭代 (API)

使用 Amazon Compreh StartFlywheelIterationend 操作开始飞轮迭代。

```
aws comprehend start-flywheel-iteration \
    --flywheel-arn  "flywheelArn"
```

响应包含以下内容。

```
{
  "FlywheelIterationArn": "arn:aws::comprehend:aws-region:111122223333:flywheel/name"
}
```

提升新模型版本 (API)

使用UpdateFlywheel操作将模型版本提升为活动模型版本。

发送 `UpdateFlywheel` 请求时，将 `ActiveModelArn` 参数设置为新活动模型版本的 ARN。

```
aws comprehend update-flywheel \
    --active-model-arn  "modelArn" \
```

响应包含以下内容。

```
{
  "FlywheelArn": "arn:aws::comprehend:aws-region:111122223333:flywheel/name",
  "ActiveModelArn": "modelArn"
}
```

描述飞轮迭代结果 (API)

在迭代运行到完成之后，DescribeFlywheelIterationAmazon Comprehend 操作会返回有关迭代的信息。

```
aws comprehend describe-flywheel-iteration \
 --flywheel-arn "flywheelArn" \
 --flywheel-iteration-id  "flywheelIterationId" \
 --region aws-region
```

响应包含以下内容。

```
{
    "FlywheelIterationProperties": {
        "FlywheelArn": "flywheelArn",
        "FlywheelIterationId": "iterationId",
        "CreationTime": <createdAt>,
        "EndTime": <endedAt>,
        "Status": <status>,
        "Message": <message>,
        "EvaluatedModelArn": "modelArn",
        "EvaluatedModelMetrics": {
            "AverageF1Score": <value>,
            "AveragePrecision": <value>,
            "AverageRecall": <value>,
            "AverageAccuracy": <value>
        },
        "TrainedModelArn": "modelArn",
        "TrainedModelMetrics": {
            "AverageF1Score": <value>,
            "AveragePrecision": <value>,
            "AverageRecall": <value>,
            "AverageAccuracy": <value>
        }
    }
```

```
}
```

获取迭代历史记录 (API)

使用该ListFlywheelIterationHistory操作可获取有关迭代历史记录的信息。

```
aws comprehend list-flywheel-iteration-history \
 --flywheel-arn "flywheelArn"
```

响应包含以下内容。

```
{
    "FlywheelIterationPropertiesList": [
        {
            "FlywheelArn": "<flywheelArn>",
            "FlywheelIterationId": "20220907T214613Z",
            "CreationTime": 1662587173.224,
            "EndTime": 1662592043.02,
            "Status": "<status>",
            "Message": "<message>",
            "EvaluatedModelArn": "modelArn",
            "EvaluatedModelMetrics": {
                "AverageF1Score": 0.8333333333333333,
                "AveragePrecision": 0.75,
                "AverageRecall": 0.9375,
                "AverageAccuracy": 0.8125
            },
            "TrainedModelArn": "modelArn",
            "TrainedModelMetrics": {
                "AverageF1Score": 0.865497076023392,
                "AveragePrecision": 0.7636363636363637,
                "AverageRecall": 1.0,
                "AverageAccuracy": 0.84375
            }
        }
    ]
}
```

使用飞轮进行分析

您可以使用飞轮的活动模型版本运行分析以进行自定义分类或实体识别。活动模型版本是可配置的。您可以使用控制台或 UpdateFlywheelAPI 操作将模型的新版本设置为活动模型版本。

如要使用飞轮，请在配置分析任务时指定飞轮 ARN 而不是自定义模型 ARN。Amazon Comprehend 使用飞轮的活动模型版本进行分析。

实时分析

您使用终端节点来运行实时分析。创建或更新终端节点时，您可以使用飞轮 ARN 而不是模型 ARN 对其进行配置。运行实时分析时，选择与飞轮关联的终端节点。Amazon Comprehend 使用飞轮的活动模型版本进行分析。

当您使用UpdateFlywheel为飞轮设置新的活动模型版本时，端点会自动更新以开始使用新的活动模型版本。如果您不希望终端节点自动更新，请将终端节点（使用 UpdateEndpoint）配置为直接使用模型版本 ARN。如果飞轮活动模型版本发生变化，终端节点将继续使用该模型版本。

要进行自定义分类，请使用 ClassifyDocumentAPI 操作。要识别自定义实体，请使用 DetectEntitiesAPI 请求。在 EndpointArn 参数中提供飞轮的终端节点。

您还可以使用控制台运行实时分析，以进行自定义分类或自定义实体识别。

异步作业

要进行自定义分类，请使用 StartDocumentClassificationJobAPI 请求启动异步作业。提供 FlywheelArn 参数而不是 DocumentClassifierArn。

要识别自定义实体，请使用 StartEntitiesDetectionJobAPI 请求。提供 FlywheelArn 参数而不是 EntityRecognizerArn。

您还可以使用控制台运行异步分析作业，以进行自定义分类或自定义实体识别。创建任务时，在识别器模型或分类器模型字段中输入飞轮 ARN。

管理 Amazon Comprehend 终端节点

在 Amazon Comprehend 中，终端节点使您的自定义模型可用于实时分类或实体检测。在创建终端节点之后，您可以根据业务需求的变化对其进行修改。例如，您可以监控终端节点利用率并应用自动扩缩来自动设置终端节点配置以满足您的容量需求。您可以从单一视图管理所有终端节点，当您不再需要某个终端节点时，可以将其删除以节省成本。

必须先创建终端节点，然后才能管理终端节点。有关更多信息，请参阅以下流程：

- 为自定义分类创建终端节点
- 为自定义实体检测创建终端节点

主题

Amazon Comprehend 终端节点概览

Amazon Comprehend 控制台中的终端节点页面为您提供了终端节点的全局视图。在终端节点概述页面上，您可以在一个地方查看所有终端节点，以了解您的终端节点使用情况与实际资源使用情况。在终端节点页面的右上角，您可以指定要查看的终端节点，包括所有终端节点、自定义分类器终端节点或您的自定义实体终端节点。

您可以在此页面创建、更新、监控和删除终端节点。在终端节点概述部分中，您可以查看终端节点列表、终端节点托管的自定义模型、它们的创建时间、预配置的吞吐量以及终端节点的状态。当您从终端节点概述表中选择特定终端节点时，将显示终端节点的详细信息。

此外，如果您是 AWS 商业支持或 AWS 企业支持客户，则可以访问特定于您的终端节点的 Trusted Advisor 检查。要了解更多信息，请参阅在亚马逊 Compreh Trusted Advisor end 上使用 Amazon Comprehend。有关检查和描述的完整列表，请参阅 Trusted Advisor 最佳实践。

有关管理终端节点的更多信息，请参阅以下主题。

- 使用 Amazon Comprehend 终端节点
- 正在监控 Amazon Comprehend 终端节点
- 正在更新 Amazon Comprehend 终端节点
- 在亚马逊 Compreh Trusted Advisor end 上使用 Amazon Comprehend
- 正在删除 Amazon Comprehend 终端节点

> **Important**
>
> 实时自定义分类的成本取决于您设置的吞吐量和终端节点处于活动状态的时间长度。如果您不再使用终端节点，或者长时间不使用该终端节点，则应设置自动扩缩策略以降低成本。或者，如果您不再使用某个终端节点，则可以删除该终端节点，以免产生额外费用。有关更多信息，请参阅 使用终端节点自动扩缩。

使用 Amazon Comprehend 终端节点

您可以使用自定义模型创建终端节点来运行实时分析。终端节点包括托管资源，这些资源使您的自定义模型可用于实时推理。

Amazon Comprehend 使用推理单元 (IU) 为终端节点分配吞吐量。一个 IU 表示每秒 100 个字符的数据吞吐量。您最多可以为终端节点配置 10 个推理单元。您可以通过更新终端节点来向上或向下扩展终端节点的吞吐量。

如果您的输入文档包括半结构化文档或图像文件，则从输入文件中提取的字符的吞吐量为每秒 100 个字符。您为终端节点配置的 IU 数量取决于输入文档的字符密度。

ClassifyDocument和 DetectEntitiesAPI 响应包括每页输入的字符数。您可以使用此信息来估算要配置的推理单元数量，以实现所需的吞吐量。

完成实时分析后，请删除终端节点，因为只要终端节点处于活动状态，就会继续收费。当您准备好进行进一步的实时分析时，可以创建另一个终端节点。

有关终端节点费用的更多信息，请参阅 Amazon Comprehend 定价。

创建终端节点后，您可以使用 Amazon 对其进行监控 CloudWatch，对其进行更新以更改其推理单位，或者在不再需要时将其删除。有关更多信息，请参阅 正在监控 Amazon Comprehend 终端节点。

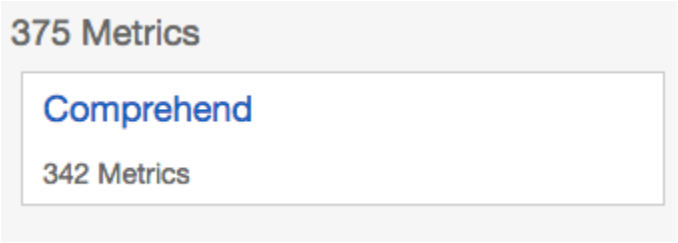
正在监控 Amazon Comprehend 终端节点

您可以通过增加或减少推理单元 (IU) 的数量来调整终端节点的吞吐量。有关更新您的终端节点的更多信息，请参阅 the section called “更新终端节点”。

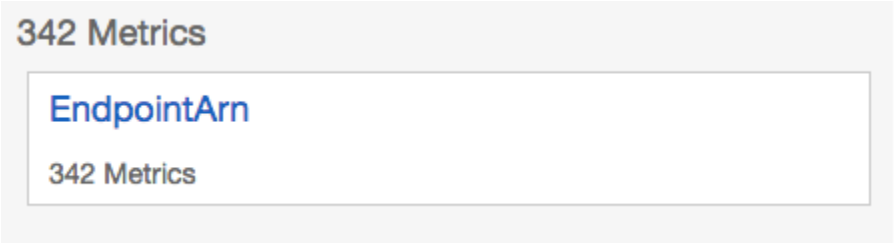
您可以通过使用 Amazon CloudWatch 控制台监控终端节点的使用情况来确定如何最好地调整终端节点的吞吐量。

使用监控您的终端节点使用情况 CloudWatch

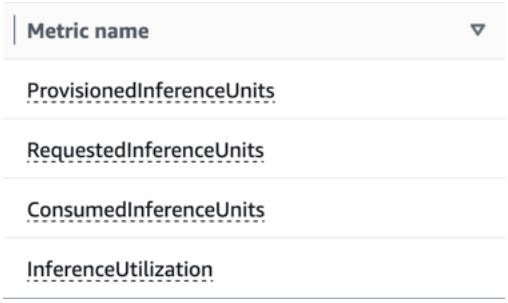
1. 登录 AWS Management Console 并打开CloudWatch 控制台。
2. 请在左侧选择指标，然后选择全部指标。
3. 在全部指标下，选择 Comprehend。



4. CloudWatch 控制台显示 Comprehend 指标的维度。选择EndpointArn维度。



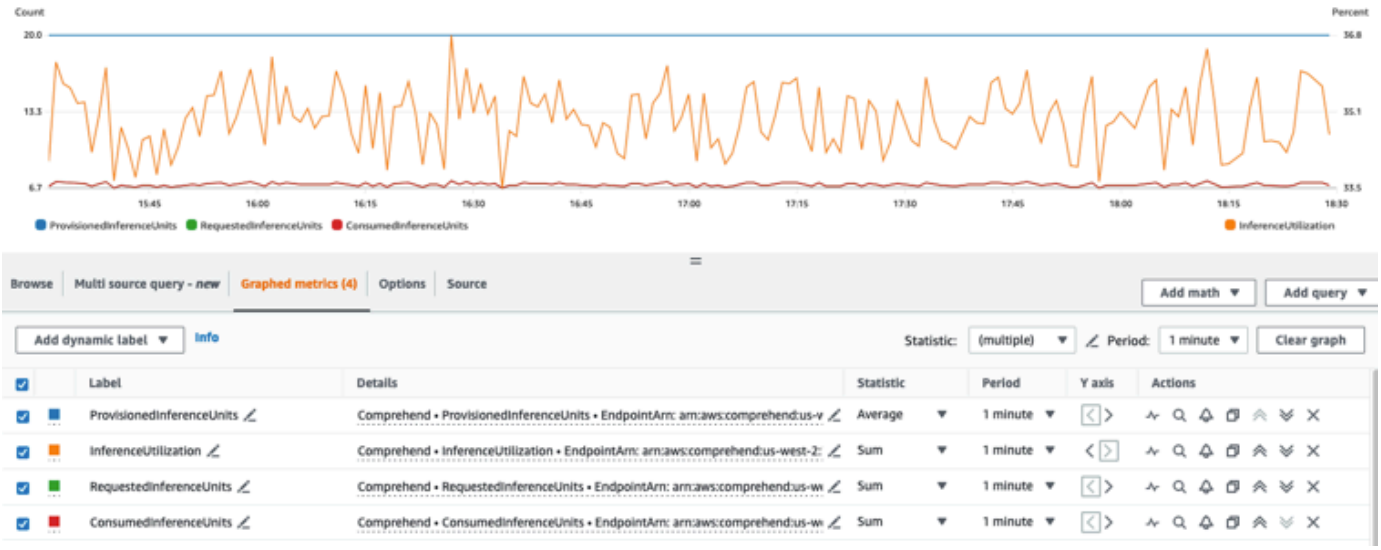
控制台会显示您的每个终端节点的ProvisionedInferenceUnitsRequestedInferenceUnitsConsumedInferenceUnits、、和InferenceUtilization。



选择四个指标，然后导航到 “图表化指标” 选项卡。

5. 将和的 “统计数据” 列设置为 ConsumedInferenceUnits“总RequestedInferenceUnits和”。
6. 将的 “统计数据” 列设置InferenceUtilization为 “总和”。
7. 将 “统计数据” 列设置ProvisionedInferenceUnits为 “平均值”。
8. 将所有指标的周期列更改为 1 分钟。
9. 选择InferenceUtilization并选择箭头，将其移动到单独的 Y 轴。

   您的图表已准备好进行分析。



根据这些 CloudWatch 指标，您还可以设置 auto scaling 以自动调整终端节点的吞吐量。有关更多在终端节点上使用自动扩缩信息，请参阅 使用终端节点自动扩缩。

- ProvisionedInferenceUnits-此指标表示发出请求时平均预配置 IU 的数量。
- RequestedInferenceUnits-这是基于提交给服务且发送待处理的每个请求的使用情况。这有助于将发送的待处理请求与实际处理的请求进行比较，而不会受到限制 () ConsumedInferenceUnits。该指标的值的计算方法是，将发送的待处理字符数除以一分钟内可处理的字符数，得出 1 IU。
- ConsumedInferenceUnits-这是基于提交给服务且成功处理（未受限制）的每个请求的使用情况。这在您比较使用情况与预配置 IU 时非常有用。该指标的计算方法是将处理的字符数除以每分钟1个 IU 可以处理的字符数。
- InferenceUtilization-这是根据请求发出的。该值的计算方法是将定义的消耗的 IU 除以ProvisionedInferenceUnits并转换为 100 中的ConsumedInferenceUnits百分比。

> **Note**
>
> 只有成功请求时才会发出所有指标。如果该指标来自因内部服务器错误或客户错误而受到节流或失败的请求，则不会显示该指标。

正在更新 Amazon Comprehend 终端节点

通常，在创建终端节点之后，您对吞吐量的需求水平可能会发生变化，或者您对需求的首次估算会发生变化。发生这种情况时，可能需要更新您的终端节点以向上或向下调整吞吐量。吞吐量取决于您为终端节点配置的推理单元数量。每个推理单元表示每秒 100 个字符的吞吐量，每秒最多可处理 2 个文档。您可能还想更新与终端节点关联的模型的版本。编辑终端节点时，可以为该终端节点选择不同版本的模型。

为终端节点添加标签也能帮助将它们整理得井井有条。这也可以在更新终端节点时来完成。有关终端节点的更多信息，请参阅 标记您的资源

要更新终端节点（控制台）

1. 登录 AWS Management Console 并打开 Amazon Comprehend 控制台，网址：https://console.aws.amazon.com/comprehend/
2. 从左侧菜单中，选择终端节点。
3. 从分类器列表中，选择要更新终端节点的自定义模型的名称，然后点击链接。此时将显示模型详细信息页面。
4. 在模型详细信息页面中，选择版本详细信息。将显示终端节点列表。
5. 选中终端节点对应的终端节点复选框。在终端节点列表的右上角，选择操作图标。
6. 选择编辑。您可以更新预配置的 IU 并编辑标签。
7. 保存您的更改。
8. 要编辑配置终端节点的推理单元的数量，请选择编辑。
9. 输入要分配给终端节点的更新的推理单元数量。每个单元代表每秒 100 个字符的吞吐量。您可以为每个终端节点分配最多 10 个推理单元。

> **Note**
>
> 使用终端节点的成本取决于运行时间和吞吐量（基于推理单元的数量）。因此，增加推理单元的数量会增加操作成本。有关更多信息，请参阅 Amazon Comprehend 定价。

10. 选择编辑终端节点。显示终端节点详细信息页面。
11. 请从页面顶部的页面导览痕迹中选择模型，以确认终端节点正在更新。在自定义模型详细信息页面上，导航到终端节点列表，并确认终端节点旁边是否显示正在更新。更新完成后，状态将显示为就绪。

    以下示例演示了如何在 AWS CLI 中使用该UpdateEndpoint操作。

    此示例的格式适用于 Unix、Linux 和 macOS。对于 Windows，请将每行末尾的反斜杠 (\) Unix 行继续符替换为脱字号 (^)。

    ```
    aws comprehend update-endpoint \
        --desired-inference-units updated number of inference units \
        --desired-model-arn arn:aws:comprehend:region:account-id:model type/model name
     \
        --desired-data-access-role-arn arn:aws:iam:account id:role/role name
        --endpoint-arn arn:aws:comprehend:region:account id:endpoint/endpoint name
    ```

    如果此操作成功，则 Amazon Comprehend 会发送回带有空 HTTP 正文的 HTTP 200 响应。
12. 要编辑附加到终端节点的自定义模型，请从自定义模型详细信息页面导航到终端节点列表。
13. 选择要更改的终端节点，然后选择编辑。
14. 在终端节点设置页面的选择分类器模型或选择识别器模型下（视终端节点而定），您可以在下拉列表中搜索模型。选择您所需的模型。
15. 在选择版本下，您可以搜索所需的模型版本。选择版本。
16. 选择编辑终端节点进行保存。

在亚马逊 Compreh Trusted Advisor end 上使用 Amazon Comprehend

AWS Trusted Advisor 是一款在线工具，可提供建议，帮助您按照 AWS 最佳实践配置资源。

如果您有 Basic 或 Developer Support 计划，则可以使用 Trusted Advisor 控制台访问 “服务限制” 类别中的所有检查和 “安全” 类别中的六项检查。如果您有商业或企业支持计划，则可以使用 Trusted Advisor 控制台和 AWS Support API 访问所有支 Trusted Advisor 票。

Amazon Comprehend 支持以下检查，通过提供 Trusted Advisor 切实可行的建议，帮助客户优化其亚马逊 Comprehend 终端节点的成本和安全。

Amazon Comprehend 未充分利用的终端节点

Amazon Comprehend 未充分利用终端节点检查会评估终端节点的吞吐量配置。当终端节点未被主动用于实时推理请求时，此检查会提示您。超过 15 天未使用的终端节点会被视为未充分利用。所有终端节点均根据设置的吞吐量以及终端节点处于活动状态的时长来累积费用。如果终端节点在过去 15 天内尚未使用，我们建议使用应用程序 Autoscaling 为资源定义扩缩策略。对于过去 30 天内未使用且已定义自动扩缩策略的终端节点，我们建议您使用异步推理或将其删除。这些检查结果每天自动刷新一次，可以在 Trusted Advisor 控制台的CostOptimization类别下查看。

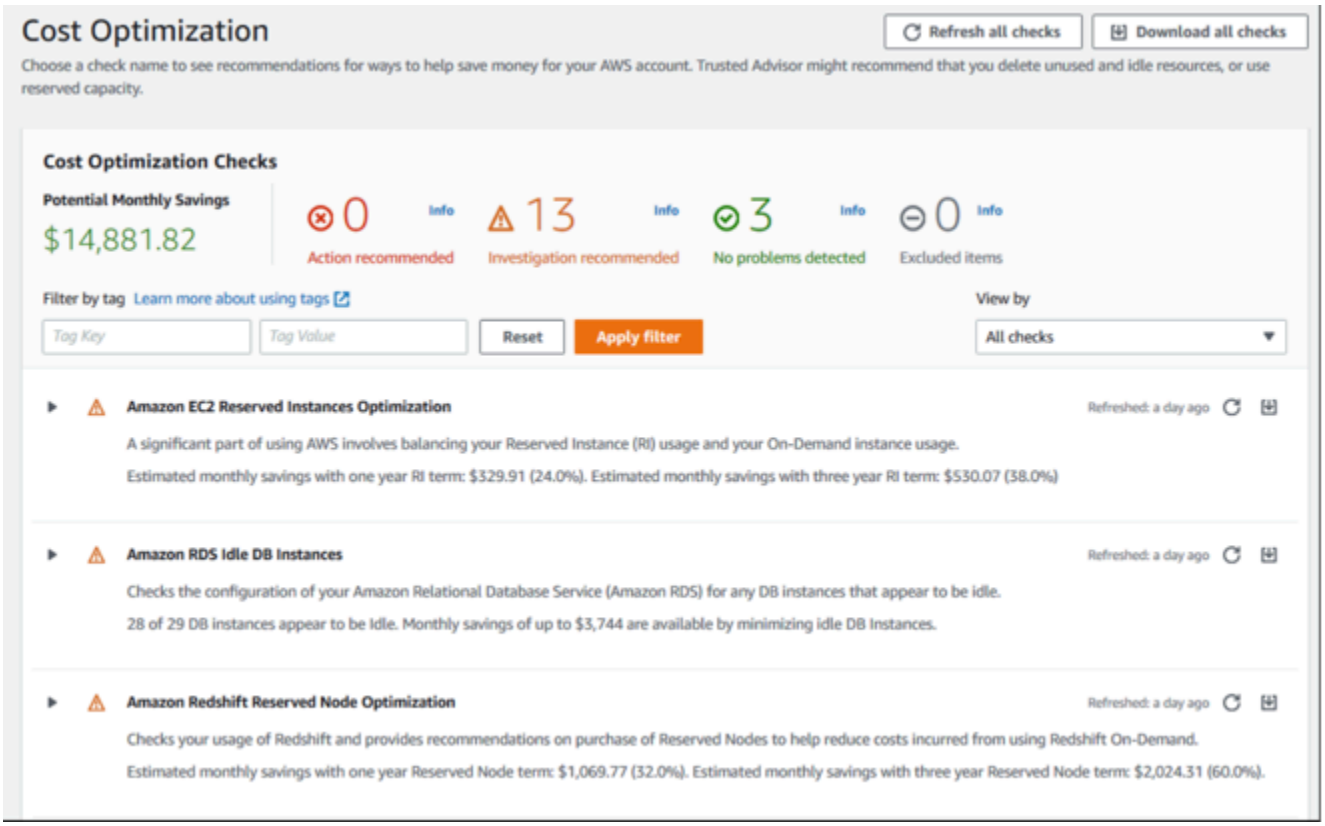
查看所有终端节点的使用状态和相应的建议

1. 登录 AWS Management Console 并打开 Trusted Advisor 控制台。
2. 在导航窗格中，选择支CostOptimization票类别。
3. 在类别页面上，您可以查看每种检查类别的摘要：
   - 建议的操作（红色）- Trusted Advisor 建议检查的操作。
   - 建议调查（黄泽） – Trusted Advisor 检测到检查的可能问题。
   - 未检测到问题（绿色）- Trusted Advisor 未检测到检查问题。
   - 排除的项目（灰色）：包含排除项目的检查次数，例如您希望检查忽略的资源。
4. 选择 Amazon Comprehend 未充分利用的终端节点检查以查看检查说明和以下详细信息：
   - 提示标准 – 描述检查将更改状态的阈值。
   - 建议的操作 – 描述此检查的建议操作。
   - 资源表：此表根据您的建议列出了您的终端节点详细信息以及每个终端节点的状态。
5. 在资源表中，如果终端节点因过去 30 天未使用警告而被标记为建议调查，则可以导航到 Amazon Comprehend 控制台上的终端节点详情页面。
   - 如果您不想再使用此终端节点，请选择删除。
   - 再次选择删除以确认删除。显示自定义模型的详细信息页面。确认您删除的终端节点旁边显示正在删除。删除后，终端节点将从终端节点列表中删除。
6. 在 Trusted Advisor 控制台的资源表中，如果某个终端节点因过去 15 天内未使用而被标记为 “建议调查” 状态，并且该终端节点已被 AutoScaling 禁用，则可以导航到 Amazon Comprehend 控制台上的终端节点详细信息页面来调整该终端节点。
   - 如果要减少为此终端节点配置的吞吐量，请单击编辑。输入要分配给终端节点的推理单元的更新数量，然后选中要确认的复选框，然后选择编辑终端节点。更新完成后，状态将显示为就绪。

- 如果您想在终端节点上自动设置终端节点配置，而不是手动调整吞吐量配置，我们建议您使用应用程序 Autoscaling。

7. 在 Trusted Advisor 控制台的资源表中，如果由于 Used Activity 原因将终端节点标记为 “未检测到问题” 状态，则表示该端点正在被积极用于运行实时推理请求，不建议采取任何操作。

以下示例显示了 Trusted Advisor 控制台上的 CostOptimization 类别视图：



Amazon Comprehend 终端节点访问风险

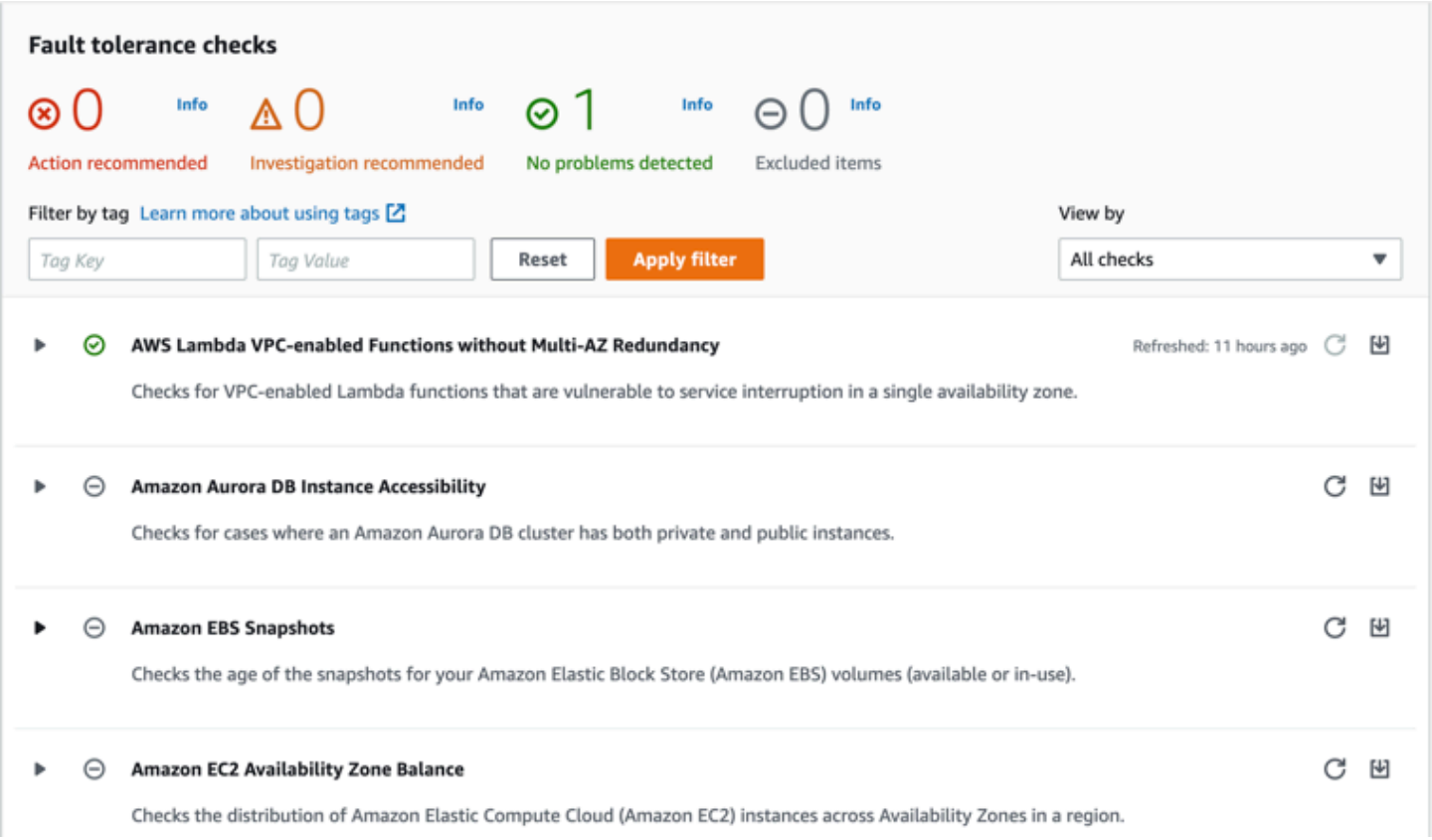
Amazon Comprehend 终端节点访问风险检查会评估使用客户自主管理型密钥加密基础模型的终端节点的 AWS Key Management Service (AWS KMS) 密钥权限。如果禁用了客户自主管理型密钥，或者更改了密钥策略以针对 Amazon Comprehend 更改允许的权限，则终端节点可用性可能会受到影响。如果密钥已禁用，我们建议您启用它。如果密钥策略已更改并且您想要继续使用终端节点，我们建议您更新密钥策略。检查结果在一天中会自动刷新多次。可以在 Trusted Advisor 控制台的容错类别下查看此检查。

查看您的亚马逊 Comprehend 终端节点的 AWS KMS 密钥状态

1. 登录 AWS Management Console 并打开 Trusted Advisor 控制台。

2. 在导航窗格中，选择支FaultTolerance票类别。
3. 在类别页面上，您可以查看每种检查类别的摘要：
   - 建议的操作（红色）- Trusted Advisor 建议检查的操作。
   - 建议调查（黄色）： Trusted Advisor 检测到检查存在的可能问题。
   - 未检测到问题（绿色）- Trusted Advisor 未检测到检查问题。
   - 排除的项目（灰色） – 包含排除项目的检查数，例如您希望检查忽略的资源。
4. 选择 Amazon Comprehend 终端节点访问风险检查，即可查看检查说明和以下详细信息：
   - 提示标准：描述检查将更改状态的阈值。
   - 建议的操作 – 描述此检查的建议操作。
   - 资源表：此表列出了您的 KMS 加密终端节点详细信息以及每个终端节点的状态，具体取决于是否有建议的操作。
5. 在资源表中，如果终端节点被标记为 “操作推荐” 状态，请选择 KMS KeyId 列中的链接，您将被重定向到相应的 AWS KMS 密钥页面。
   - 要启用已禁用的 AWS KMS 密钥，请选择按键操作，然后选择启用。
   - 如果密钥状态列为已启用，请在密钥策略部分中选择切换到策略视图来更新密钥策略。编辑密钥策略文档以向 Amazon Comprehend 提供必要的权限，然后选择保存更改。

以下是 Trusted Advisor 控制台上 FaultTolerance 类别视图的示例：



也可以通过参阅 AWS Support API 的 Trusted Advisor 部分来查看这些检查及其结果。

要了解有关使用设置警报的更多信息 CloudWatch，请参阅：使用创建 Trusted Advisor 警报 CloudWatch。有关完整 Trusted Advisor 的最佳实践检查，请参阅：AWS Trusted Advisor 最佳实践清单。

正在删除 Amazon Comprehend 终端节点

一旦您不再需要您的终端节点，就应该将其删除，这样您就不会因此而产生费用。您可以随时从终端节点部分轻松创建另一个终端节点。

删除终端节点 (控制台)

1. 登录 AWS Management Console 并打开 Amazon Comprehend 控制台，网址：https://console.aws.amazon.com/comprehend/
2. 从左侧菜单中，选择终端节点。
3. 从终端节点列表中，找到您要删除的终端节点。您可以搜索或筛选所有终端节点以找到所需的终端节点。

4. 选中要删除的终端节点对应的终端节点复选框。在终端节点列表的右上角，选择操作图标。
5. 选择 删除。
6. 再次选择删除以确认删除。显示终端节点页面。确认您删除的终端节点旁边显示正在删除。删除后，终端节点将从终端节点列表中删除。

删除终端节点 (AWS CLI)

以下示例演示了如何在 AWS CLI 中使用该DeleteEndpoint操作。

此示例的格式适用于 Unix、Linux 和 macOS。对于 Windows，请将每行末尾的反斜杠 (\) Unix 行继续符替换为脱字号 (^)。

```
aws comprehend delete-endpoint \
    --endpoint-arn arn:aws:comprehend:region:account-id endpoint/endpoint name
```

如果此操作成功，则 Amazon Comprehend 会发送回带有空 HTTP 正文的 HTTP 200 响应。

使用终端节点自动扩缩

您可以使用自动扩缩自动设置终端节点配置以满足您的容量需求，而无需手动调整为文档分类终端节点和实体识别器终端节点配置的推理单元数量。

使用自动扩缩调整为终端节点预置的推理单元数量有两种方法：

- 目标跟踪：设置自动扩缩以根据使用情况调整终端节点配置以满足容量需求。
- 计划扩展：设置自动扩缩以按照指定计划调整终端节点配置以满足容量需求。

只能使用 AWS Command Line Interface (AWS CLI) 设置自动扩缩。有关自动扩缩的更多信息，请参阅什么是应用程序自动扩缩？

目标跟踪

通过目标跟踪，您可以根据使用情况调整终端节点配置，以满足您的容量需求。推理单元的数量会自动调整，使已利用容量保持在预配置容量的目标百分比之内。您可以使用目标跟踪来适应文档分类终端节点和实体识别器终端节点的临时使用激增。有关更多信息，请参阅 Application Auto Scaling 的目标跟踪扩缩策略。

> **Note**
>
> 以下示例的格式适用于 Unix、Linux 和 macOS。对于 Windows，请将每行末尾的反斜杠 (\) Unix 行继续符替换为脱字号 (^)。

设置目标跟踪

要为终端节点设置目标跟踪，您可以使用 AWS CLI 命令注册可扩展目标，然后创建扩缩策略。可扩展目标将推理单元定义为用于调整终端节点配置的资源，扩缩策略定义了控制预配置容量自动扩缩的指标。

设置目标跟踪

1. 注册可扩展目标。以下示例注册了一个可扩展的目标，以调整终端节点配置，其最小容量为 1 个推理单元，最大容量为 2 个推理单元。

   对于文档分类终端节点，请使用 AWS CLI 命令：

   ```
   aws application-autoscaling register-scalable-target \
       --service-namespace comprehend \
       --resource-id arn:aws:comprehend:region:account-id:document-classifier-
   endpoint/name \
       --scalable-dimension comprehend:document-classifier-
   endpoint:DesiredInferenceUnits \
       --min-capacity 1 \
       --max-capacity 2
   ```

   对于实体识别器终端节点，请使用以下 AWS CLI 命令：

   ```
   aws application-autoscaling register-scalable-target \
       --service-namespace comprehend \
       --resource-id arn:aws:comprehend:region:account-id:entity-recognizer-
   endpoint/name \
       --scalable-dimension comprehend:entity-recognizer-
   endpoint:DesiredInferenceUnits \
       --min-capacity 1 \
       --max-capacity 2
   ```

2. 要验证可扩展目标的注册，请使用以下 AWS CLI 命令：

```
aws application-autoscaling describe-scalable-targets \
    --service-namespace comprehend \
    --resource-id endpoint ARN
```

3. 为扩缩策略创建目标跟踪配置，并将该配置保存在名为 config.json 的文件中。以下是目标跟踪配置的示例，该配置会自动调整推理单元的数量，使已利用容量始终为预配置容量的 70%。

```
{
  "TargetValue": 70,
  "PredefinedMetricSpecification":
  {
  "PredefinedMetricType": "ComprehendInferenceUtilization"
  }
}
```

4. 创建扩缩策略。以下示例根据 config.json 文件中定义的目标跟踪配置创建扩缩策略。

   对于文档分类终端节点，请使用 AWS CLI 命令：

```
aws application-autoscaling put-scaling-policy \
    --service-namespace comprehend \
    --resource-id arn:aws:comprehend:region:account-id:document-classifier-
endpoint/name \
    --scalable-dimension comprehend:document-classifier-
endpoint:DesiredInferenceUnits \
    --policy-name TestPolicy \
    --policy-type TargetTrackingScaling \
    --target-tracking-scaling-policy-configuration file://config.json
```

   对于实体识别器终端节点，请使用以下 AWS CLI 命令：

```
aws application-autoscaling put-scaling-policy \
    --service-namespace comprehend \
    --resource-id arn:aws:comprehend:region:account-id:entity-recognizer-
endpoint/name \
    --scalable-dimension comprehend:entity-recognizer-
endpoint:DesiredInferenceUnits \
    --policy-name TestPolicy \
    --policy-type TargetTrackingScaling \
```

```
    --target-tracking-scaling-policy-configuration file://config.json
```

正在删除目标跟踪

要为终端节点删除目标跟踪，您可以使用 AWS CLI 命令删除扩缩策略，然后注销可扩展目标。

删除目标跟踪

1. 删除扩缩策略。以下示例删除了指定扩缩策略。

   对于文档分类终端节点，请使用 AWS CLI 命令：

   ```
   aws application-autoscaling delete-scaling-policy \
       --service-namespace comprehend \
       --resource-id arn:aws:comprehend:region:account-id:document-classifier-
   endpoint/name \
       --scalable-dimension comprehend:document-classifier-
   endpoint:DesiredInferenceUnits \
       --policy-name TestPolicy \
   ```

   对于实体识别器终端节点，请使用以下 AWS CLI 命令：

   ```
   aws application-autoscaling delete-scaling-policy \
       --service-namespace comprehend \
       --resource-id arn:aws:comprehend:region:account-id:entity-recognizer-
   endpoint/name \
       --scalable-dimension comprehend:entity-recognizer-
   endpoint:DesiredInferenceUnits \
       --policy-name TestPolicy
   ```

2. 取消注册可扩展目标。以下示例将注销指定的可扩展目标。

   对于文档分类终端节点，请使用 AWS CLI 命令：

   ```
   aws application-autoscaling deregister-scalable-target \
       --service-namespace comprehend \
       --resource-id arn:aws:comprehend:region:account-id:document-classifier-
   endpoint/name \
   ```

```
    --scalable-dimension comprehend:document-classifier-
endpoint:DesiredInferenceUnits
```

对于实体识别器终端节点，请使用以下 AWS CLI 命令：

```
aws application-autoscaling deregister-scalable-target \
    --service-namespace comprehend \
    --resource-id arn:aws:comprehend:region:account-id:entity-recognizer-
endpoint/name \
    --scalable-dimension comprehend:entity-recognizer-
endpoint:DesiredInferenceUnits
```

计划扩展

通过计划扩缩，您可以调整终端节点的配置以适应指定计划的容量需求。计划扩缩会自动调整推理单元的数量，以适应特定时间的使用量激增。您可以对文档分类终端节点和实体识别器终端节点使用计划扩缩。有关计划扩缩的更多信息，请参阅应用程序自动扩缩中的计划扩缩。

> **Note**
>
> 以下示例的格式适用于 Unix、Linux 和 macOS。对于 Windows，请将每行末尾的反斜杠 (\) Unix 行继续符替换为脱字号 (^)。

设置计划扩缩

要为终端节点设置计划扩缩，您可以使用 AWS CLI 命令注册可扩展目标，然后创建计划操作。可扩展目标将推理单元定义为用于调整终端节点配置的资源，而计划操作控制了预配置容量在特定时间的自动扩缩。

设置计划扩缩

1. 注册可扩展目标。以下示例注册了一个可扩展的目标，以调整终端节点配置，其最小容量为 1 个推理单元，最大容量为 2 个推理单元。

   对于文档分类终端节点，请使用 AWS CLI 命令：

   ```
   aws application-autoscaling register-scalable-target \
       --service-namespace comprehend \
   ```

```
    --resource-id arn:aws:comprehend:region:account-id:document-classifier-
endpoint/name \
    --scalable-dimension comprehend:document-classifier-
endpoint:DesiredInferenceUnits \
    --min-capacity 1 \
    --max-capacity 2
```

对于实体识别器终端节点，请使用以下 AWS CLI 命令：

```
aws application-autoscaling register-scalable-target \
    --service-namespace comprehend \
    --resource-id arn:aws:comprehend:region:account-id:entity-recognizer-
endpoint/name \
    --scalable-dimension comprehend:entity-recognizer-
endpoint:DesiredInferenceUnits \
    --min-capacity 1 \
    --max-capacity 2
```

2. 创建计划操作。以下示例创建了一个计划操作，以在每天 12:00 UTC 自动调整预配置容量，最小为 2 个推理单元，最大为 5 个推理单元。有关时间表达式和计划扩缩的更多信息，请参阅计划表达式。

   对于文档分类终端节点，请使用 AWS CLI 命令：

```
aws application-autoscaling put-scheduled-action \
    --service-namespace comprehend \
    --resource-id arn:aws:comprehend:region:account-id:document-classifier-
endpoint/name \
    --scalable-dimension comprehend:document-classifier-
endpoint:DesiredInferenceUnits \
    --scheduled-action-name TestScheduledAction \
    --schedule "cron(0 12 * * ? *)" \
    --scalable-target-action MinCapacity=2,MaxCapacity=5
```

   对于实体识别器终端节点，请使用以下 AWS CLI 命令：

```
aws application-autoscaling put-scheduled-action \
    --service-namespace comprehend \
    --resource-id arn:aws:comprehend:region:account-id:entity-recognizer-
endpoint/name \
    --scalable-dimension comprehend:entity-recognizer-
endpoint:DesiredInferenceUnits \
```

```
    --scheduled-action-name TestScheduledAction \
    --schedule "cron(0 12 * * ? *)" \
    --scalable-target-action MinCapacity=2,MaxCapacity=5
```

删除计划扩缩

要为终端节点删除计划扩缩，您可以使用 AWS CLI 命令删除计划操作，然后注销可扩展目标。

删除计划扩缩

1. 删除计划操作。以下示例将删除指定的计划操作。

   对于文档分类终端节点，请使用 AWS CLI 命令：

   ```
   aws application-autoscaling delete-scheduled-action \
       --service-namespace comprehend \
       --resource-id arn:aws:comprehend:region:account-id:document-classifier-
   endpoint/name \
       --scalable-dimension comprehend:document-classifier-
   endpoint:DesiredInferenceUnits \
       --scheduled-action-name TestScheduledAction
   ```

   对于实体识别器终端节点，请使用以下 AWS CLI 命令：

   ```
   aws application-autoscaling delete-scheduled-action \
       --service-namespace comprehend \
       --resource-id arn:aws:comprehend:region:account-id:entity-recognizer-
   endpoint/name \
       --scalable-dimension comprehend:entity-recognizer-
   endpoint:DesiredInferenceUnits \
       --scheduled-action-name TestScheduledAction
   ```

2. 取消注册可扩展目标。以下示例将注销指定的可扩展目标。

   对于文档分类终端节点，请使用 AWS CLI 命令：

   ```
   aws application-autoscaling deregister-scalable-target \
       --service-namespace comprehend \
       --resource-id arn:aws:comprehend:region:account-id:document-classifier-
   endpoint/name \
   ```

```
    --scalable-dimension comprehend:document-classifier-
endpoint:DesiredInferenceUnits
```

对于实体识别器终端节点，请使用以下 AWS CLI 命令：

```
aws application-autoscaling deregister-scalable-target \
    --service-namespace comprehend \
    --resource-id arn:aws:comprehend:region:account-id:entity-recognizer-
endpoint/name \
    --scalable-dimension comprehend:entity-recognizer-
endpoint:DesiredInferenceUnits
```

标记您的资源

标签是您可以将其作为元数据添加到 Amazon Comprehend 资源的键值对。您可以在分析作业、自定义分类模型、自定义实体识别模型或终端节点上使用标签。标签有两个主要功能：组织您的资源和提供基于标签的访问控制。

要使用标签组织资源，您可以添加标签键“部门”和标签值“销售”或“法律”。然后，您可以搜索和筛选与贵公司法律部门相关的资源。

要提供基于标签的访问控制，请创建具有基于标签的权限的 IAM policy。策略可以根据您的请求中提供的标签（请求标签）或与您正在调用的资源关联的标签（资源标签）来允许或禁止某项操作。有关在 IAM 中使用标签的更多信息，请参阅《IAM 用户指南》中的使用标签控制访问。

在 Amazon Comprehend 中使用标签的注意事项：

- 您最多可以为每个资源添加 50 个标签，并且可以在创建资源时添加标签，也可以追溯添加标签。
- 标签键是必填字段，但标签值是可选的。
- 资源之间的标签不必是唯一的，但是给定资源不能有重复的标签键。
- 标签键和值区分大小写。
- 一个标签键最多可以有 127 个字符；一个标签值最多可以有 255 个字符。
- 保留“aws:”前缀以供 AWS 使用；您无法添加、编辑或删除其键以 aws: 开头的标签。这些标签不计入你的 50 个上 tags-per-resource 限。

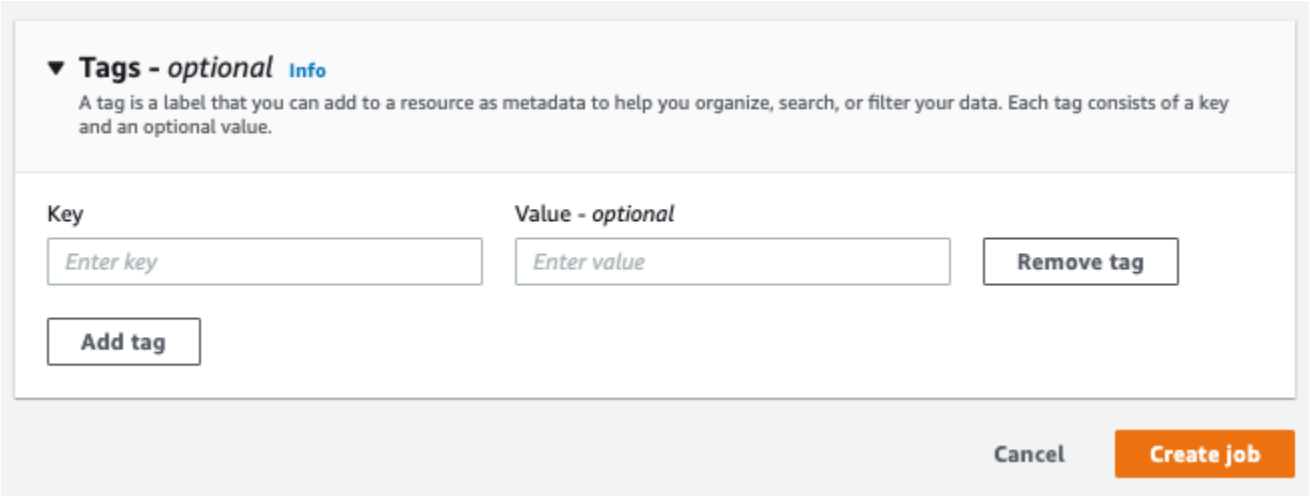
> **Note**
>
> 如果您计划在多个 AWS 服务和资源中使用添加标签方案，请记得其他服务可能对允许使用的字符有不同要求。

主题

标记新的资源

您可以将标签添加到分析作业、自定义分类模型、自定义实体识别模型或终端节点。

1. 登录 AWS Management Console 并打开亚马逊 Comprehend 控制台，网址为 https://console.aws.amazon.com/comprehend/
2. 从左侧导航窗格中选择要创建的资源（“分析作业”、“自定义分类”或“自定义实体识别”）。
3. 单击创建作业（或创建新模型）。这会将您引导至资源的主“创建”页面。在此页面的底部，你会看到一个 “标签- 可选” 面板。



输入一个标签键和可选的标签值。选择添加标签为资源添加另一个标签。重复此过程，直到添加完所有标签。请注意，每个资源的标签键必须是唯一的。

4. 选择创建或创建作业按钮以继续创建您的资源。

您也可以使用 AWS CLI 添加标签。此示例说明如何使用start-entities-detection-job命令添加标签。

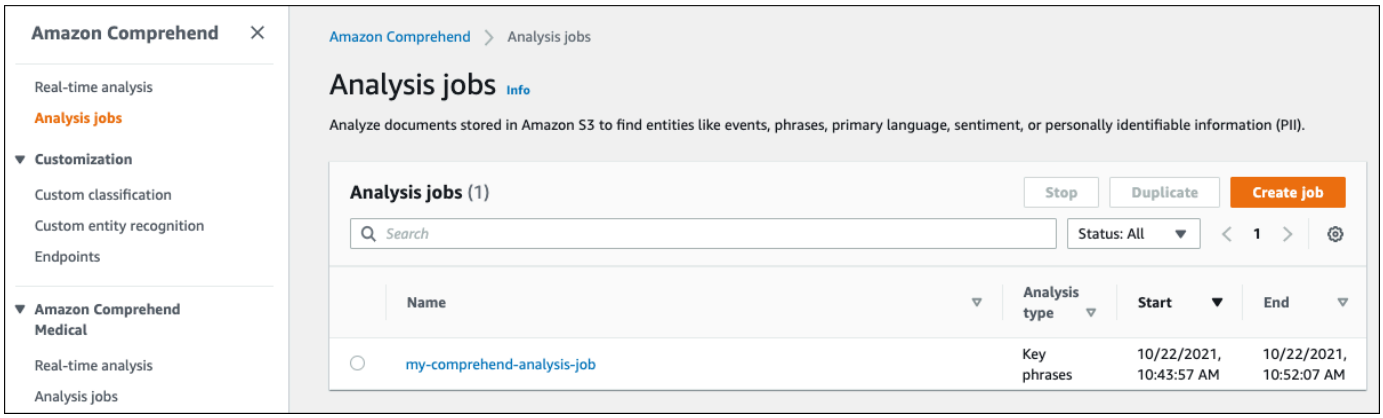
```
aws comprehend start-entities-detection-job \
--language-code "en" \
--input-data-config "{\"S3Uri\": \"s3://test-input/TEST.csv\"}" \
--output-data-config "{\"S3Uri\": \"s3://test-output\"}" \
--data-access-role-arn arn:aws:iam::123456789012:role/test \
--tags "[{\"Key\": \"color\",\"Value\": \"orange\"}]"
```

查看、编辑和删除与资源关联的标签

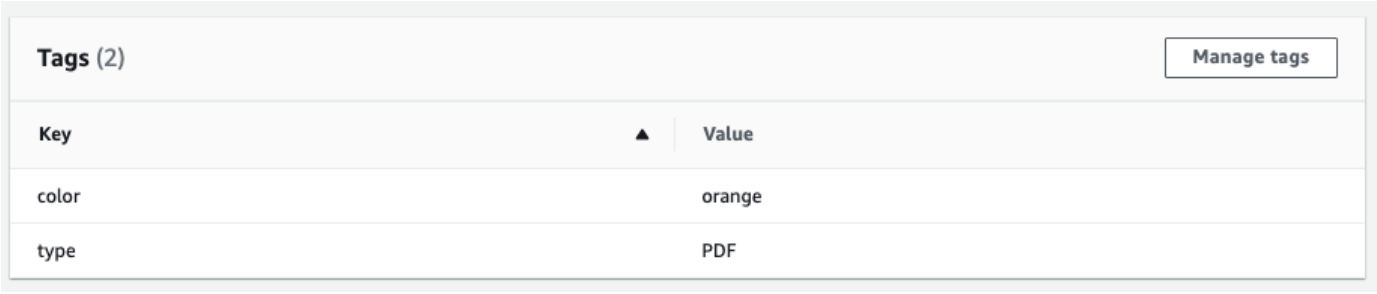
您可以查看与分析作业、自定义分类模型或自定义实体识别模型关联的标签。

1. 登录 AWS Management Console 并打开亚马逊 Comprehend 控制台，网址为 https://console.aws.amazon.com/comprehend/

2. 选择包含要查看、修改或删除的标签的文件的资源（分析作业、自定义分类或自定义实体识别）。这将显示您所选资源的现有文件列表。

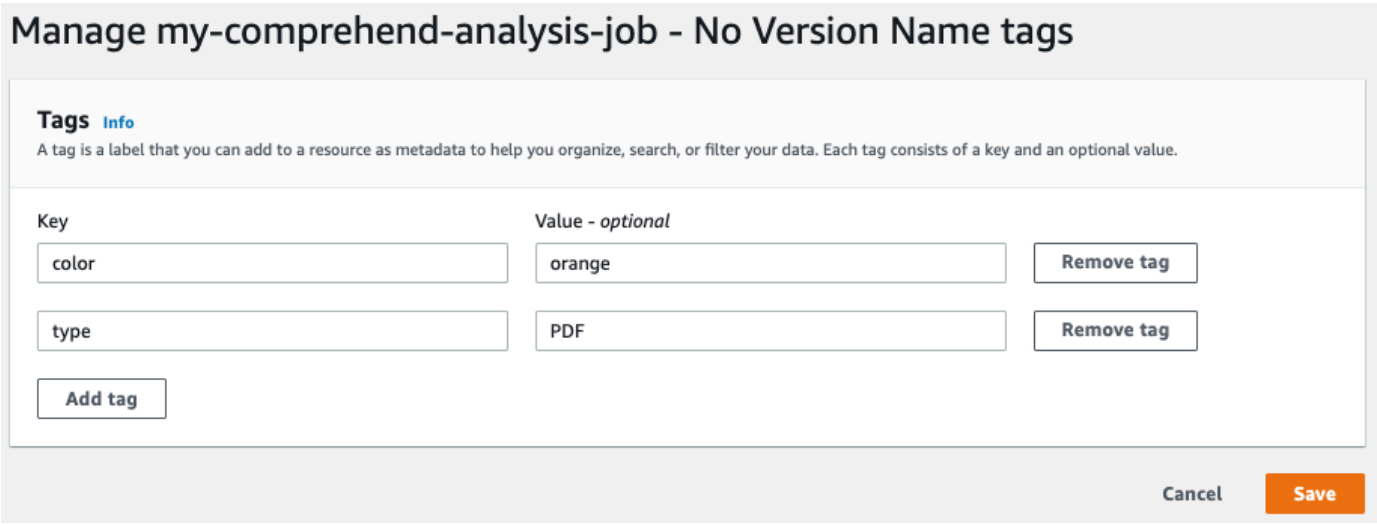


3. 单击要查看、修改或删除其标签的文件（或模型）的名称。这会将您转到该文件（或模型）的详细信息页面。向下滚动，直到看到标签框。在这里，您可以看到与所选文件（或模型）关联的所有标签。



选择管理标签以编辑或删除资源中的标签。

4. 点击要修改的文本，然后编辑标签。您也可以通过选择删除标签来删除标签。要添加新标签，请选择添加标签，然后在空白字段中输入所需的文本。



修改完标签后，选择保存。

使用软件开发工具包的 Amazon Comprehend 的代码示例 AWS

以下代码示例展示了如何将 Amazon Comprehend 与软件开发套件 (SDK) AWS 配合使用。

操作是大型程序的代码摘录，必须在上下文中运行。您可以通过操作了解如何调用单个服务函数，还可以通过函数相关场景和跨服务示例的上下文查看操作。

场景 是展示如何通过在同一服务中调用多个函数来完成特定任务的代码示例。

跨服务示例是指跨多个 AWS 服务工作的示例应用程序。

有关 S AWS DK 开发者指南和代码示例的完整列表，请参阅将 Amazon Comprehend 与 SDK 配合 AWS 使用。本主题还包括有关入门的信息以及有关先前的 SDK 版本的详细信息。

代码示例

- 使用软件开发工具包对 Amazon Comprehend 执行的操作 AWS
  - CreateDocumentClassifier与 AWS SDK 或 CLI 配合使用
  - DeleteDocumentClassifier与 AWS SDK 或 CLI 配合使用
  - DescribeDocumentClassificationJob与 AWS SDK 或 CLI 配合使用
  - DescribeDocumentClassifier与 AWS SDK 或 CLI 配合使用
  - DescribeTopicsDetectionJob与 AWS SDK 或 CLI 配合使用
  - DetectDominantLanguage与 AWS SDK 或 CLI 配合使用
  - DetectEntities与 AWS SDK 或 CLI 配合使用
  - DetectKeyPhrases与 AWS SDK 或 CLI 配合使用
  - DetectPiiEntities与 AWS SDK 或 CLI 配合使用
  - DetectSentiment与 AWS SDK 或 CLI 配合使用
  - DetectSyntax与 AWS SDK 或 CLI 配合使用
  - ListDocumentClassificationJobs与 AWS SDK 或 CLI 配合使用
  - ListDocumentClassifiers与 AWS SDK 或 CLI 配合使用
  - ListTopicsDetectionJobs与 AWS SDK 或 CLI 配合使用
  - StartDocumentClassificationJob与 AWS SDK 或 CLI 配合使用
  - StartTopicsDetectionJob与 AWS SDK 或 CLI 配合使用
- 使用软件开发工具包的 Amazon Comprehend 场景 AWS

- 使用 Amazon Comprehend 和软件开发工具包检测文档元素 AWS
- 使用软件开发工具包对示例数据运行 Amazon Comprehend 主题建模作业 AWS
- 训练自定义 Amazon Comprehend 分类器并使用软件开发工具包对文档进行分类 AWS

- 使用软件开发工具包的 Amazon Comprehend 的跨服务示例 AWS
  - 构建 Amazon Transcribe 流式传输应用程序
  - 创建 Amazon Lex 聊天机器人来吸引您的网站访客
  - 使用 Amazon SQS 创建用于发送和检索消息的网络应用程序
  - 创建用于分析客户反馈和合成音频的应用程序
  - 使用 AWS SDK 检测从图像中提取的文本中的实体

使用软件开发工具包对 Amazon Comprehend 执行的操作 AWS

以下代码示例演示了如何使用软件开发工具包执行单个 Amazon Compre AWS hend 操作。这些代码节选调用了 Amazon Comprehend API，是必须在上下文中运行的大型程序的代码节选。每个示例都包含一个指向的链接 GitHub，您可以在其中找到有关设置和运行代码的说明。

以下示例仅包括最常用的操作。有关完整列表，请参阅 Amazon Comprehend API 参考。

示例

- CreateDocumentClassifier与 AWS SDK 或 CLI 配合使用
- DeleteDocumentClassifier与 AWS SDK 或 CLI 配合使用
- DescribeDocumentClassificationJob与 AWS SDK 或 CLI 配合使用
- DescribeDocumentClassifier与 AWS SDK 或 CLI 配合使用
- DescribeTopicsDetectionJob与 AWS SDK 或 CLI 配合使用
- DetectDominantLanguage与 AWS SDK 或 CLI 配合使用
- DetectEntities与 AWS SDK 或 CLI 配合使用
- DetectKeyPhrases与 AWS SDK 或 CLI 配合使用
- DetectPiiEntities与 AWS SDK 或 CLI 配合使用
- DetectSentiment与 AWS SDK 或 CLI 配合使用
- DetectSyntax与 AWS SDK 或 CLI 配合使用
- ListDocumentClassificationJobs与 AWS SDK 或 CLI 配合使用
- ListDocumentClassifiers与 AWS SDK 或 CLI 配合使用

- ListTopicsDetectionJobs与 AWS SDK 或 CLI 配合使用
- StartDocumentClassificationJob与 AWS SDK 或 CLI 配合使用
- StartTopicsDetectionJob与 AWS SDK 或 CLI 配合使用

`CreateDocumentClassifier`与 AWS SDK 或 CLI 配合使用

以下代码示例演示如何使用 `CreateDocumentClassifier`。

操作示例是大型程序的代码摘录，必须在上下文中运行。在以下代码示例中，您可以查看此操作的上下文：

- 训练自定义分类器并对文档进行分类

CLI

AWS CLI

创建文档分类器对文档进行分类

以下 `create-document-classifier` 示例启动文档分类器模型的训练过程。训练数据文件 `training.csv` 位于 `--input-data-config` 标签处。`training.csv` 是一个两列文档，其中第一列提供标签或分类，第二列提供文档。

```
aws comprehend create-document-classifier \
    --document-classifier-name example-classifier \
    --data-access-arn arn:aws:comprehend:us-west-2:111122223333:pii-entities-
detection-job/123456abcdeb0e11022f22a11EXAMPLE \
    --input-data-config "S3Uri=s3://DOC-EXAMPLE-BUCKET/" \
    --language-code en
```

输出：

```
{
    "DocumentClassifierArn": "arn:aws:comprehend:us-west-2:111122223333:document-
classifier/example-classifier"
}
```

有关更多信息，请参阅《Amazon Comprehend 开发人员指南》中的自定义分类。

- 有关 API 的详细信息，请参阅AWS CLI 命令参考CreateDocumentClassifier中的。

Java

适用于 Java 2.x 的 SDK

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.comprehend.ComprehendClient;
import software.amazon.awssdk.services.comprehend.model.ComprehendException;
import
 software.amazon.awssdk.services.comprehend.model.CreateDocumentClassifierRequest;
import
 software.amazon.awssdk.services.comprehend.model.CreateDocumentClassifierResponse;
import
 software.amazon.awssdk.services.comprehend.model.DocumentClassifierInputDataConfig;

/**
 * Before running this code example, you can setup the necessary resources, such
 * as the CSV file and IAM Roles, by following this document:
 * https://aws.amazon.com/blogs/machine-learning/building-a-custom-classifier-
using-amazon-comprehend/
 *
 * Also, set up your development environment, including your credentials.
 *
 * For more information, see the following documentation topic:
 *
 * https://docs.aws.amazon.com/sdk-for-java/latest/developer-guide/get-
started.html
 */
public class DocumentClassifierDemo {
    public static void main(String[] args) {
        final String usage = """

                Usage:    <dataAccessRoleArn> <s3Uri> <documentClassifierName>

                Where:
                  dataAccessRoleArn - The ARN value of the role used for this
 operation.
```

```
                  s3Uri - The Amazon S3 bucket that contains the CSV file.
                  documentClassifierName - The name of the document classifier.
                """";

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

        String dataAccessRoleArn = args[0];
        String s3Uri = args[1];
        String documentClassifierName = args[2];

        Region region = Region.US_EAST_1;
        ComprehendClient comClient = ComprehendClient.builder()
                .region(region)
                .build();

        createDocumentClassifier(comClient, dataAccessRoleArn, s3Uri,
documentClassifierName);
        comClient.close();
    }

    public static void createDocumentClassifier(ComprehendClient comClient,
String dataAccessRoleArn, String s3Uri,
            String documentClassifierName) {
        try {
            DocumentClassifierInputDataConfig config =
DocumentClassifierInputDataConfig.builder()
                    .s3Uri(s3Uri)
                    .build();

            CreateDocumentClassifierRequest createDocumentClassifierRequest =
CreateDocumentClassifierRequest.builder()
                    .documentClassifierName(documentClassifierName)
                    .dataAccessRoleArn(dataAccessRoleArn)
                    .languageCode("en")
                    .inputDataConfig(config)
                    .build();

            CreateDocumentClassifierResponse createDocumentClassifierResult =
comClient
                    .createDocumentClassifier(createDocumentClassifierRequest);
```

```
            String documentClassifierArn =
 createDocumentClassifierResult.documentClassifierArn();
            System.out.println("Document Classifier ARN: " +
 documentClassifierArn);

        } catch (ComprehendException e) {
            System.err.println(e.awsErrorDetails().errorMessage());
            System.exit(1);
        }
    }
}
```

- 有关 API 的详细信息，请参阅 AWS SDK for Java 2.x API 参考[CreateDocumentClassifier](#)中的。

Python

SDK for Python (Boto3)

> **Note**
>
> 还有更多相关信息 GitHub。在 [AWS 代码示例存储库](#)中查找完整示例，了解如何进行设置和运行。

```
class ComprehendClassifier:
    """Encapsulates an Amazon Comprehend custom classifier."""

    def __init__(self, comprehend_client):
        """
        :param comprehend_client: A Boto3 Comprehend client.
        """
        self.comprehend_client = comprehend_client
        self.classifier_arn = None


    def create(
        self,
        name,
        language_code,
```

```
        training_bucket,
        training_key,
        data_access_role_arn,
        mode,
    ):
        """
        Creates a custom classifier. After the classifier is created, it
immediately
        starts training on the data found in the specified Amazon S3 bucket.
Training
        can take 30 minutes or longer. The `describe_document_classifier`
function
        can be used to get training status and returns a status of TRAINED when
the
        classifier is ready to use.

        :param name: The name of the classifier.
        :param language_code: The language the classifier can operate on.
        :param training_bucket: The Amazon S3 bucket that contains the training
data.
        :param training_key: The prefix used to find training data in the
training
                             bucket. If multiple objects have the same prefix,
all
                             of them are used.
        :param data_access_role_arn: The Amazon Resource Name (ARN) of a role
that
                                     grants Comprehend permission to read from
the
                                     training bucket.
        :return: The ARN of the newly created classifier.
        """
        try:
            response = self.comprehend_client.create_document_classifier(
                DocumentClassifierName=name,
                LanguageCode=language_code,
                InputDataConfig={"S3Uri": f"s3://{training_bucket}/
{training_key}"},
                DataAccessRoleArn=data_access_role_arn,
                Mode=mode.value,
            )
            self.classifier_arn = response["DocumentClassifierArn"]
            logger.info("Started classifier creation. Arn is: %s.",
self.classifier_arn)
```

```
        except ClientError:
            logger.exception("Couldn't create classifier %s.", name)
            raise
        else:
            return self.classifier_arn
```

- 有关 API 的详细信息，请参阅适用CreateDocumentClassifier于 Python 的AWS SDK (Boto3) API 参考。

有关 S AWS DK 开发者指南和代码示例的完整列表，请参阅将 Amazon Comprehend 与 SDK 配合 AWS 使用。本主题还包括有关入门的信息以及有关先前的 SDK 版本的详细信息。

`DeleteDocumentClassifier`与 AWS SDK 或 CLI 配合使用

以下代码示例演示如何使用 `DeleteDocumentClassifier`。

操作示例是大型程序的代码摘录，必须在上下文中运行。在以下代码示例中，您可以查看此操作的上下文：

- 训练自定义分类器并对文档进行分类

CLI

AWS CLI

删除自定义文档分类器

以下 `delete-document-classifier` 示例删除了自定义文档分类器模型。

```
aws comprehend delete-document-classifier \
    --document-classifier-arn arn:aws:comprehend:us-west-2:111122223333:document-classifier/example-classifier-1
```

此命令不生成任何输出。

有关更多信息，请参阅《Amazon Comprehend 开发人员指南》中的管理 Amazon Comprehend 端点。

- 有关 API 的详细信息，请参阅AWS CLI 命令参考DeleteDocumentClassifier中的。

Python

SDK for Python (Boto3)

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
class ComprehendClassifier:
    """Encapsulates an Amazon Comprehend custom classifier."""

    def __init__(self, comprehend_client):
        """
        :param comprehend_client: A Boto3 Comprehend client.
        """
        self.comprehend_client = comprehend_client
        self.classifier_arn = None


    def delete(self):
        """
        Deletes the classifier.
        """
        try:
            self.comprehend_client.delete_document_classifier(
                DocumentClassifierArn=self.classifier_arn
            )
            logger.info("Deleted classifier %s.", self.classifier_arn)
            self.classifier_arn = None
        except ClientError:
            logger.exception("Couldn't deleted classifier %s.",
 self.classifier_arn)
            raise
```

- 有关 API 的详细信息，请参阅适用DeleteDocumentClassifier于 Python 的AWS SDK (Boto3) API 参考。

有关 S AWS DK 开发者指南和代码示例的完整列表，请参阅将 Amazon Comprehend 与 SDK 配合 AWS 使用。本主题还包括有关入门的信息以及有关先前的 SDK 版本的详细信息。

DescribeDocumentClassificationJob与 AWS SDK 或 CLI 配合使用

以下代码示例演示如何使用 DescribeDocumentClassificationJob。

操作示例是大型程序的代码摘录，必须在上下文中运行。在以下代码示例中，您可以查看此操作的上下文：

- 训练自定义分类器并对文档进行分类

CLI

AWS CLI

描述文档分类作业

以下 describe-document-classification-job 示例将获取异步文档分类作业的属性。

```
aws comprehend describe-document-classification-job \
    --job-id 123456abcdeb0e11022f22a11EXAMPLE
```

输出：

```
{
    "DocumentClassificationJobProperties": {
        "JobId": "123456abcdeb0e11022f22a11EXAMPLE",
        "JobArn": "arn:aws:comprehend:us-west-2:111122223333:document-
classification-job/123456abcdeb0e11022f22a11EXAMPLE",
        "JobName": "exampleclassificationjob",
        "JobStatus": "COMPLETED",
        "SubmitTime": "2023-06-14T17:09:51.788000+00:00",
        "EndTime": "2023-06-14T17:15:58.582000+00:00",
        "DocumentClassifierArn": "arn:aws:comprehend:us-
west-2:111122223333:document-classifier/mymodel/version/1",
        "InputDataConfig": {
            "S3Uri": "s3://DOC-EXAMPLE-BUCKET/jobdata/",
            "InputFormat": "ONE_DOC_PER_LINE"
        },
        "OutputDataConfig": {
```

```
            "S3Uri": "s3://DOC-EXAMPLE-DESTINATION-BUCKET/
testfolder/111122223333-CLN-123456abcdeb0e11022f22a11EXAMPLE/output/
output.tar.gz"
        },
        "DataAccessRoleArn": "arn:aws:iam::111122223333:role/service-role/
AmazonComprehendServiceRole-servicerole"
    }
}
```

有关更多信息，请参阅《Amazon Comprehend 开发人员指南》中的自定义分类。

- 有关 API 的详细信息，请参阅AWS CLI 命令参考DescribeDocumentClassificationJob中的。

Python

SDK for Python (Boto3)

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
class ComprehendClassifier:
    """Encapsulates an Amazon Comprehend custom classifier."""

    def __init__(self, comprehend_client):
        """
        :param comprehend_client: A Boto3 Comprehend client.
        """
        self.comprehend_client = comprehend_client
        self.classifier_arn = None


    def describe_job(self, job_id):
        """
        Gets metadata about a classification job.

        :param job_id: The ID of the job to look up.
        :return: Metadata about the job.
        """
        try:
```

```
            response =
self.comprehend_client.describe_document_classification_job(
                JobId=job_id
            )
            job = response["DocumentClassificationJobProperties"]
            logger.info("Got classification job %s.", job["JobName"])
        except ClientError:
            logger.exception("Couldn't get classification job %s.", job_id)
            raise
        else:
            return job
```

- 有关 API 的详细信息，请参阅适用DescribeDocumentClassificationJob于 Python 的AWS SDK (Boto3) API 参考。

有关 S AWS DK 开发者指南和代码示例的完整列表，请参阅将 Amazon Comprehend 与 SDK 配合 AWS 使用。本主题还包括有关入门的信息以及有关先前的 SDK 版本的详细信息。

`DescribeDocumentClassifier`与 AWS SDK 或 CLI 配合使用

以下代码示例演示如何使用 `DescribeDocumentClassifier`。

操作示例是大型程序的代码摘录，必须在上下文中运行。在以下代码示例中，您可以查看此操作的上下文：

- 训练自定义分类器并对文档进行分类

CLI

AWS CLI

描述文档分类器

以下 `describe-document-classifier` 示例将获取自定义文档分类器模型的属性。

```
aws comprehend describe-document-classifier \
    --document-classifier-arn arn:aws:comprehend:us-west-2:111122223333:document-
classifier/example-classifier-1
```

输出：

```
{
    "DocumentClassifierProperties": {
        "DocumentClassifierArn": "arn:aws:comprehend:us-
west-2:111122223333:document-classifier/example-classifier-1",
        "LanguageCode": "en",
        "Status": "TRAINED",
        "SubmitTime": "2023-06-13T19:04:15.735000+00:00",
        "EndTime": "2023-06-13T19:42:31.752000+00:00",
        "TrainingStartTime": "2023-06-13T19:08:20.114000+00:00",
        "TrainingEndTime": "2023-06-13T19:41:35.080000+00:00",
        "InputDataConfig": {
            "DataFormat": "COMPREHEND_CSV",
            "S3Uri": "s3://DOC-EXAMPLE-BUCKET/trainingdata"
        },
        "OutputDataConfig": {},
        "ClassifierMetadata": {
            "NumberOfLabels": 3,
            "NumberOfTrainedDocuments": 5016,
            "NumberOfTestDocuments": 557,
            "EvaluationMetrics": {
                "Accuracy": 0.9856,
                "Precision": 0.9919,
                "Recall": 0.9459,
                "F1Score": 0.9673,
                "MicroPrecision": 0.9856,
                "MicroRecall": 0.9856,
                "MicroF1Score": 0.9856,
                "HammingLoss": 0.0144
            }
        },
        "DataAccessRoleArn": "arn:aws:iam::111122223333:role/service-role/
AmazonComprehendServiceRole-example-role",
        "Mode": "MULTI_CLASS"
    }
}
```

有关更多信息，请参阅《Amazon Comprehend 开发人员指南》中的创建和管理自定义模型。

- 有关 API 的详细信息，请参阅AWS CLI 命令参考DescribeDocumentClassifier中的。

Python

SDK for Python (Boto3)

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
class ComprehendClassifier:
    """Encapsulates an Amazon Comprehend custom classifier."""

    def __init__(self, comprehend_client):
        """
        :param comprehend_client: A Boto3 Comprehend client.
        """
        self.comprehend_client = comprehend_client
        self.classifier_arn = None


    def describe(self, classifier_arn=None):
        """
        Gets metadata about a custom classifier, including its current status.

        :param classifier_arn: The ARN of the classifier to look up.
        :return: Metadata about the classifier.
        """
        if classifier_arn is not None:
            self.classifier_arn = classifier_arn
        try:
            response = self.comprehend_client.describe_document_classifier(
                DocumentClassifierArn=self.classifier_arn
            )
            classifier = response["DocumentClassifierProperties"]
            logger.info("Got classifier %s.", self.classifier_arn)
        except ClientError:
            logger.exception("Couldn't get classifier %s.", self.classifier_arn)
            raise
        else:
            return classifier
```

- 有关 API 的详细信息，请参阅适用DescribeDocumentClassifier于 Python 的AWS SDK (Boto3) API 参考。

有关 S AWS DK 开发者指南和代码示例的完整列表，请参阅将 Amazon Comprehend 与 SDK 配合 AWS 使用。本主题还包括有关入门的信息以及有关先前的 SDK 版本的详细信息。

DescribeTopicsDetectionJob与 AWS SDK 或 CLI 配合使用

以下代码示例演示如何使用 DescribeTopicsDetectionJob。

操作示例是大型程序的代码摘录，必须在上下文中运行。在以下代码示例中，您可以查看此操作的上下文：

- 对示例数据运行主题建模任务

CLI

AWS CLI

描述主题检测作业

以下 describe-topics-detection-job 示例获取异步主题检测作业的属性。

```
aws comprehend describe-topics-detection-job \
    --job-id 123456abcdeb0e11022f22a11EXAMPLE
```

输出：

```
{
    "TopicsDetectionJobProperties": {
        "JobId": "123456abcdeb0e11022f22a11EXAMPLE",
        "JobArn": "arn:aws:comprehend:us-west-2:111122223333:topics-detection-
job/123456abcdeb0e11022f22a11EXAMPLE",
        "JobName": "example_topics_detection",
        "JobStatus": "IN_PROGRESS",
        "SubmitTime": "2023-06-09T18:44:43.414000+00:00",
        "InputDataConfig": {
            "S3Uri": "s3://DOC-EXAMPLE-BUCKET",
            "InputFormat": "ONE_DOC_PER_LINE"
```

```
        },
        "OutputDataConfig": {
            "S3Uri": "s3://DOC-EXAMPLE-DESTINATION-BUCKET/
testfolder/111122223333-TOPICS-123456abcdeb0e11022f22a11EXAMPLE/output/
output.tar.gz"
        },
        "NumberOfTopics": 10,
        "DataAccessRoleArn": "arn:aws:iam::111122223333:role/service-role/
AmazonComprehendServiceRole-examplerole"
    }
}
```

有关更多信息，请参阅《Amazon Comprehend 开发人员指南》中的 Amazon Comprehend 洞察的异步分析。

- 有关 API 的详细信息，请参阅AWS CLI 命令参考DescribeTopicsDetectionJob中的。

Python

SDK for Python (Boto3)

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
class ComprehendTopicModeler:
    """Encapsulates a Comprehend topic modeler."""

    def __init__(self, comprehend_client):
        """
        :param comprehend_client: A Boto3 Comprehend client.
        """
        self.comprehend_client = comprehend_client


    def describe_job(self, job_id):
        """
        Gets metadata about a topic modeling job.

        :param job_id: The ID of the job to look up.
```

```
        :return: Metadata about the job.
        """
        try:
            response = self.comprehend_client.describe_topics_detection_job(
                JobId=job_id
            )
            job = response["TopicsDetectionJobProperties"]
            logger.info("Got topic detection job %s.", job_id)
        except ClientError:
            logger.exception("Couldn't get topic detection job %s.", job_id)
            raise
        else:
            return job
```

- 有关 API 的详细信息，请参阅适用DescribeTopicsDetectionJob于 Python 的AWS SDK (Boto3) API 参考。

有关 S AWS DK 开发者指南和代码示例的完整列表，请参阅将 Amazon Comprehend 与 SDK 配合 AWS 使用。本主题还包括有关入门的信息以及有关先前的 SDK 版本的详细信息。

`DetectDominantLanguage`与 AWS SDK 或 CLI 配合使用

以下代码示例演示如何使用 `DetectDominantLanguage`。

操作示例是大型程序的代码摘录，必须在上下文中运行。在以下代码示例中，您可以查看此操作的上下文：

- 检测文档元素

.NET

AWS SDK for .NET

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
   using System;
   using System.Threading.Tasks;
   using Amazon.Comprehend;
   using Amazon.Comprehend.Model;

   /// <summary>
   /// This example calls the Amazon Comprehend service to determine the
   /// dominant language.
   /// </summary>
   public static class DetectDominantLanguage
   {
       /// <summary>
       /// Calls Amazon Comprehend to determine the dominant language used in
       /// the sample text.
       /// </summary>
       public static async Task Main()
       {
           string text = "It is raining today in Seattle.";

           var comprehendClient = new
AmazonComprehendClient(Amazon.RegionEndpoint.USWest2);

           Console.WriteLine("Calling DetectDominantLanguage\n");
           var detectDominantLanguageRequest = new
DetectDominantLanguageRequest()
           {
               Text = text,
           };

           var detectDominantLanguageResponse = await
comprehendClient.DetectDominantLanguageAsync(detectDominantLanguageRequest);
           foreach (var dl in detectDominantLanguageResponse.Languages)
           {
               Console.WriteLine($"Language Code: {dl.LanguageCode}, Score:
{dl.Score}");
           }

           Console.WriteLine("Done");
       }
   }
```

- 有关 API 的详细信息，请参阅 AWS SDK for .NET API 参考DetectDominantLanguage中的。

CLI

AWS CLI

检测输入文本的主要语言

以下 detect-dominant-language 分析输入文本并识别主要语言。预训练模型的置信度分数也是输出。

```
aws comprehend detect-dominant-language \
    --text "It is a beautiful day in Seattle."
```

输出：

```
{
    "Languages": [
        {
            "LanguageCode": "en",
            "Score": 0.9877256155014038
        }
    ]
}
```

有关更多信息，请参阅《Amazon Comprehend 开发人员指南》中的主要语言。

- 有关 API 的详细信息，请参阅AWS CLI 命令参考DetectDominantLanguage中的。

Java

适用于 Java 2.x 的 SDK

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.comprehend.ComprehendClient;
```

```
import software.amazon.awssdk.services.comprehend.model.ComprehendException;
import
 software.amazon.awssdk.services.comprehend.model.DetectDominantLanguageRequest;
import
 software.amazon.awssdk.services.comprehend.model.DetectDominantLanguageResponse;
import software.amazon.awssdk.services.comprehend.model.DominantLanguage;
import java.util.List;

/**
 * Before running this Java V2 code example, set up your development
 * environment, including your credentials.
 *
 * For more information, see the following documentation topic:
 *
 * https://docs.aws.amazon.com/sdk-for-java/latest/developer-guide/get-
started.html
 */
public class DetectLanguage {
    public static void main(String[] args) {
        // Specify French text - "It is raining today in Seattle".
        String text = "Il pleut aujourd'hui à Seattle";
        Region region = Region.US_EAST_1;

        ComprehendClient comClient = ComprehendClient.builder()
                .region(region)
                .build();

        System.out.println("Calling DetectDominantLanguage");
        detectTheDominantLanguage(comClient, text);
        comClient.close();
    }

    public static void detectTheDominantLanguage(ComprehendClient comClient,
 String text) {
        try {
            DetectDominantLanguageRequest request =
 DetectDominantLanguageRequest.builder()
                    .text(text)
                    .build();

            DetectDominantLanguageResponse resp =
 comClient.detectDominantLanguage(request);
            List<DominantLanguage> allLanList = resp.languages();
            for (DominantLanguage lang : allLanList) {
```

```
                System.out.println("Language is " + lang.languageCode());
            }

        } catch (ComprehendException e) {
            System.err.println(e.awsErrorDetails().errorMessage());
            System.exit(1);
        }
    }
}
```

- 有关 API 的详细信息，请参阅 AWS SDK for Java 2.x API 参考DetectDominantLanguage中的。

Python

SDK for Python (Boto3)

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
class ComprehendDetect:
    """Encapsulates Comprehend detection functions."""

    def __init__(self, comprehend_client):
        """
        :param comprehend_client: A Boto3 Comprehend client.
        """
        self.comprehend_client = comprehend_client


    def detect_languages(self, text):
        """
        Detects languages used in a document.

        :param text: The document to inspect.
        :return: The list of languages along with their confidence scores.
        """
```

```
        try:
            response = self.comprehend_client.detect_dominant_language(Text=text)
            languages = response["Languages"]
            logger.info("Detected %s languages.", len(languages))
        except ClientError:
            logger.exception("Couldn't detect languages.")
            raise
        else:
            return languages
```

- 有关 API 的详细信息，请参阅适用DetectDominantLanguage于 Python 的AWS SDK (Boto3) API 参考。

有关 S AWS DK 开发者指南和代码示例的完整列表，请参阅将 Amazon Comprehend 与 SDK 配合 AWS 使用。本主题还包括有关入门的信息以及有关先前的 SDK 版本的详细信息。

`DetectEntities`与 AWS SDK 或 CLI 配合使用

以下代码示例演示如何使用 `DetectEntities`。

操作示例是大型程序的代码摘录，必须在上下文中运行。在以下代码示例中，您可以查看此操作的上下文：

- 检测文档元素

.NET

AWS SDK for .NET

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
using System;
using System.Threading.Tasks;
```

```
    using Amazon.Comprehend;
    using Amazon.Comprehend.Model;

    /// <summary>
    /// This example shows how to use the AmazonComprehend service detect any
    /// entities in submitted text.
    /// </summary>
    public static class DetectEntities
    {
        /// <summary>
        /// The main method calls the DetectEntitiesAsync method to find any
        /// entities in the sample code.
        /// </summary>
        public static async Task Main()
        {
            string text = "It is raining today in Seattle";

            var comprehendClient = new AmazonComprehendClient();

            Console.WriteLine("Calling DetectEntities\n");
            var detectEntitiesRequest = new DetectEntitiesRequest()
            {
                Text = text,
                LanguageCode = "en",
            };
            var detectEntitiesResponse = await
comprehendClient.DetectEntitiesAsync(detectEntitiesRequest);

            foreach (var e in detectEntitiesResponse.Entities)
            {
                Console.WriteLine($"Text: {e.Text}, Type: {e.Type}, Score:
{e.Score}, BeginOffset: {e.BeginOffset}, EndOffset: {e.EndOffset}");
            }

            Console.WriteLine("Done");
        }
    }
```

- 有关 API 的详细信息，请参阅 AWS SDK for .NET API 参考DetectEntities中的。

CLI

AWS CLI

检测输入文本中的命名实体

以下 detect-entities 示例分析输入文本并返回命名实体。预训练模型的置信度分数也是每个预测的输出。

```
aws comprehend detect-entities \
    --language-code en \
    --text "Hello Zhang Wei, I am John. Your AnyCompany Financial Services, LLC
 credit card \
    account 1111-XXXX-1111-XXXX has a minimum payment of $24.53 that is due by
 July 31st. Based on your autopay settings, \
    we will withdraw your payment on the due date from your bank account number
 XXXXXX1111 with the routing number XXXXX0000. \
    Customer feedback for Sunshine Spa, 123 Main St, Anywhere. Send comments to
 Alice at AnySpa@example.com."
```

输出：

```
{
    "Entities": [
        {
            "Score": 0.9994556307792664,
            "Type": "PERSON",
            "Text": "Zhang Wei",
            "BeginOffset": 6,
            "EndOffset": 15
        },
        {
            "Score": 0.9981022477149963,
            "Type": "PERSON",
            "Text": "John",
            "BeginOffset": 22,
            "EndOffset": 26
        },
        {
            "Score": 0.9986887574195862,
            "Type": "ORGANIZATION",
            "Text": "AnyCompany Financial Services, LLC",
            "BeginOffset": 33,
```

```
            "EndOffset": 67
        },
        {
            "Score": 0.9959119558334351,
            "Type": "OTHER",
            "Text": "1111-XXXX-1111-XXXX",
            "BeginOffset": 88,
            "EndOffset": 107
        },
        {
            "Score": 0.9708039164543152,
            "Type": "QUANTITY",
            "Text": ".53",
            "BeginOffset": 133,
            "EndOffset": 136
        },
        {
            "Score": 0.9987268447875977,
            "Type": "DATE",
            "Text": "July 31st",
            "BeginOffset": 152,
            "EndOffset": 161
        },
        {
            "Score": 0.9858865737915039,
            "Type": "OTHER",
            "Text": "XXXXXX1111",
            "BeginOffset": 271,
            "EndOffset": 281
        },
        {
            "Score": 0.9700471758842468,
            "Type": "OTHER",
            "Text": "XXXXX0000",
            "BeginOffset": 306,
            "EndOffset": 315
        },
        {
            "Score": 0.9591118693351746,
            "Type": "ORGANIZATION",
            "Text": "Sunshine Spa",
            "BeginOffset": 340,
            "EndOffset": 352
        },
```

```
        {
            "Score": 0.9797496795654297,
            "Type": "LOCATION",
            "Text": "123 Main St",
            "BeginOffset": 354,
            "EndOffset": 365
        },
        {
            "Score": 0.994929313659668,
            "Type": "PERSON",
            "Text": "Alice",
            "BeginOffset": 394,
            "EndOffset": 399
        },
        {
            "Score": 0.9949769377708435,
            "Type": "OTHER",
            "Text": "AnySpa@example.com",
            "BeginOffset": 403,
            "EndOffset": 418
        }
    ]
}
```

有关更多信息，请参阅《Amazon Comprehend 开发人员指南》中的实体。

- 有关 API 的详细信息，请参阅AWS CLI 命令参考DetectEntities中的。

Java

适用于 Java 2.x 的 SDK

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.comprehend.ComprehendClient;
import software.amazon.awssdk.services.comprehend.model.DetectEntitiesRequest;
import software.amazon.awssdk.services.comprehend.model.DetectEntitiesResponse;
```

```
import software.amazon.awssdk.services.comprehend.model.Entity;
import software.amazon.awssdk.services.comprehend.model.ComprehendException;
import java.util.List;

/**
 * Before running this Java V2 code example, set up your development
 * environment, including your credentials.
 *
 * For more information, see the following documentation topic:
 *
 * https://docs.aws.amazon.com/sdk-for-java/latest/developer-guide/get-
started.html
 */
public class DetectEntities {
    public static void main(String[] args) {
        String text = "Amazon.com, Inc. is located in Seattle, WA and was founded
 July 5th, 1994 by Jeff Bezos, allowing customers to buy everything from books to
 blenders. Seattle is north of Portland and south of Vancouver, BC. Other notable
 Seattle - based companies are Starbucks and Boeing.";
        Region region = Region.US_EAST_1;
        ComprehendClient comClient = ComprehendClient.builder()
                .region(region)
                .build();

        System.out.println("Calling DetectEntities");
        detectAllEntities(comClient, text);
        comClient.close();
    }

    public static void detectAllEntities(ComprehendClient comClient, String text)
 {
        try {
            DetectEntitiesRequest detectEntitiesRequest =
 DetectEntitiesRequest.builder()
                    .text(text)
                    .languageCode("en")
                    .build();

            DetectEntitiesResponse detectEntitiesResult =
 comClient.detectEntities(detectEntitiesRequest);
            List<Entity> entList = detectEntitiesResult.entities();
            for (Entity entity : entList) {
                System.out.println("Entity text is " + entity.text());
            }
```

```
        } catch (ComprehendException e) {
            System.err.println(e.awsErrorDetails().errorMessage());
            System.exit(1);
        }
    }
}
```

- 有关 API 的详细信息，请参阅 AWS SDK for Java 2.x API 参考DetectEntities中的。

Python

SDK for Python (Boto3)

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
class ComprehendDetect:
    """Encapsulates Comprehend detection functions."""

    def __init__(self, comprehend_client):
        """
        :param comprehend_client: A Boto3 Comprehend client.
        """
        self.comprehend_client = comprehend_client


    def detect_entities(self, text, language_code):
        """
        Detects entities in a document. Entities can be things like people and
 places
        or other common terms.

        :param text: The document to inspect.
        :param language_code: The language of the document.
        :return: The list of entities along with their confidence scores.
        """
        try:
```

```
            response = self.comprehend_client.detect_entities(
                Text=text, LanguageCode=language_code
            )
            entities = response["Entities"]
            logger.info("Detected %s entities.", len(entities))
        except ClientError:
            logger.exception("Couldn't detect entities.")
            raise
        else:
            return entities
```

- 有关 API 的详细信息，请参阅适用DetectEntities于 Python 的AWS SDK (Boto3) API 参考。

有关 S AWS DK 开发者指南和代码示例的完整列表，请参阅将 Amazon Comprehend 与 SDK 配合 AWS 使用。本主题还包括有关入门的信息以及有关先前的 SDK 版本的详细信息。

DetectKeyPhrases与 AWS SDK 或 CLI 配合使用

以下代码示例演示如何使用 DetectKeyPhrases。

操作示例是大型程序的代码摘录，必须在上下文中运行。在以下代码示例中，您可以查看此操作的上下文：

- 检测文档元素

.NET

AWS SDK for .NET

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
using System;
using System.Threading.Tasks;
using Amazon.Comprehend;
```

```
    using Amazon.Comprehend.Model;

    /// <summary>
    /// This example shows how to use the Amazon Comprehend service to
    /// search text for key phrases.
    /// </summary>
    public static class DetectKeyPhrase
    {
        /// <summary>
        /// This method calls the Amazon Comprehend method DetectKeyPhrasesAsync
        /// to detect any key phrases in the sample text.
        /// </summary>
        public static async Task Main()
        {
            string text = "It is raining today in Seattle";

            var comprehendClient = new
AmazonComprehendClient(Amazon.RegionEndpoint.USWest2);

            // Call DetectKeyPhrases API
            Console.WriteLine("Calling DetectKeyPhrases");
            var detectKeyPhrasesRequest = new DetectKeyPhrasesRequest()
            {
                Text = text,
                LanguageCode = "en",
            };
            var detectKeyPhrasesResponse = await
comprehendClient.DetectKeyPhrasesAsync(detectKeyPhrasesRequest);
            foreach (var kp in detectKeyPhrasesResponse.KeyPhrases)
            {
                Console.WriteLine($"Text: {kp.Text}, Score: {kp.Score},
BeginOffset: {kp.BeginOffset}, EndOffset: {kp.EndOffset}");
            }

            Console.WriteLine("Done");
        }
    }
```

- 有关 API 的详细信息，请参阅 AWS SDK for .NET API 参考DetectKeyPhrases中的。

CLI

AWS CLI

检测输入文本中的关键词

以下 detect-key-phrases 示例分析输入文本并识别关键名词短语。预训练模型的置信度分数也是每个预测的输出。

```
aws comprehend detect-key-phrases \
    --language-code en \
    --text "Hello Zhang Wei, I am John. Your AnyCompany Financial Services, LLC
 credit card \
        account 1111-XXXX-1111-XXXX has a minimum payment of $24.53 that is due
 by July 31st. Based on your autopay settings, \
        we will withdraw your payment on the due date from your bank account
 number XXXXXX1111 with the routing number XXXXX0000. \
        Customer feedback for Sunshine Spa, 123 Main St, Anywhere. Send comments
 to Alice at AnySpa@example.com."
```

输出：

```
{
    "KeyPhrases": [
        {
            "Score": 0.8996376395225525,
            "Text": "Zhang Wei",
            "BeginOffset": 6,
            "EndOffset": 15
        },
        {
            "Score": 0.9992469549179077,
            "Text": "John",
            "BeginOffset": 22,
            "EndOffset": 26
        },
        {
            "Score": 0.988385021686554,
            "Text": "Your AnyCompany Financial Services",
            "BeginOffset": 28,
            "EndOffset": 62
        },
        {
```

```
            "Score": 0.8740853071212769,
            "Text": "LLC credit card account 1111-XXXX-1111-XXXX",
            "BeginOffset": 64,
            "EndOffset": 107
        },
        {
            "Score": 0.9999437928199768,
            "Text": "a minimum payment",
            "BeginOffset": 112,
            "EndOffset": 129
        },
        {
            "Score": 0.9998900890350342,
            "Text": ".53",
            "BeginOffset": 133,
            "EndOffset": 136
        },
        {
            "Score": 0.9979453086853027,
            "Text": "July 31st",
            "BeginOffset": 152,
            "EndOffset": 161
        },
        {
            "Score": 0.9983011484146118,
            "Text": "your autopay settings",
            "BeginOffset": 172,
            "EndOffset": 193
        },
        {
            "Score": 0.9996572136878967,
            "Text": "your payment",
            "BeginOffset": 211,
            "EndOffset": 223
        },
        {
            "Score": 0.9995037317276001,
            "Text": "the due date",
            "BeginOffset": 227,
            "EndOffset": 239
        },
        {
            "Score": 0.9702621698379517,
            "Text": "your bank account number XXXXXX1111",
```

```
            "BeginOffset": 245,
            "EndOffset": 280
        },
        {
            "Score": 0.9179925918579102,
            "Text": "the routing number XXXXX0000.Customer feedback",
            "BeginOffset": 286,
            "EndOffset": 332
        },
        {
            "Score": 0.9978160858154297,
            "Text": "Sunshine Spa",
            "BeginOffset": 337,
            "EndOffset": 349
        },
        {
            "Score": 0.9706913232803345,
            "Text": "123 Main St",
            "BeginOffset": 351,
            "EndOffset": 362
        },
        {
            "Score": 0.9941995143890381,
            "Text": "comments",
            "BeginOffset": 379,
            "EndOffset": 387
        },
        {
            "Score": 0.9759287238121033,
            "Text": "Alice",
            "BeginOffset": 391,
            "EndOffset": 396
        },
        {
            "Score": 0.8376792669296265,
            "Text": "AnySpa@example.com",
            "BeginOffset": 400,
            "EndOffset": 415
        }
    ]
}
```

有关更多信息，请参阅《Amazon Comprehend 开发人员指南》中的关键词。

- 有关 API 的详细信息，请参阅AWS CLI 命令参考DetectKeyPhrases中的。

Java

适用于 Java 2.x 的 SDK

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.comprehend.ComprehendClient;
import software.amazon.awssdk.services.comprehend.model.DetectKeyPhrasesRequest;
import software.amazon.awssdk.services.comprehend.model.DetectKeyPhrasesResponse;
import software.amazon.awssdk.services.comprehend.model.KeyPhrase;
import software.amazon.awssdk.services.comprehend.model.ComprehendException;
import java.util.List;

/**
 * Before running this Java V2 code example, set up your development
 * environment, including your credentials.
 *
 * For more information, see the following documentation topic:
 *
 * https://docs.aws.amazon.com/sdk-for-java/latest/developer-guide/get-
started.html
 */
public class DetectKeyPhrases {
    public static void main(String[] args) {
        String text = "Amazon.com, Inc. is located in Seattle, WA and was founded
 July 5th, 1994 by Jeff Bezos, allowing customers to buy everything from books to
 blenders. Seattle is north of Portland and south of Vancouver, BC. Other notable
 Seattle - based companies are Starbucks and Boeing.";
        Region region = Region.US_EAST_1;
        ComprehendClient comClient = ComprehendClient.builder()
                .region(region)
                .build();

        System.out.println("Calling DetectKeyPhrases");
        detectAllKeyPhrases(comClient, text);
```

```
        comClient.close();
    }

    public static void detectAllKeyPhrases(ComprehendClient comClient, String
 text) {
        try {
            DetectKeyPhrasesRequest detectKeyPhrasesRequest =
 DetectKeyPhrasesRequest.builder()
                    .text(text)
                    .languageCode("en")
                    .build();

            DetectKeyPhrasesResponse detectKeyPhrasesResult =
 comClient.detectKeyPhrases(detectKeyPhrasesRequest);
            List<KeyPhrase> phraseList = detectKeyPhrasesResult.keyPhrases();
            for (KeyPhrase keyPhrase : phraseList) {
                System.out.println("Key phrase text is " + keyPhrase.text());
            }

        } catch (ComprehendException e) {
            System.err.println(e.awsErrorDetails().errorMessage());
            System.exit(1);
        }
    }
}
```

- 有关 API 的详细信息，请参阅 AWS SDK for Java 2.x API 参考DetectKeyPhrases中的。

Python

SDK for Python (Boto3)

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
class ComprehendDetect:
    """Encapsulates Comprehend detection functions."""
```

```
    def __init__(self, comprehend_client):
        """
        :param comprehend_client: A Boto3 Comprehend client.
        """
        self.comprehend_client = comprehend_client


    def detect_key_phrases(self, text, language_code):
        """
        Detects key phrases in a document. A key phrase is typically a noun and
its
        modifiers.

        :param text: The document to inspect.
        :param language_code: The language of the document.
        :return: The list of key phrases along with their confidence scores.
        """
        try:
            response = self.comprehend_client.detect_key_phrases(
                Text=text, LanguageCode=language_code
            )
            phrases = response["KeyPhrases"]
            logger.info("Detected %s phrases.", len(phrases))
        except ClientError:
            logger.exception("Couldn't detect phrases.")
            raise
        else:
            return phrases
```

- 有关 API 的详细信息，请参阅适用DetectKeyPhrases于 Python 的AWS SDK (Boto3) API 参考。

有关 S AWS DK 开发者指南和代码示例的完整列表，请参阅将 Amazon Comprehend 与 SDK 配合 AWS 使用。本主题还包括有关入门的信息以及有关先前的 SDK 版本的详细信息。

DetectPiiEntities与 AWS SDK 或 CLI 配合使用

以下代码示例演示如何使用 DetectPiiEntities。

操作示例是大型程序的代码摘录，必须在上下文中运行。在以下代码示例中，您可以查看此操作的上下文：

- 检测文档元素

.NET

AWS SDK for .NET

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
using System;
using System.Threading.Tasks;
using Amazon.Comprehend;
using Amazon.Comprehend.Model;

/// <summary>
/// This example shows how to use the Amazon Comprehend service to find
/// personally identifiable information (PII) within text submitted to the
/// DetectPiiEntitiesAsync method.
/// </summary>
public class DetectingPII
{
    /// <summary>
    /// This method calls the DetectPiiEntitiesAsync method to locate any
    /// personally dientifiable information within the supplied text.
    /// </summary>
    public static async Task Main()
    {
        var comprehendClient = new AmazonComprehendClient();
        var text = @"Hello Paul Santos. The latest statement for your
                    credit card account 1111-0000-1111-0000 was
                    mailed to 123 Any Street, Seattle, WA 98109.";

        var request = new DetectPiiEntitiesRequest
        {
            Text = text,
```

```
                LanguageCode = "EN",
            };

            var response = await
comprehendClient.DetectPiiEntitiesAsync(request);

            if (response.Entities.Count > 0)
            {
                foreach (var entity in response.Entities)
                {
                    var entityValue = text.Substring(entity.BeginOffset,
entity.EndOffset - entity.BeginOffset);
                    Console.WriteLine($"{entity.Type}: {entityValue}");
                }
            }
        }
    }
```

- 有关 API 的详细信息，请参阅 AWS SDK for .NET API 参考DetectPiiEntities中的。

CLI

AWS CLI

检测输入文本中的 PII 实体

以下 detect-pii-entities 示例分析输入文本，并识别包含个人身份信息（PII）的实体。预训练模型的置信度分数也是每个预测的输出。

```
aws comprehend detect-pii-entities \
    --language-code en \
    --text "Hello Zhang Wei, I am John. Your AnyCompany Financial Services, LLC
 credit card \
        account 1111-XXXX-1111-XXXX has a minimum payment of $24.53 that is due
 by July 31st. Based on your autopay settings, \
        we will withdraw your payment on the due date from your bank account
 number XXXXXX1111 with the routing number XXXXX0000. \
        Customer feedback for Sunshine Spa, 123 Main St, Anywhere. Send comments
 to Alice at AnySpa@example.com."
```

输出：


```
{
    "Entities": [
        {
            "Score": 0.9998322129249573,
            "Type": "NAME",
            "BeginOffset": 6,
            "EndOffset": 15
        },
        {
            "Score": 0.9998878240585327,
            "Type": "NAME",
            "BeginOffset": 22,
            "EndOffset": 26
        },
        {
            "Score": 0.9994089603424072,
            "Type": "CREDIT_DEBIT_NUMBER",
            "BeginOffset": 88,
            "EndOffset": 107
        },
        {
            "Score": 0.9999760985374451,
            "Type": "DATE_TIME",
            "BeginOffset": 152,
            "EndOffset": 161
        },
        {
            "Score": 0.9999449253082275,
            "Type": "BANK_ACCOUNT_NUMBER",
            "BeginOffset": 271,
            "EndOffset": 281
        },
        {
            "Score": 0.9999847412109375,
            "Type": "BANK_ROUTING",
            "BeginOffset": 306,
            "EndOffset": 315
        },
        {
            "Score": 0.999925434589386,
            "Type": "ADDRESS",
            "BeginOffset": 354,
```

```
            "EndOffset": 365
        },
        {
            "Score": 0.9989161491394043,
            "Type": "NAME",
            "BeginOffset": 394,
            "EndOffset": 399
        },
        {
            "Score": 0.9994171857833862,
            "Type": "EMAIL",
            "BeginOffset": 403,
            "EndOffset": 418
        }
    ]
}
```

有关更多信息，请参阅《Amazon Comprehend 开发人员指南》中的个人身份信息（PII）。

- 有关 API 的详细信息，请参阅AWS CLI 命令参考DetectPiiEntities中的。

Python

SDK for Python (Boto3)

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
class ComprehendDetect:
    """Encapsulates Comprehend detection functions."""

    def __init__(self, comprehend_client):
        """
        :param comprehend_client: A Boto3 Comprehend client.
        """
        self.comprehend_client = comprehend_client


    def detect_pii(self, text, language_code):
```

```
        """
        Detects personally identifiable information (PII) in a document. PII can
be
        things like names, account numbers, or addresses.

        :param text: The document to inspect.
        :param language_code: The language of the document.
        :return: The list of PII entities along with their confidence scores.
        """
        try:
            response = self.comprehend_client.detect_pii_entities(
                Text=text, LanguageCode=language_code
            )
            entities = response["Entities"]
            logger.info("Detected %s PII entities.", len(entities))
        except ClientError:
            logger.exception("Couldn't detect PII entities.")
            raise
        else:
            return entities
```

- 有关 API 的详细信息，请参阅适用DetectPiiEntities于 Python 的AWS SDK (Boto3) API 参考。

有关 S AWS DK 开发者指南和代码示例的完整列表，请参阅将 Amazon Comprehend 与 SDK 配合 AWS 使用。本主题还包括有关入门的信息以及有关先前的 SDK 版本的详细信息。

`DetectSentiment`与 AWS SDK 或 CLI 配合使用

以下代码示例演示如何使用 `DetectSentiment`。

操作示例是大型程序的代码摘录，必须在上下文中运行。在以下代码示例中，您可以查看此操作的上下文：

- 检测文档元素

.NET

AWS SDK for .NET

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
    using System;
    using System.Threading.Tasks;
    using Amazon.Comprehend;
    using Amazon.Comprehend.Model;

    /// <summary>
    /// This example shows how to detect the overall sentiment of the supplied
    /// text using the Amazon Comprehend service.
    /// </summary>
    public static class DetectSentiment
    {
        /// <summary>
        /// This method calls the DetetectSentimentAsync method to analyze the
        /// supplied text and determine the overal sentiment.
        /// </summary>
        public static async Task Main()
        {
            string text = "It is raining today in Seattle";

            var comprehendClient = new
AmazonComprehendClient(Amazon.RegionEndpoint.USWest2);

            // Call DetectKeyPhrases API
            Console.WriteLine("Calling DetectSentiment");
            var detectSentimentRequest = new DetectSentimentRequest()
            {
                Text = text,
                LanguageCode = "en",
            };
            var detectSentimentResponse = await
comprehendClient.DetectSentimentAsync(detectSentimentRequest);
            Console.WriteLine($"Sentiment: {detectSentimentResponse.Sentiment}");
```

```
            Console.WriteLine("Done");
        }
    }
```

- 有关 API 的详细信息，请参阅 AWS SDK for .NET API 参考DetectSentiment中的。

CLI

AWS CLI

检测输入文本的情绪

以下 detect-sentiment 示例分析输入文本，并返回占主导地位的情绪（POSITIVE、NEUTRAL、MIXED 或NEGATIVE）的推断。

```
aws comprehend detect-sentiment \
    --language-code en \
    --text "It is a beautiful day in Seattle"
```

输出：

```
{
    "Sentiment": "POSITIVE",
    "SentimentScore": {
        "Positive": 0.9976957440376282,
        "Negative": 9.653854067437351e-05,
        "Neutral": 0.002169104292988777,
        "Mixed": 3.857641786453314e-05
    }
}
```

有关更多信息，请参阅《Amazon Comprehend 开发人员指南》中的情绪。

- 有关 API 的详细信息，请参阅AWS CLI 命令参考DetectSentiment中的。

Java

适用于 Java 2.x 的 SDK

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.comprehend.ComprehendClient;
import software.amazon.awssdk.services.comprehend.model.ComprehendException;
import software.amazon.awssdk.services.comprehend.model.DetectSentimentRequest;
import software.amazon.awssdk.services.comprehend.model.DetectSentimentResponse;

/**
 * Before running this Java V2 code example, set up your development
 * environment, including your credentials.
 *
 * For more information, see the following documentation topic:
 *
 * https://docs.aws.amazon.com/sdk-for-java/latest/developer-guide/get-
started.html
 */
public class DetectSentiment {
    public static void main(String[] args) {
        String text = "Amazon.com, Inc. is located in Seattle, WA and was founded
 July 5th, 1994 by Jeff Bezos, allowing customers to buy everything from books to
 blenders. Seattle is north of Portland and south of Vancouver, BC. Other notable
 Seattle - based companies are Starbucks and Boeing.";
        Region region = Region.US_EAST_1;
        ComprehendClient comClient = ComprehendClient.builder()
                .region(region)
                .build();

        System.out.println("Calling DetectSentiment");
        detectSentiments(comClient, text);
        comClient.close();
    }
```

```
    public static void detectSentiments(ComprehendClient comClient, String text)
 {
        try {
            DetectSentimentRequest detectSentimentRequest =
 DetectSentimentRequest.builder()
                    .text(text)
                    .languageCode("en")
                    .build();

            DetectSentimentResponse detectSentimentResult =
 comClient.detectSentiment(detectSentimentRequest);
            System.out.println("The Neutral value is " +
 detectSentimentResult.sentimentScore().neutral());

        } catch (ComprehendException e) {
            System.err.println(e.awsErrorDetails().errorMessage());
            System.exit(1);
        }
    }
}
```

- 有关 API 的详细信息，请参阅 AWS SDK for Java 2.x API 参考DetectSentiment中的。

Python

SDK for Python (Boto3)

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
class ComprehendDetect:
    """Encapsulates Comprehend detection functions."""

    def __init__(self, comprehend_client):
        """
        :param comprehend_client: A Boto3 Comprehend client.
        """
        self.comprehend_client = comprehend_client
```

```
    def detect_sentiment(self, text, language_code):
        """
        Detects the overall sentiment expressed in a document. Sentiment can
        be positive, negative, neutral, or a mixture.

        :param text: The document to inspect.
        :param language_code: The language of the document.
        :return: The sentiments along with their confidence scores.
        """
        try:
            response = self.comprehend_client.detect_sentiment(
                Text=text, LanguageCode=language_code
            )
            logger.info("Detected primary sentiment %s.", response["Sentiment"])
        except ClientError:
            logger.exception("Couldn't detect sentiment.")
            raise
        else:
            return response
```

- 有关 API 的详细信息，请参阅适用DetectSentiment于 Python 的AWS SDK (Boto3) API 参考。

有关 S AWS DK 开发者指南和代码示例的完整列表，请参阅将 Amazon Comprehend 与 SDK 配合 AWS 使用。本主题还包括有关入门的信息以及有关先前的 SDK 版本的详细信息。

DetectSyntax与 AWS SDK 或 CLI 配合使用

以下代码示例演示如何使用 DetectSyntax。

操作示例是大型程序的代码摘录，必须在上下文中运行。在以下代码示例中，您可以查看此操作的上下文：

- 检测文档元素

.NET

AWS SDK for .NET

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
    using System;
    using System.Threading.Tasks;
    using Amazon.Comprehend;
    using Amazon.Comprehend.Model;

    /// <summary>
    /// This example shows how to use Amazon Comprehend to detect syntax
    /// elements by calling the DetectSyntaxAsync method.
    /// </summary>
    public class DetectingSyntax
    {
        /// <summary>
        /// This method calls DetectSynaxAsync to identify the syntax elements
        /// in the sample text.
        /// </summary>
        public static async Task Main()
        {
            string text = "It is raining today in Seattle";

            var comprehendClient = new AmazonComprehendClient();

            // Call DetectSyntax API
            Console.WriteLine("Calling DetectSyntaxAsync\n");
            var detectSyntaxRequest = new DetectSyntaxRequest()
            {
                Text = text,
                LanguageCode = "en",
            };
            DetectSyntaxResponse detectSyntaxResponse = await
comprehendClient.DetectSyntaxAsync(detectSyntaxRequest);
            foreach (SyntaxToken s in detectSyntaxResponse.SyntaxTokens)
            {
```

```
                Console.WriteLine($"Text: {s.Text}, PartOfSpeech:
 {s.PartOfSpeech.Tag}, BeginOffset: {s.BeginOffset}, EndOffset: {s.EndOffset}");
            }

            Console.WriteLine("Done");
        }
    }
```

- 有关 API 的详细信息，请参阅 AWS SDK for .NET API 参考[DetectSyntax](#)中的。

CLI

AWS CLI

检测输入文本中的语音部分

以下 `detect-syntax` 示例分析输入文本的语法并返回语音的不同部分。预训练模型的置信度分数也是每个预测的输出。

```
aws comprehend detect-syntax \
    --language-code en \
    --text "It is a beautiful day in Seattle."
```

输出：

```
{
    "SyntaxTokens": [
        {
            "TokenId": 1,
            "Text": "It",
            "BeginOffset": 0,
            "EndOffset": 2,
            "PartOfSpeech": {
                "Tag": "PRON",
                "Score": 0.9999740719795227
            }
        },
        {
            "TokenId": 2,
            "Text": "is",
```

```
            "BeginOffset": 3,
            "EndOffset": 5,
            "PartOfSpeech": {
                "Tag": "VERB",
                "Score": 0.999901294708252
            }
        },
        {
            "TokenId": 3,
            "Text": "a",
            "BeginOffset": 6,
            "EndOffset": 7,
            "PartOfSpeech": {
                "Tag": "DET",
                "Score": 0.9999938607215881
            }
        },
        {
            "TokenId": 4,
            "Text": "beautiful",
            "BeginOffset": 8,
            "EndOffset": 17,
            "PartOfSpeech": {
                "Tag": "ADJ",
                "Score": 0.9987351894378662
            }
        },
        {
            "TokenId": 5,
            "Text": "day",
            "BeginOffset": 18,
            "EndOffset": 21,
            "PartOfSpeech": {
                "Tag": "NOUN",
                "Score": 0.9999796748161316
            }
        },
        {
            "TokenId": 6,
            "Text": "in",
            "BeginOffset": 22,
            "EndOffset": 24,
            "PartOfSpeech": {
                "Tag": "ADP",
```

```
                "Score": 0.9998047947883606
            }
        },
        {
            "TokenId": 7,
            "Text": "Seattle",
            "BeginOffset": 25,
            "EndOffset": 32,
            "PartOfSpeech": {
                "Tag": "PROPN",
                "Score": 0.9940530061721802
            }
        }
    ]
}
```

有关更多信息，请参阅《Amazon Comprehend 开发人员指南》中的语法分析。

- 有关 API 的详细信息，请参阅AWS CLI 命令参考DetectSyntax中的。

Java

适用于 Java 2.x 的 SDK

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.comprehend.ComprehendClient;
import software.amazon.awssdk.services.comprehend.model.ComprehendException;
import software.amazon.awssdk.services.comprehend.model.DetectSyntaxRequest;
import software.amazon.awssdk.services.comprehend.model.DetectSyntaxResponse;
import software.amazon.awssdk.services.comprehend.model.SyntaxToken;
import java.util.List;

/**
 * Before running this Java V2 code example, set up your development
 * environment, including your credentials.
 *
```

```
 * For more information, see the following documentation topic:
 *
 * https://docs.aws.amazon.com/sdk-for-java/latest/developer-guide/get-
started.html
 */
public class DetectSyntax {
    public static void main(String[] args) {
        String text = "Amazon.com, Inc. is located in Seattle, WA and was founded
 July 5th, 1994 by Jeff Bezos, allowing customers to buy everything from books to
 blenders. Seattle is north of Portland and south of Vancouver, BC. Other notable
 Seattle - based companies are Starbucks and Boeing.";
        Region region = Region.US_EAST_1;
        ComprehendClient comClient = ComprehendClient.builder()
                .region(region)
                .build();

        System.out.println("Calling DetectSyntax");
        detectAllSyntax(comClient, text);
        comClient.close();
    }

    public static void detectAllSyntax(ComprehendClient comClient, String text) {
        try {
            DetectSyntaxRequest detectSyntaxRequest =
 DetectSyntaxRequest.builder()
                    .text(text)
                    .languageCode("en")
                    .build();

            DetectSyntaxResponse detectSyntaxResult =
 comClient.detectSyntax(detectSyntaxRequest);
            List<SyntaxToken> syntaxTokens = detectSyntaxResult.syntaxTokens();
            for (SyntaxToken token : syntaxTokens) {
                System.out.println("Language is " + token.text());
                System.out.println("Part of speech is " +
 token.partOfSpeech().tagAsString());
            }

        } catch (ComprehendException e) {
            System.err.println(e.awsErrorDetails().errorMessage());
            System.exit(1);
        }
    }
}
```

- 有关 API 的详细信息，请参阅 AWS SDK for Java 2.x API 参考DetectSyntax中的。

Python

SDK for Python (Boto3)

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
class ComprehendDetect:
    """Encapsulates Comprehend detection functions."""

    def __init__(self, comprehend_client):
        """
        :param comprehend_client: A Boto3 Comprehend client.
        """
        self.comprehend_client = comprehend_client


    def detect_syntax(self, text, language_code):
        """
        Detects syntactical elements of a document. Syntax tokens are portions of
        text along with their use as parts of speech, such as nouns, verbs, and
        interjections.

        :param text: The document to inspect.
        :param language_code: The language of the document.
        :return: The list of syntax tokens along with their confidence scores.
        """
        try:
            response = self.comprehend_client.detect_syntax(
                Text=text, LanguageCode=language_code
            )
            tokens = response["SyntaxTokens"]
            logger.info("Detected %s syntax tokens.", len(tokens))
        except ClientError:
            logger.exception("Couldn't detect syntax.")
```

```
            raise
        else:
            return tokens
```

- 有关 API 的详细信息，请参阅适用DetectSyntax于 Python 的AWS SDK (Boto3) API 参考。

有关 S AWS DK 开发者指南和代码示例的完整列表，请参阅将 Amazon Comprehend 与 SDK 配合 AWS 使用。本主题还包括有关入门的信息以及有关先前的 SDK 版本的详细信息。

`ListDocumentClassificationJobs`与 AWS SDK 或 CLI 配合使用

以下代码示例演示如何使用 `ListDocumentClassificationJobs`。

操作示例是大型程序的代码摘录，必须在上下文中运行。在以下代码示例中，您可以查看此操作的上下文：

- 训练自定义分类器并对文档进行分类

CLI

AWS CLI

列出所有文档分类作业

以下 `list-document-classification-jobs` 示例列出所有文档分类作业。

```
aws comprehend list-document-classification-jobs
```

输出：

```
{
    "DocumentClassificationJobPropertiesList": [
        {
            "JobId": "123456abcdeb0e11022f22a11EXAMPLE",
            "JobArn": "arn:aws:comprehend:us-west-2:1234567890101:document-
classification-job/123456abcdeb0e11022f22a11EXAMPLE",
            "JobName": "exampleclassificationjob",
            "JobStatus": "COMPLETED",
```

```
            "SubmitTime": "2023-06-14T17:09:51.788000+00:00",
            "EndTime": "2023-06-14T17:15:58.582000+00:00",
            "DocumentClassifierArn": "arn:aws:comprehend:us-
west-2:1234567890101:document-classifier/mymodel/version/12",
            "InputDataConfig": {
                "S3Uri": "s3://DOC-EXAMPLE-BUCKET/jobdata/",
                "InputFormat": "ONE_DOC_PER_LINE"
            },
            "OutputDataConfig": {
                "S3Uri": "s3://DOC-EXAMPLE-DESTINATION-BUCKET/
thefolder/1234567890101-CLN-e758dd56b824aa717ceab551f11749fb/output/
output.tar.gz"
            },
            "DataAccessRoleArn": "arn:aws:iam::1234567890101:role/service-role/
AmazonComprehendServiceRole-example-role"
        },
        {
            "JobId": "123456abcdeb0e11022f22a1EXAMPLE2",
            "JobArn": "arn:aws:comprehend:us-west-2:1234567890101:document-
classification-job/123456abcdeb0e11022f22a1EXAMPLE2",
            "JobName": "exampleclassificationjob2",
            "JobStatus": "COMPLETED",
            "SubmitTime": "2023-06-14T17:22:39.829000+00:00",
            "EndTime": "2023-06-14T17:28:46.107000+00:00",
            "DocumentClassifierArn": "arn:aws:comprehend:us-
west-2:1234567890101:document-classifier/mymodel/version/12",
            "InputDataConfig": {
                "S3Uri": "s3://DOC-EXAMPLE-BUCKET/jobdata/",
                "InputFormat": "ONE_DOC_PER_LINE"
            },
            "OutputDataConfig": {
                "S3Uri": "s3://DOC-EXAMPLE-DESTINATION-BUCKET/
thefolder/1234567890101-CLN-123456abcdeb0e11022f22a1EXAMPLE2/output/
output.tar.gz"
            },
            "DataAccessRoleArn": "arn:aws:iam::1234567890101:role/service-role/
AmazonComprehendServiceRole-example-role"
        }
    ]
}
```

有关更多信息，请参阅《Amazon Comprehend 开发人员指南》中的自定义分类。

- 有关 API 的详细信息，请参阅AWS CLI 命令参考ListDocumentClassificationJobs中的。

Python

SDK for Python (Boto3)

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
class ComprehendClassifier:
    """Encapsulates an Amazon Comprehend custom classifier."""

    def __init__(self, comprehend_client):
        """
        :param comprehend_client: A Boto3 Comprehend client.
        """
        self.comprehend_client = comprehend_client
        self.classifier_arn = None


    def list_jobs(self):
        """
        Lists the classification jobs for the current account.

        :return: The list of jobs.
        """
        try:
            response = self.comprehend_client.list_document_classification_jobs()
            jobs = response["DocumentClassificationJobPropertiesList"]
            logger.info("Got %s document classification jobs.", len(jobs))
        except ClientError:
            logger.exception(
                "Couldn't get document classification jobs.",
            )
            raise
        else:
            return jobs
```

- 有关 API 的详细信息，请参阅适用ListDocumentClassificationJobs于 Python 的AWS SDK (Boto3) API 参考。

有关 S AWS DK 开发者指南和代码示例的完整列表，请参阅将 Amazon Comprehend 与 SDK 配合 AWS 使用。本主题还包括有关入门的信息以及有关先前的 SDK 版本的详细信息。

`ListDocumentClassifiers`与 AWS SDK 或 CLI 配合使用

以下代码示例演示如何使用 `ListDocumentClassifiers`。

操作示例是大型程序的代码摘录，必须在上下文中运行。在以下代码示例中，您可以查看此操作的上下文：

- 训练自定义分类器并对文档进行分类

CLI

AWS CLI

列出所有文档分类器

以下 `list-document-classifiers` 示例列出所有经过训练和正在训练的文档分类器模型。

```
aws comprehend list-document-classifiers
```

输出：

```
{
    "DocumentClassifierPropertiesList": [
        {
            "DocumentClassifierArn": "arn:aws:comprehend:us-
west-2:111122223333:document-classifier/exampleclassifier1",
            "LanguageCode": "en",
            "Status": "TRAINED",
            "SubmitTime": "2023-06-13T19:04:15.735000+00:00",
            "EndTime": "2023-06-13T19:42:31.752000+00:00",
            "TrainingStartTime": "2023-06-13T19:08:20.114000+00:00",
            "TrainingEndTime": "2023-06-13T19:41:35.080000+00:00",
            "InputDataConfig": {
                "DataFormat": "COMPREHEND_CSV",
```

```
                "S3Uri": "s3://DOC-EXAMPLE-BUCKET/trainingdata"
            },
            "OutputDataConfig": {},
            "ClassifierMetadata": {
                "NumberOfLabels": 3,
                "NumberOfTrainedDocuments": 5016,
                "NumberOfTestDocuments": 557,
                "EvaluationMetrics": {
                    "Accuracy": 0.9856,
                    "Precision": 0.9919,
                    "Recall": 0.9459,
                    "F1Score": 0.9673,
                    "MicroPrecision": 0.9856,
                    "MicroRecall": 0.9856,
                    "MicroF1Score": 0.9856,
                    "HammingLoss": 0.0144
                }
            },
            "DataAccessRoleArn": "arn:aws:iam::111122223333:role/service-role/
AmazonComprehendServiceRole-testorle",
            "Mode": "MULTI_CLASS"
        },
        {
            "DocumentClassifierArn": "arn:aws:comprehend:us-
west-2:111122223333:document-classifier/exampleclassifier2",
            "LanguageCode": "en",
            "Status": "TRAINING",
            "SubmitTime": "2023-06-13T21:20:28.690000+00:00",
            "InputDataConfig": {
                "DataFormat": "COMPREHEND_CSV",
                "S3Uri": "s3://DOC-EXAMPLE-BUCKET/trainingdata"
            },
            "OutputDataConfig": {},
            "DataAccessRoleArn": "arn:aws:iam::111122223333:role/service-role/
AmazonComprehendServiceRole-testorle",
            "Mode": "MULTI_CLASS"
        }
    ]
}
```

有关更多信息，请参阅《Amazon Comprehend 开发人员指南》中的创建和管理自定义模型。

- 有关 API 的详细信息，请参阅AWS CLI 命令参考ListDocumentClassifiers中的。

Python

SDK for Python (Boto3)

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
class ComprehendClassifier:
    """Encapsulates an Amazon Comprehend custom classifier."""

    def __init__(self, comprehend_client):
        """
        :param comprehend_client: A Boto3 Comprehend client.
        """
        self.comprehend_client = comprehend_client
        self.classifier_arn = None


    def list(self):
        """
        Lists custom classifiers for the current account.

        :return: The list of classifiers.
        """
        try:
            response = self.comprehend_client.list_document_classifiers()
            classifiers = response["DocumentClassifierPropertiesList"]
            logger.info("Got %s classifiers.", len(classifiers))
        except ClientError:
            logger.exception(
                "Couldn't get classifiers.",
            )
            raise
        else:
            return classifiers
```

- 有关 API 的详细信息，请参阅适用ListDocumentClassifiers于 Python 的AWS SDK (Boto3) API 参考。

有关 S AWS DK 开发者指南和代码示例的完整列表，请参阅将 Amazon Comprehend 与 SDK 配合 AWS 使用。本主题还包括有关入门的信息以及有关先前的 SDK 版本的详细信息。

`ListTopicsDetectionJobs`与 AWS SDK 或 CLI 配合使用

以下代码示例演示如何使用 `ListTopicsDetectionJobs`。

操作示例是大型程序的代码摘录，必须在上下文中运行。在以下代码示例中，您可以查看此操作的上下文：

- 对示例数据运行主题建模任务

CLI

AWS CLI

列出所有主题检测作业

以下 `list-topics-detection-jobs` 示例列出所有正在进行和已完成的异步主题检测作业。

```
aws comprehend list-topics-detection-jobs
```

输出：

```
{
    "TopicsDetectionJobPropertiesList": [
        {
            "JobId": "123456abcdeb0e11022f22a11EXAMPLE",
            "JobArn": "arn:aws:comprehend:us-west-2:111122223333:topics-
detection-job/123456abcdeb0e11022f22a11EXAMPLE",
            "JobName" "topic-analysis-1"
            "JobStatus": "IN_PROGRESS",
            "SubmitTime": "2023-06-09T18:40:35.384000+00:00",
            "EndTime": "2023-06-09T18:46:41.936000+00:00",
            "InputDataConfig": {
                "S3Uri": "s3://DOC-EXAMPLE-BUCKET",
                "InputFormat": "ONE_DOC_PER_LINE"
```

```
            },
            "OutputDataConfig": {
                "S3Uri": "s3://DOC-EXAMPLE-DESTINATION-BUCKET/
thefolder/111122223333-TOPICS-123456abcdeb0e11022f22a11EXAMPLE/output/
output.tar.gz"
            },
            "NumberOfTopics": 10,
            "DataAccessRoleArn": "arn:aws:iam::111122223333:role/service-role/
AmazonComprehendServiceRole-example-role"
        },
        {
            "JobId": "123456abcdeb0e11022f22a1EXAMPLE2",
            "JobArn": "arn:aws:comprehend:us-west-2:111122223333:topics-
detection-job/123456abcdeb0e11022f22a1EXAMPLE2",
            "JobName": "topic-analysis-2",
            "JobStatus": "COMPLETED",
            "SubmitTime": "2023-06-09T18:44:43.414000+00:00",
            "EndTime": "2023-06-09T18:50:50.872000+00:00",
            "InputDataConfig": {
                "S3Uri": "s3://DOC-EXAMPLE-BUCKET",
                "InputFormat": "ONE_DOC_PER_LINE"
            },
            "OutputDataConfig": {
                "S3Uri": "s3://DOC-EXAMPLE-DESTINATION-BUCKET/
thefolder/111122223333-TOPICS-123456abcdeb0e11022f22a1EXAMPLE2/output/
output.tar.gz"
            },
            "NumberOfTopics": 10,
            "DataAccessRoleArn": "arn:aws:iam::111122223333:role/service-role/
AmazonComprehendServiceRole-example-role"
        },
        {
            "JobId": "123456abcdeb0e11022f22a1EXAMPLE3",
            "JobArn": "arn:aws:comprehend:us-west-2:111122223333:topics-
detection-job/123456abcdeb0e11022f22a1EXAMPLE3",
            "JobName": "topic-analysis-2",
            "JobStatus": "IN_PROGRESS",
            "SubmitTime": "2023-06-09T18:50:56.737000+00:00",
            "InputDataConfig": {
                "S3Uri": "s3://DOC-EXAMPLE-BUCKET",
                "InputFormat": "ONE_DOC_PER_LINE"
            },
            "OutputDataConfig": {
```

```
                "S3Uri": "s3://DOC-EXAMPLE-DESTINATION-BUCKET/
thefolder/111122223333-TOPICS-123456abcdeb0e11022f22a1EXAMPLE3/output/
output.tar.gz"
            },
            "NumberOfTopics": 10,
            "DataAccessRoleArn": "arn:aws:iam::111122223333:role/service-role/
AmazonComprehendServiceRole-example-role"
        }
    ]
}
```

有关更多信息，请参阅《Amazon Comprehend 开发人员指南》中的 Amazon Comprehend 洞察的异步分析。

- 有关 API 的详细信息，请参阅AWS CLI 命令参考ListTopicsDetectionJobs中的。

Python

SDK for Python (Boto3)

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
class ComprehendTopicModeler:
    """Encapsulates a Comprehend topic modeler."""

    def __init__(self, comprehend_client):
        """
        :param comprehend_client: A Boto3 Comprehend client.
        """
        self.comprehend_client = comprehend_client


    def list_jobs(self):
        """
        Lists topic modeling jobs for the current account.

        :return: The list of jobs.
        """
```

```
        try:
            response = self.comprehend_client.list_topics_detection_jobs()
            jobs = response["TopicsDetectionJobPropertiesList"]
            logger.info("Got %s topic detection jobs.", len(jobs))
        except ClientError:
            logger.exception("Couldn't get topic detection jobs.")
            raise
        else:
            return jobs
```

- 有关 API 的详细信息，请参阅适用ListTopicsDetectionJobs于 Python 的AWS SDK (Boto3) API 参考。

有关 S AWS DK 开发者指南和代码示例的完整列表，请参阅将 Amazon Comprehend 与 SDK 配合 AWS 使用。本主题还包括有关入门的信息以及有关先前的 SDK 版本的详细信息。

`StartDocumentClassificationJob`与 AWS SDK 或 CLI 配合使用

以下代码示例演示如何使用 `StartDocumentClassificationJob`。

操作示例是大型程序的代码摘录，必须在上下文中运行。在以下代码示例中，您可以查看此操作的上下文：

- 训练自定义分类器并对文档进行分类

CLI

AWS CLI

列出文档分类作业

以下 `start-document-classification-job` 示例以自定义模型启动文档分类作业，该作业对 `--input-data-config` 标签所指定地址处的所有文件都使用自定义模型。在此示例中，输入 S3 存储桶包含 `SampleSMStext1.txt`、`SampleSMStext2.txt`、和 `SampleSMStext3.txt`。该模型之前曾接受过关于垃圾邮件和非垃圾邮件，或“ham”、短信的文档分类训练。作业完成后，`output.tar.gz` 将放置在 `--output-data-config` 标签指定

的位置。output.tar.gz 包含 predictions.jsonl，其中列出了每个文档的分类。Json 输出在每个文件的一行上打印，但是为了便于阅读，此处设置了格式。

```
aws comprehend start-document-classification-job \
    --job-name exampleclassificationjob \
    --input-data-config "S3Uri=s3://DOC-EXAMPLE-BUCKET-INPUT/jobdata/" \
    --output-data-config "S3Uri=s3://DOC-EXAMPLE-DESTINATION-BUCKET/testfolder/"
 \
    --data-access-role-arn arn:aws:iam::111122223333:role/service-role/
AmazonComprehendServiceRole-example-role \
    --document-classifier-arn arn:aws:comprehend:us-west-2:111122223333:document-
classifier/mymodel/version/12
```

SampleSMStext1.txt 的内容：

```
"CONGRATULATIONS! TXT 2155550100 to win $5000"
```

SampleSMStext2.txt 的内容：

```
"Hi, when do you want me to pick you up from practice?"
```

SampleSMStext3.txt 的内容：

```
"Plz send bank account # to 2155550100 to claim prize!!"
```

输出：

```
{
    "JobId": "e758dd56b824aa717ceab551fEXAMPLE",
    "JobArn": "arn:aws:comprehend:us-west-2:111122223333:document-classification-
job/e758dd56b824aa717ceab551fEXAMPLE",
    "JobStatus": "SUBMITTED"
}
```

predictions.jsonl 的内容：

```
{"File": "SampleSMSText1.txt", "Line": "0", "Classes": [{"Name": "spam", "Score":
 0.9999}, {"Name": "ham", "Score": 0.0001}]}
{"File": "SampleSMStext2.txt", "Line": "0", "Classes": [{"Name": "ham", "Score":
 0.9994}, {"Name": "spam", "Score": 0.0006}]}
```

```
{"File": "SampleSMSText3.txt", "Line": "0", "Classes": [{"Name": "spam", "Score":
 0.9999}, {"Name": "ham", "Score": 0.0001}]}
```

有关更多信息，请参阅《Amazon Comprehend 开发人员指南》中的自定义分类。

- 有关 API 的详细信息，请参阅AWS CLI 命令参考StartDocumentClassificationJob中的。

Python

SDK for Python (Boto3)

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
class ComprehendClassifier:
    """Encapsulates an Amazon Comprehend custom classifier."""

    def __init__(self, comprehend_client):
        """
        :param comprehend_client: A Boto3 Comprehend client.
        """
        self.comprehend_client = comprehend_client
        self.classifier_arn = None


    def start_job(
        self,
        job_name,
        input_bucket,
        input_key,
        input_format,
        output_bucket,
        output_key,
        data_access_role_arn,
    ):
        """
        Starts a classification job. The classifier must be trained or the job
        will fail. Input is read from the specified Amazon S3 input bucket and
        written to the specified output bucket. Output data is stored in a tar
```

```
        archive compressed in gzip format. The job runs asynchronously, so you
can
        call `describe_document_classification_job` to get job status until it
        returns a status of SUCCEEDED.

        :param job_name: The name of the job.
        :param input_bucket: The Amazon S3 bucket that contains input data.
        :param input_key: The prefix used to find input data in the input
                          bucket. If multiple objects have the same prefix, all
                          of them are used.
        :param input_format: The format of the input data, either one document
per
                             file or one document per line.
        :param output_bucket: The Amazon S3 bucket where output data is written.
        :param output_key: The prefix prepended to the output data.
        :param data_access_role_arn: The Amazon Resource Name (ARN) of a role
that
                                     grants Comprehend permission to read from
the
                                     input bucket and write to the output bucket.
        :return: Information about the job, including the job ID.
        """
        try:
            response = self.comprehend_client.start_document_classification_job(
                DocumentClassifierArn=self.classifier_arn,
                JobName=job_name,
                InputDataConfig={
                    "S3Uri": f"s3://{input_bucket}/{input_key}",
                    "InputFormat": input_format.value,
                },
                OutputDataConfig={"S3Uri": f"s3://{output_bucket}/{output_key}"},
                DataAccessRoleArn=data_access_role_arn,
            )
            logger.info(
                "Document classification job %s is %s.", job_name,
response["JobStatus"]
            )
        except ClientError:
            logger.exception("Couldn't start classification job %s.", job_name)
            raise
        else:
            return response
```

- 有关 API 的详细信息，请参阅适用StartDocumentClassificationJob于 Python 的AWS SDK (Boto3) API 参考。

有关 S AWS DK 开发者指南和代码示例的完整列表，请参阅将 Amazon Comprehend 与 SDK 配合 AWS 使用。本主题还包括有关入门的信息以及有关先前的 SDK 版本的详细信息。

StartTopicsDetectionJob与 AWS SDK 或 CLI 配合使用

以下代码示例演示如何使用 StartTopicsDetectionJob。

操作示例是大型程序的代码摘录，必须在上下文中运行。在以下代码示例中，您可以查看此操作的上下文：

- 对示例数据运行主题建模任务

.NET

AWS SDK for .NET

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
using System;
using System.Threading.Tasks;
using Amazon.Comprehend;
using Amazon.Comprehend.Model;

/// <summary>
/// This example scans the documents in an Amazon Simple Storage Service
/// (Amazon S3) bucket and analyzes it for topics. The results are stored
/// in another bucket and then the resulting job properties are displayed
/// on the screen. This example was created using the AWS SDK for .NEt
/// version 3.7 and .NET Core version 5.0.
/// </summary>
public static class TopicModeling
```

```
    {
        /// <summary>
        /// This methos calls a topic detection job by calling the Amazon
        /// Comprehend StartTopicsDetectionJobRequest.
        /// </summary>
        public static async Task Main()
        {
            var comprehendClient = new AmazonComprehendClient();

            string inputS3Uri = "s3://input bucket/input path";
            InputFormat inputDocFormat = InputFormat.ONE_DOC_PER_FILE;
            string outputS3Uri = "s3://output bucket/output path";
            string dataAccessRoleArn = "arn:aws:iam::account ID:role/data access
role";
            int numberOfTopics = 10;

            var startTopicsDetectionJobRequest = new
StartTopicsDetectionJobRequest()
            {
                InputDataConfig = new InputDataConfig()
                {
                    S3Uri = inputS3Uri,
                    InputFormat = inputDocFormat,
                },
                OutputDataConfig = new OutputDataConfig()
                {
                    S3Uri = outputS3Uri,
                },
                DataAccessRoleArn = dataAccessRoleArn,
                NumberOfTopics = numberOfTopics,
            };

            var startTopicsDetectionJobResponse = await
comprehendClient.StartTopicsDetectionJobAsync(startTopicsDetectionJobRequest);

            var jobId = startTopicsDetectionJobResponse.JobId;
            Console.WriteLine("JobId: " + jobId);

            var describeTopicsDetectionJobRequest = new
DescribeTopicsDetectionJobRequest()
            {
                JobId = jobId,
            };
```

```
            var describeTopicsDetectionJobResponse = await
comprehendClient.DescribeTopicsDetectionJobAsync(describeTopicsDetectionJobRequest);

PrintJobProperties(describeTopicsDetectionJobResponse.TopicsDetectionJobProperties);

            var listTopicsDetectionJobsResponse = await
comprehendClient.ListTopicsDetectionJobsAsync(new
ListTopicsDetectionJobsRequest());
            foreach (var props in
listTopicsDetectionJobsResponse.TopicsDetectionJobPropertiesList)
            {
                PrintJobProperties(props);
            }
        }

        /// <summary>
        /// This method is a helper method that displays the job properties
        /// from the call to StartTopicsDetectionJobRequest.
        /// </summary>
        /// <param name="props">A list of properties from the call to
        /// StartTopicsDetectionJobRequest.</param>
        private static void PrintJobProperties(TopicsDetectionJobProperties
props)
        {
            Console.WriteLine($"JobId: {props.JobId}, JobName: {props.JobName},
JobStatus: {props.JobStatus}");
            Console.WriteLine($"NumberOfTopics:
{props.NumberOfTopics}\nInputS3Uri: {props.InputDataConfig.S3Uri}");
            Console.WriteLine($"InputFormat: {props.InputDataConfig.InputFormat},
OutputS3Uri: {props.OutputDataConfig.S3Uri}");
        }
    }
```

- 有关 API 的详细信息，请参阅 AWS SDK for .NET API 参考StartTopicsDetectionJob中的。

CLI

AWS CLI

启动主题检测分析作业

以下 start-topics-detection-job 示例为位于 --input-data-config 标签指定地址的所有文件启动异步主题检测作业。作业完成后，文件夹 output 将放置在 --ouput-data-config 标签指定的位置。output 包含 topic-terms.csv 和 doc-topics.csv。第一个输出文件 topic-terms.csv 是集合中的主题列表。对于每个主题，默认情况下，该列表按权重排列主题列出根据其的热门术语。第二个文件 doc-topics.csv 列出了与主题相关的文档以及与该主题相关的文档比例。

```
aws comprehend start-topics-detection-job \
    --job-name example_topics_detection_job \
    --language-code en \
    --input-data-config "S3Uri=s3://DOC-EXAMPLE-BUCKET/" \
    --output-data-config "S3Uri=s3://DOC-EXAMPLE-DESTINATION-BUCKET/testfolder/"
 \
    --data-access-role-arn arn:aws:iam::111122223333:role/service-role/
AmazonComprehendServiceRole-example-role \
    --language-code en
```

输出：

```
{
    "JobId": "123456abcdeb0e11022f22a11EXAMPLE",
    "JobArn": "arn:aws:comprehend:us-west-2:111122223333:key-phrases-detection-
job/123456abcdeb0e11022f22a11EXAMPLE",
    "JobStatus": "SUBMITTED"
}
```

有关更多信息，请参阅《Amazon Comprehend 开发人员指南》中的主题建模。

- 有关 API 的详细信息，请参阅AWS CLI 命令参考StartTopicsDetectionJob中的。

Python

SDK for Python (Boto3)

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
class ComprehendTopicModeler:
    """Encapsulates a Comprehend topic modeler."""

    def __init__(self, comprehend_client):
        """
        :param comprehend_client: A Boto3 Comprehend client.
        """
        self.comprehend_client = comprehend_client


    def start_job(
        self,
        job_name,
        input_bucket,
        input_key,
        input_format,
        output_bucket,
        output_key,
        data_access_role_arn,
    ):
        """
        Starts a topic modeling job. Input is read from the specified Amazon S3
        input bucket and written to the specified output bucket. Output data is
 stored
        in a tar archive compressed in gzip format. The job runs asynchronously,
 so you
        can call `describe_topics_detection_job` to get job status until it
        returns a status of SUCCEEDED.

        :param job_name: The name of the job.
        :param input_bucket: An Amazon S3 bucket that contains job input.
        :param input_key: The prefix used to find input data in the input
                          bucket. If multiple objects have the same prefix,
 all
                          of them are used.
        :param input_format: The format of the input data, either one document
 per
                          file or one document per line.
        :param output_bucket: The Amazon S3 bucket where output data is written.
        :param output_key: The prefix prepended to the output data.
        :param data_access_role_arn: The Amazon Resource Name (ARN) of a role
 that
```

```
                                    grants Comprehend permission to read from
the
                                    input bucket and write to the output bucket.
        :return: Information about the job, including the job ID.
        """
        try:
            response = self.comprehend_client.start_topics_detection_job(
                JobName=job_name,
                DataAccessRoleArn=data_access_role_arn,
                InputDataConfig={
                    "S3Uri": f"s3://{input_bucket}/{input_key}",
                    "InputFormat": input_format.value,
                },
                OutputDataConfig={"S3Uri": f"s3://{output_bucket}/{output_key}"},
            )
            logger.info("Started topic modeling job %s.", response["JobId"])
        except ClientError:
            logger.exception("Couldn't start topic modeling job.")
            raise
        else:
            return response
```

- 有关 API 的详细信息，请参阅适用StartTopicsDetectionJob于 Python 的AWS SDK (Boto3) API 参考。

有关 S AWS DK 开发者指南和代码示例的完整列表，请参阅将 Amazon Comprehend 与 SDK 配合 AWS 使用。本主题还包括有关入门的信息以及有关先前的 SDK 版本的详细信息。

使用软件开发工具包的 Amazon Comprehend 场景 AWS

以下代码示例向您展示了如何使用软件开发工具包在 Amazon Comprehend 中实现常见场景。 AWS 这些场景向您展示了如何通过在 Amazon Comprehend 中调用多个函数来完成特定任务。每个场景都包含一个指向的链接 GitHub，您可以在其中找到有关如何设置和运行代码的说明。

示例

- 使用 Amazon Comprehend 和软件开发工具包检测文档元素 AWS
- 使用软件开发工具包对示例数据运行 Amazon Comprehend 主题建模作业 AWS
- 训练自定义 Amazon Comprehend 分类器并使用软件开发工具包对文档进行分类 AWS

使用 Amazon Comprehend 和软件开发工具包检测文档元素 AWS

以下代码示例展示了如何：

- 检测文档中的语言、实体和关键短语。
- 检测文档中的个人身份信息 (PII)。
- 检测文档的情绪。
- 检测文档的语法元素。

Python

SDK for Python (Boto3)

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

创建一个包装 Amazon Comprehend 操作的类。

```
import logging
from pprint import pprint
import boto3
from botocore.exceptions import ClientError

logger = logging.getLogger(__name__)

class ComprehendDetect:
    """Encapsulates Comprehend detection functions."""

    def __init__(self, comprehend_client):
        """
        :param comprehend_client: A Boto3 Comprehend client.
        """
        self.comprehend_client = comprehend_client


    def detect_languages(self, text):
        """
```

```
        Detects languages used in a document.

        :param text: The document to inspect.
        :return: The list of languages along with their confidence scores.
        """
        try:
            response = self.comprehend_client.detect_dominant_language(Text=text)
            languages = response["Languages"]
            logger.info("Detected %s languages.", len(languages))
        except ClientError:
            logger.exception("Couldn't detect languages.")
            raise
        else:
            return languages


    def detect_entities(self, text, language_code):
        """
        Detects entities in a document. Entities can be things like people and
places
        or other common terms.

        :param text: The document to inspect.
        :param language_code: The language of the document.
        :return: The list of entities along with their confidence scores.
        """
        try:
            response = self.comprehend_client.detect_entities(
                Text=text, LanguageCode=language_code
            )
            entities = response["Entities"]
            logger.info("Detected %s entities.", len(entities))
        except ClientError:
            logger.exception("Couldn't detect entities.")
            raise
        else:
            return entities


    def detect_key_phrases(self, text, language_code):
        """
        Detects key phrases in a document. A key phrase is typically a noun and
its
        modifiers.
```

```
        :param text: The document to inspect.
        :param language_code: The language of the document.
        :return: The list of key phrases along with their confidence scores.
        """
        try:
            response = self.comprehend_client.detect_key_phrases(
                Text=text, LanguageCode=language_code
            )
            phrases = response["KeyPhrases"]
            logger.info("Detected %s phrases.", len(phrases))
        except ClientError:
            logger.exception("Couldn't detect phrases.")
            raise
        else:
            return phrases


    def detect_pii(self, text, language_code):
        """
        Detects personally identifiable information (PII) in a document. PII can
be
        things like names, account numbers, or addresses.

        :param text: The document to inspect.
        :param language_code: The language of the document.
        :return: The list of PII entities along with their confidence scores.
        """
        try:
            response = self.comprehend_client.detect_pii_entities(
                Text=text, LanguageCode=language_code
            )
            entities = response["Entities"]
            logger.info("Detected %s PII entities.", len(entities))
        except ClientError:
            logger.exception("Couldn't detect PII entities.")
            raise
        else:
            return entities


    def detect_sentiment(self, text, language_code):
        """
        Detects the overall sentiment expressed in a document. Sentiment can
```

```
        be positive, negative, neutral, or a mixture.

        :param text: The document to inspect.
        :param language_code: The language of the document.
        :return: The sentiments along with their confidence scores.
        """
        try:
            response = self.comprehend_client.detect_sentiment(
                Text=text, LanguageCode=language_code
            )
            logger.info("Detected primary sentiment %s.", response["Sentiment"])
        except ClientError:
            logger.exception("Couldn't detect sentiment.")
            raise
        else:
            return response


    def detect_syntax(self, text, language_code):
        """
        Detects syntactical elements of a document. Syntax tokens are portions of
        text along with their use as parts of speech, such as nouns, verbs, and
        interjections.

        :param text: The document to inspect.
        :param language_code: The language of the document.
        :return: The list of syntax tokens along with their confidence scores.
        """
        try:
            response = self.comprehend_client.detect_syntax(
                Text=text, LanguageCode=language_code
            )
            tokens = response["SyntaxTokens"]
            logger.info("Detected %s syntax tokens.", len(tokens))
        except ClientError:
            logger.exception("Couldn't detect syntax.")
            raise
        else:
            return tokens
```

调用包装类上的函数来检测文档中的实体、短语等。

```
def usage_demo():
    print("-" * 88)
    print("Welcome to the Amazon Comprehend detection demo!")
    print("-" * 88)

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

    comp_detect = ComprehendDetect(boto3.client("comprehend"))
    with open("detect_sample.txt") as sample_file:
        sample_text = sample_file.read()

    demo_size = 3

    print("Sample text used for this demo:")
    print("-" * 88)
    print(sample_text)
    print("-" * 88)

    print("Detecting languages.")
    languages = comp_detect.detect_languages(sample_text)
    pprint(languages)
    lang_code = languages[0]["LanguageCode"]

    print("Detecting entities.")
    entities = comp_detect.detect_entities(sample_text, lang_code)
    print(f"The first {demo_size} are:")
    pprint(entities[:demo_size])

    print("Detecting key phrases.")
    phrases = comp_detect.detect_key_phrases(sample_text, lang_code)
    print(f"The first {demo_size} are:")
    pprint(phrases[:demo_size])

    print("Detecting personally identifiable information (PII).")
    pii_entities = comp_detect.detect_pii(sample_text, lang_code)
    print(f"The first {demo_size} are:")
    pprint(pii_entities[:demo_size])

    print("Detecting sentiment.")
    sentiment = comp_detect.detect_sentiment(sample_text, lang_code)
    print(f"Sentiment: {sentiment['Sentiment']}")
    print("SentimentScore:")
```

```
    pprint(sentiment["SentimentScore"])

    print("Detecting syntax elements.")
    syntax_tokens = comp_detect.detect_syntax(sample_text, lang_code)
    print(f"The first {demo_size} are:")
    pprint(syntax_tokens[:demo_size])

    print("Thanks for watching!")
    print("-" * 88)
```

- 有关 API 详细信息，请参阅《AWS SDK for Python (Boto3) API 参考》中的以下主题。
  - DetectDominantLanguage
  - DetectEntities
  - DetectKeyPhrases
  - DetectPiiEntities
  - DetectSentiment
  - DetectSyntax

有关 S AWS DK 开发者指南和代码示例的完整列表，请参阅将 Amazon Comprehend 与 SDK 配合 AWS 使用。本主题还包括有关入门的信息以及有关先前的 SDK 版本的详细信息。

使用软件开发工具包对示例数据运行 Amazon Comprehend 主题建模作业 AWS

以下代码示例展示了如何：

- 对示例数据运行 Amazon Comprehend 主题建模任务。
- 获取该任务的相关信息。
- 从 Amazon S3 提取任务输出数据。

Python

SDK for Python (Boto3)

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

创建一个包装类来调用 Amazon Comprehend 主题建模操作。

```
class ComprehendTopicModeler:
    """Encapsulates a Comprehend topic modeler."""

    def __init__(self, comprehend_client):
        """
        :param comprehend_client: A Boto3 Comprehend client.
        """
        self.comprehend_client = comprehend_client


    def start_job(
        self,
        job_name,
        input_bucket,
        input_key,
        input_format,
        output_bucket,
        output_key,
        data_access_role_arn,
    ):
        """
        Starts a topic modeling job. Input is read from the specified Amazon S3
        input bucket and written to the specified output bucket. Output data is
 stored
        in a tar archive compressed in gzip format. The job runs asynchronously,
 so you
        can call `describe_topics_detection_job` to get job status until it
        returns a status of SUCCEEDED.

        :param job_name: The name of the job.
        :param input_bucket: An Amazon S3 bucket that contains job input.
```

```
        :param input_key: The prefix used to find input data in the input
                          bucket. If multiple objects have the same prefix,
all
                          of them are used.
        :param input_format: The format of the input data, either one document
per
                          file or one document per line.
        :param output_bucket: The Amazon S3 bucket where output data is written.
        :param output_key: The prefix prepended to the output data.
        :param data_access_role_arn: The Amazon Resource Name (ARN) of a role
that
                                     grants Comprehend permission to read from
the
                                     input bucket and write to the output bucket.
        :return: Information about the job, including the job ID.
        """
        try:
            response = self.comprehend_client.start_topics_detection_job(
                JobName=job_name,
                DataAccessRoleArn=data_access_role_arn,
                InputDataConfig={
                    "S3Uri": f"s3://{input_bucket}/{input_key}",
                    "InputFormat": input_format.value,
                },
                OutputDataConfig={"S3Uri": f"s3://{output_bucket}/{output_key}"},
            )
            logger.info("Started topic modeling job %s.", response["JobId"])
        except ClientError:
            logger.exception("Couldn't start topic modeling job.")
            raise
        else:
            return response


    def describe_job(self, job_id):
        """
        Gets metadata about a topic modeling job.

        :param job_id: The ID of the job to look up.
        :return: Metadata about the job.
        """
        try:
            response = self.comprehend_client.describe_topics_detection_job(
                JobId=job_id
```

```
            )
            job = response["TopicsDetectionJobProperties"]
            logger.info("Got topic detection job %s.", job_id)
        except ClientError:
            logger.exception("Couldn't get topic detection job %s.", job_id)
            raise
        else:
            return job


    def list_jobs(self):
        """
        Lists topic modeling jobs for the current account.

        :return: The list of jobs.
        """
        try:
            response = self.comprehend_client.list_topics_detection_jobs()
            jobs = response["TopicsDetectionJobPropertiesList"]
            logger.info("Got %s topic detection jobs.", len(jobs))
        except ClientError:
            logger.exception("Couldn't get topic detection jobs.")
            raise
        else:
            return jobs
```

使用包装器类运行主题建模任务并获取任务数据。

```
def usage_demo():
    print("-" * 88)
    print("Welcome to the Amazon Comprehend topic modeling demo!")
    print("-" * 88)

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

    input_prefix = "input/"
    output_prefix = "output/"
    demo_resources = ComprehendDemoResources(
        boto3.resource("s3"), boto3.resource("iam")
    )
```

```
    topic_modeler = ComprehendTopicModeler(boto3.client("comprehend"))

    print("Setting up storage and security resources needed for the demo.")
    demo_resources.setup("comprehend-topic-modeler-demo")
    print("Copying sample data from public bucket into input bucket.")
    demo_resources.bucket.copy(
        {"Bucket": "public-sample-us-west-2", "Key": "TopicModeling/Sample.txt"},
        f"{input_prefix}sample.txt",
    )

    print("Starting topic modeling job on sample data.")
    job_info = topic_modeler.start_job(
        "demo-topic-modeling-job",
        demo_resources.bucket.name,
        input_prefix,
        JobInputFormat.per_line,
        demo_resources.bucket.name,
        output_prefix,
        demo_resources.data_access_role.arn,
    )

    print(
        f"Waiting for job {job_info['JobId']} to complete. This typically takes "
        f"20 - 30 minutes."
    )
    job_waiter = JobCompleteWaiter(topic_modeler.comprehend_client)
    job_waiter.wait(job_info["JobId"])

    job = topic_modeler.describe_job(job_info["JobId"])
    print(f"Job {job['JobId']} complete:")
    pprint(job)

    print(
        f"Getting job output data from the output Amazon S3 bucket: "
        f"{job['OutputDataConfig']['S3Uri']}."
    )
    job_output = demo_resources.extract_job_output(job)
    lines = 10
    print(f"First {lines} lines of document topics output:")
    pprint(job_output["doc-topics.csv"]["data"][:lines])
    print(f"First {lines} lines of terms output:")
    pprint(job_output["topic-terms.csv"]["data"][:lines])

    print("Cleaning up resources created for the demo.")
```

```
    demo_resources.cleanup()

    print("Thanks for watching!")
    print("-" * 88)
```

- 有关 API 详细信息，请参阅《AWS SDK for Python (Boto3) API 参考》中的以下主题。
  - DescribeTopicsDetectionJob
  - ListTopicsDetectionJobs
  - StartTopicsDetectionJob

有关 S AWS DK 开发者指南和代码示例的完整列表，请参阅将 Amazon Comprehend 与 SDK 配合 AWS 使用。本主题还包括有关入门的信息以及有关先前的 SDK 版本的详细信息。

训练自定义 Amazon Comprehend 分类器并使用软件开发工具包对文档进行分类 AWS

以下代码示例展示了如何：

- 创建 Amazon Comprehend 多标签分类器。
- 在示例数据上训练分类器。
- 对第二组数据运行分类任务。
- 从 Amazon S3 提取任务输出数据。

Python

SDK for Python (Boto3)

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

创建一个包装类来调用 Amazon Comprehend 文档分类器操作。

```
class ComprehendClassifier:
    """Encapsulates an Amazon Comprehend custom classifier."""

    def __init__(self, comprehend_client):
        """
        :param comprehend_client: A Boto3 Comprehend client.
        """
        self.comprehend_client = comprehend_client
        self.classifier_arn = None


    def create(
        self,
        name,
        language_code,
        training_bucket,
        training_key,
        data_access_role_arn,
        mode,
    ):
        """
        Creates a custom classifier. After the classifier is created, it
immediately
        starts training on the data found in the specified Amazon S3 bucket.
Training
        can take 30 minutes or longer. The `describe_document_classifier`
function
        can be used to get training status and returns a status of TRAINED when
the
        classifier is ready to use.

        :param name: The name of the classifier.
        :param language_code: The language the classifier can operate on.
        :param training_bucket: The Amazon S3 bucket that contains the training
data.
        :param training_key: The prefix used to find training data in the
training
                             bucket. If multiple objects have the same prefix,
all
                             of them are used.
        :param data_access_role_arn: The Amazon Resource Name (ARN) of a role
that
```

```
                                    grants Comprehend permission to read from
 the
                                    training bucket.
        :return: The ARN of the newly created classifier.
        """
        try:
            response = self.comprehend_client.create_document_classifier(
                DocumentClassifierName=name,
                LanguageCode=language_code,
                InputDataConfig={"S3Uri": f"s3://{training_bucket}/
{training_key}"},
                DataAccessRoleArn=data_access_role_arn,
                Mode=mode.value,
            )
            self.classifier_arn = response["DocumentClassifierArn"]
            logger.info("Started classifier creation. Arn is: %s.",
 self.classifier_arn)
        except ClientError:
            logger.exception("Couldn't create classifier %s.", name)
            raise
        else:
            return self.classifier_arn


    def describe(self, classifier_arn=None):
        """
        Gets metadata about a custom classifier, including its current status.

        :param classifier_arn: The ARN of the classifier to look up.
        :return: Metadata about the classifier.
        """
        if classifier_arn is not None:
            self.classifier_arn = classifier_arn
        try:
            response = self.comprehend_client.describe_document_classifier(
                DocumentClassifierArn=self.classifier_arn
            )
            classifier = response["DocumentClassifierProperties"]
            logger.info("Got classifier %s.", self.classifier_arn)
        except ClientError:
            logger.exception("Couldn't get classifier %s.", self.classifier_arn)
            raise
        else:
            return classifier
```

```
    def list(self):
        """
        Lists custom classifiers for the current account.

        :return: The list of classifiers.
        """
        try:
            response = self.comprehend_client.list_document_classifiers()
            classifiers = response["DocumentClassifierPropertiesList"]
            logger.info("Got %s classifiers.", len(classifiers))
        except ClientError:
            logger.exception(
                "Couldn't get classifiers.",
            )
            raise
        else:
            return classifiers


    def delete(self):
        """
        Deletes the classifier.
        """
        try:
            self.comprehend_client.delete_document_classifier(
                DocumentClassifierArn=self.classifier_arn
            )
            logger.info("Deleted classifier %s.", self.classifier_arn)
            self.classifier_arn = None
        except ClientError:
            logger.exception("Couldn't deleted classifier %s.",
self.classifier_arn)
            raise


    def start_job(
        self,
        job_name,
        input_bucket,
        input_key,
        input_format,
        output_bucket,
```

```
        output_key,
        data_access_role_arn,
    ):
        """
        Starts a classification job. The classifier must be trained or the job
        will fail. Input is read from the specified Amazon S3 input bucket and
        written to the specified output bucket. Output data is stored in a tar
        archive compressed in gzip format. The job runs asynchronously, so you
can
        call `describe_document_classification_job` to get job status until it
        returns a status of SUCCEEDED.

        :param job_name: The name of the job.
        :param input_bucket: The Amazon S3 bucket that contains input data.
        :param input_key: The prefix used to find input data in the input
                          bucket. If multiple objects have the same prefix, all
                          of them are used.
        :param input_format: The format of the input data, either one document
per
                             file or one document per line.
        :param output_bucket: The Amazon S3 bucket where output data is written.
        :param output_key: The prefix prepended to the output data.
        :param data_access_role_arn: The Amazon Resource Name (ARN) of a role
that
                                     grants Comprehend permission to read from
the
                                     input bucket and write to the output bucket.
        :return: Information about the job, including the job ID.
        """
        try:
            response = self.comprehend_client.start_document_classification_job(
                DocumentClassifierArn=self.classifier_arn,
                JobName=job_name,
                InputDataConfig={
                    "S3Uri": f"s3://{input_bucket}/{input_key}",
                    "InputFormat": input_format.value,
                },
                OutputDataConfig={"S3Uri": f"s3://{output_bucket}/{output_key}"},
                DataAccessRoleArn=data_access_role_arn,
            )
            logger.info(
                "Document classification job %s is %s.", job_name,
response["JobStatus"]
            )
```

```
        except ClientError:
            logger.exception("Couldn't start classification job %s.", job_name)
            raise
        else:
            return response


    def describe_job(self, job_id):
        """
        Gets metadata about a classification job.

        :param job_id: The ID of the job to look up.
        :return: Metadata about the job.
        """
        try:
            response =
self.comprehend_client.describe_document_classification_job(
                JobId=job_id
            )
            job = response["DocumentClassificationJobProperties"]
            logger.info("Got classification job %s.", job["JobName"])
        except ClientError:
            logger.exception("Couldn't get classification job %s.", job_id)
            raise
        else:
            return job


    def list_jobs(self):
        """
        Lists the classification jobs for the current account.

        :return: The list of jobs.
        """
        try:
            response = self.comprehend_client.list_document_classification_jobs()
            jobs = response["DocumentClassificationJobPropertiesList"]
            logger.info("Got %s document classification jobs.", len(jobs))
        except ClientError:
            logger.exception(
                "Couldn't get document classification jobs.",
            )
            raise
        else:
```

```
            return jobs
```

创建帮组运行场景的类。

```
class ClassifierDemo:
    """
    Encapsulates functions used to run the demonstration.
    """

    def __init__(self, demo_resources):
        """
        :param demo_resources: A ComprehendDemoResources class that manages
 resources
                               for the demonstration.
        """
        self.demo_resources = demo_resources
        self.training_prefix = "training/"
        self.input_prefix = "input/"
        self.input_format = JobInputFormat.per_line
        self.output_prefix = "output/"

    def setup(self):
        """Creates AWS resources used by the demo."""
        self.demo_resources.setup("comprehend-classifier-demo")

    def cleanup(self):
        """Deletes AWS resources used by the demo."""
        self.demo_resources.cleanup()

    @staticmethod
    def _sanitize_text(text):
        """Removes characters that cause errors for the document parser."""
        return text.replace("\r", " ").replace("\n", " ").replace(",", ";")

    @staticmethod
    def _get_issues(query, issue_count):
        """
        Gets issues from GitHub using the specified query parameters.
```

```
        :param query: The query string used to request issues from the GitHub
 API.
        :param issue_count: The number of issues to retrieve.
        :return: The list of issues retrieved from GitHub.
        """
        issues = []
        logger.info("Requesting issues from %s?%s.", GITHUB_SEARCH_URL, query)
        response = requests.get(f"{GITHUB_SEARCH_URL}?
{query}&per_page={issue_count}")
        if response.status_code == 200:
            issue_page = response.json()["items"]
            logger.info("Got %s issues.", len(issue_page))
            issues = [
                {
                    "title": ClassifierDemo._sanitize_text(issue["title"]),
                    "body": ClassifierDemo._sanitize_text(issue["body"]),
                    "labels": {label["name"] for label in issue["labels"]},
                }
                for issue in issue_page
            ]
        else:
            logger.error(
                "GitHub returned error code %s with message %s.",
                response.status_code,
                response.json(),
            )
        logger.info("Found %s issues.", len(issues))
        return issues

    def get_training_issues(self, training_labels):
        """
        Gets issues used for training the custom classifier. Training issues are
        closed issues from the Boto3 repo that have known labels. Comprehend
        requires a minimum of ten training issues per label.

        :param training_labels: The issue labels to use for training.
        :return: The set of issues used for training.
        """
        issues = []
        per_label_count = 15
        for label in training_labels:
            issues += self._get_issues(
                f"q=type:issue+repo:boto/boto3+state:closed+label:{label}",
                per_label_count,
```

```
            )
            for issue in issues:
                issue["labels"] = issue["labels"].intersection(training_labels)
        return issues

    def get_input_issues(self, training_labels):
        """
        Gets input issues from GitHub. For demonstration purposes, input issues
        are open issues from the Boto3 repo with known labels, though in practice
        any issue could be submitted to the classifier for labeling.

        :param training_labels: The set of labels to query for.
        :return: The set of issues used for input.
        """
        issues = []
        per_label_count = 5
        for label in training_labels:
            issues += self._get_issues(
                f"q=type:issue+repo:boto/boto3+state:open+label:{label}",
                per_label_count,
            )
        return issues

    def upload_issue_data(self, issues, training=False):
        """
        Uploads issue data to an Amazon S3 bucket, either for training or for
input.
        The data is first put into the format expected by Comprehend. For
training,
        the set of pipe-delimited labels is prepended to each document. For
        input, labels are not sent.

        :param issues: The set of issues to upload to Amazon S3.
        :param training: Indicates whether the issue data is used for training or
                         input.
        """
        try:
            obj_key = (
                self.training_prefix if training else self.input_prefix
            ) + "issues.txt"
            if training:
                issue_strings = [
                    f"{'|'.join(issue['labels'])},{issue['title']}
{issue['body']}"
```

```
                    for issue in issues
                ]
            else:
                issue_strings = [
                    f"{issue['title']} {issue['body']}" for issue in issues
                ]
            issue_bytes = BytesIO("\n".join(issue_strings).encode("utf-8"))
            self.demo_resources.bucket.upload_fileobj(issue_bytes, obj_key)
            logger.info(
                "Uploaded data as %s to bucket %s.",
                obj_key,
                self.demo_resources.bucket.name,
            )
        except ClientError:
            logger.exception(
                "Couldn't upload data to bucket %s.",
self.demo_resources.bucket.name
            )
            raise

    def extract_job_output(self, job):
        """Extracts job output from Amazon S3."""
        return self.demo_resources.extract_job_output(job)

    @staticmethod
    def reconcile_job_output(input_issues, output_dict):
        """
        Reconciles job output with the list of input issues. Because the input
issues
        have known labels, these can be compared with the labels added by the
        classifier to judge the accuracy of the output.

        :param input_issues: The list of issues used as input.
        :param output_dict: The dictionary of data that is output by the
classifier.
        :return: The list of reconciled input and output data.
        """
        reconciled = []
        for archive in output_dict.values():
            for line in archive["data"]:
                in_line = int(line["Line"])
                in_labels = input_issues[in_line]["labels"]
                out_labels = {
                    label["Name"]
```

```
                    for label in line["Labels"]
                    if float(label["Score"]) > 0.3
                }
                reconciled.append(
                    f"{line['File']}, line {in_line} has labels {in_labels}.\n"
                    f"\tClassifier assigned {out_labels}."
                )
        logger.info("Reconciled input and output labels.")
        return reconciled
```

使用已知标签对分类器进行一系列 GitHub 问题训练，然后将第二组 GitHub 问题发送给分类器以便对其进行标记。

```
def usage_demo():
    print("-" * 88)
    print("Welcome to the Amazon Comprehend custom document classifier demo!")
    print("-" * 88)

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

    comp_demo = ClassifierDemo(
        ComprehendDemoResources(boto3.resource("s3"), boto3.resource("iam"))
    )
    comp_classifier = ComprehendClassifier(boto3.client("comprehend"))
    classifier_trained_waiter = ClassifierTrainedWaiter(
        comp_classifier.comprehend_client
    )
    training_labels = {"bug", "feature-request", "dynamodb", "s3"}

    print("Setting up storage and security resources needed for the demo.")
    comp_demo.setup()

    print("Getting training data from GitHub and uploading it to Amazon S3.")
    training_issues = comp_demo.get_training_issues(training_labels)
    comp_demo.upload_issue_data(training_issues, True)

    classifier_name = "doc-example-classifier"
    print(f"Creating document classifier {classifier_name}.")
    comp_classifier.create(
        classifier_name,
```

```
        "en",
        comp_demo.demo_resources.bucket.name,
        comp_demo.training_prefix,
        comp_demo.demo_resources.data_access_role.arn,
        ClassifierMode.multi_label,
    )
    print(
        f"Waiting until {classifier_name} is trained. This typically takes "
        f"30-40 minutes."
    )
    classifier_trained_waiter.wait(comp_classifier.classifier_arn)

    print(f"Classifier {classifier_name} is trained:")
    pprint(comp_classifier.describe())

    print("Getting input data from GitHub and uploading it to Amazon S3.")
    input_issues = comp_demo.get_input_issues(training_labels)
    comp_demo.upload_issue_data(input_issues)

    print("Starting classification job on input data.")
    job_info = comp_classifier.start_job(
        "issue_classification_job",
        comp_demo.demo_resources.bucket.name,
        comp_demo.input_prefix,
        comp_demo.input_format,
        comp_demo.demo_resources.bucket.name,
        comp_demo.output_prefix,
        comp_demo.demo_resources.data_access_role.arn,
    )
    print(f"Waiting for job {job_info['JobId']} to complete.")
    job_waiter = JobCompleteWaiter(comp_classifier.comprehend_client)
    job_waiter.wait(job_info["JobId"])

    job = comp_classifier.describe_job(job_info["JobId"])
    print(f"Job {job['JobId']} complete:")
    pprint(job)

    print(
        f"Getting job output data from Amazon S3: "
        f"{job['OutputDataConfig']['S3Uri']}."
    )
    job_output = comp_demo.extract_job_output(job)
    print("Job output:")
    pprint(job_output)
```

```
    print("Reconciling job output with labels from GitHub:")
    reconciled_output = comp_demo.reconcile_job_output(input_issues, job_output)
    print(*reconciled_output, sep="\n")

    answer = input(f"Do you want to delete the classifier {classifier_name} (y/
n)? ")
    if answer.lower() == "y":
        print(f"Deleting {classifier_name}.")
        comp_classifier.delete()

    print("Cleaning up resources created for the demo.")
    comp_demo.cleanup()

    print("Thanks for watching!")
    print("-" * 88)
```

- 有关 API 详细信息，请参阅《AWS SDK for Python (Boto3) API 参考》中的以下主题。
  - CreateDocumentClassifier
  - DeleteDocumentClassifier
  - DescribeDocumentClassificationJob
  - DescribeDocumentClassifier
  - ListDocumentClassificationJobs
  - ListDocumentClassifiers
  - StartDocumentClassificationJob

有关 S AWS DK 开发者指南和代码示例的完整列表，请参阅将 Amazon Comprehend 与 SDK 配合 AWS 使用。本主题还包括有关入门的信息以及有关先前的 SDK 版本的详细信息。

使用软件开发工具包的 Amazon Comprehend 的跨服务示例 AWS

以下示例应用程序使用 AWS 软件开发工具包将 Amazon Comprehend 与其他应用程序组合在一起。AWS 服务每个示例都包含一个指向的链接 GitHub，您可以在其中找到有关如何设置和运行应用程序的说明。

示例

- 构建 Amazon Transcribe 流式传输应用程序
- 创建 Amazon Lex 聊天机器人来吸引您的网站访客
- 使用 Amazon SQS 创建用于发送和检索消息的网络应用程序
- 创建用于分析客户反馈和合成音频的应用程序
- 使用 AWS SDK 检测从图像中提取的文本中的实体

构建 Amazon Transcribe 流式传输应用程序

以下代码示例展示如何构建可实时录制、转录与翻译实时音频，并通过电子邮件发送结果的应用程序。

JavaScript

适用于 JavaScript (v3) 的软件开发工具包

演示了如何使用 Amazon Transcribe 构建可实时录制、转录与翻译实时音频，并通过 Amazon Simple Email Service (Amazon SES) 以电子邮件发送结果的应用程序。

有关如何设置和运行的完整源代码和说明，请参阅上的完整示例GitHub。

本示例中使用的服务

- Amazon Comprehend
- Amazon SES
- Amazon Transcribe
- Amazon Translate

有关 S AWS DK 开发者指南和代码示例的完整列表，请参阅将 Amazon Comprehend 与 SDK 配合 AWS 使用。本主题还包括有关入门的信息以及有关先前的 SDK 版本的详细信息。

创建 Amazon Lex 聊天机器人来吸引您的网站访客

以下代码示例显示如何创建用于吸引网站访客的聊天机器人。

Java

适用于 Java 2.x 的 SDK

演示如何使用 Amazon Lex API 在 Web 应用程序中创建聊天机器人，以吸引网站访问者。

有关如何设置和运行的完整源代码和说明，请参阅上的完整示例GitHub。

本示例中使用的服务

- Amazon Comprehend
- Amazon Lex
- Amazon Translate

JavaScript

适用于 JavaScript (v3) 的软件开发工具包

展示如何使用 Amazon Lex API 在 Web 应用程序中创建聊天机器人，以吸引您的网站访问者。

有关如何设置和运行的完整源代码和说明，请参阅 AWS SDK for JavaScript 开发者指南中的构建 Amazon Lex 聊天机器人的完整示例。

本示例中使用的服务

- Amazon Comprehend
- Amazon Lex
- Amazon Translate

有关 S AWS DK 开发者指南和代码示例的完整列表，请参阅将 Amazon Comprehend 与 SDK 配合 AWS 使用。本主题还包括有关入门的信息以及有关先前的 SDK 版本的详细信息。

使用 Amazon SQS 创建用于发送和检索消息的网络应用程序

以下代码示例显示如何使用 Amazon SQS 创建消息传输应用程序。

Java

适用于 Java 2.x 的 SDK

演示如何使用 Amazon SQS API 开发用于发送和检索消息的 Spring REST API。

有关如何设置和运行的完整源代码和说明，请参阅上的完整示例GitHub。

本示例中使用的服务

- Amazon Comprehend
- Amazon SQS

Kotlin

适用于 Kotlin 的 SDK

演示如何使用 Amazon SQS API 开发用于发送和检索消息的 Spring REST API。

有关如何设置和运行的完整源代码和说明，请参阅上的完整示例GitHub。

本示例中使用的服务

- Amazon Comprehend
- Amazon SQS

有关 S AWS DK 开发者指南和代码示例的完整列表，请参阅将 Amazon Comprehend 与 SDK 配合 AWS 使用。本主题还包括有关入门的信息以及有关先前的 SDK 版本的详细信息。

创建用于分析客户反馈和合成音频的应用程序

以下代码示例显示如何创建应用程序来分析客户意见卡、翻译其母语、确定其情绪并根据译后的文本生成音频文件。

.NET

AWS SDK for .NET

此示例应用程序可分析并存储客户反馈卡。具体来说，它满足了纽约市一家虚构酒店的需求。酒店以实体意见卡的形式收集来自不同语种的客人的反馈。该反馈通过 Web 客户端上传到应用程序中。意见卡图片上传后，将执行以下步骤：

- 使用 Amazon Textract 从图片中提取文本。
- Amazon Comprehend 确定所提取文本的情绪及其语言。
- 使用 Amazon Translate 将所提取文本翻译为英语。
- Amazon Polly 根据所提取文本合成音频文件。

完整的应用程序可使用 AWS CDK 进行部署。有关源代码和部署说明，请参阅中的项目 GitHub。

本示例中使用的服务

- Amazon Comprehend
- Lambda
- Amazon Polly

- Amazon Textract
- Amazon Translate

Java

适用于 Java 2.x 的 SDK

此示例应用程序可分析并存储客户反馈卡。具体来说，它满足了纽约市一家虚构酒店的需求。酒店以实体意见卡的形式收集来自不同语种的客人的反馈。该反馈通过 Web 客户端上传到应用程序中。意见卡图片上传后，将执行以下步骤：

- 使用 Amazon Textract 从图片中提取文本。
- Amazon Comprehend 确定所提取文本的情绪及其语言。
- 使用 Amazon Translate 将所提取文本翻译为英语。
- Amazon Polly 根据所提取文本合成音频文件。

完整的应用程序可使用 AWS CDK 进行部署。有关源代码和部署说明，请参阅中的项目 GitHub。

本示例中使用的服务

- Amazon Comprehend
- Lambda
- Amazon Polly
- Amazon Textract
- Amazon Translate

JavaScript

适用于 JavaScript (v3) 的软件开发工具包

此示例应用程序可分析并存储客户反馈卡。具体来说，它满足了纽约市一家虚构酒店的需求。酒店以实体意见卡的形式收集来自不同语种的客人的反馈。该反馈通过 Web 客户端上传到应用程序中。意见卡图片上传后，将执行以下步骤：

- 使用 Amazon Textract 从图片中提取文本。
- Amazon Comprehend 确定所提取文本的情绪及其语言。
- 使用 Amazon Translate 将所提取文本翻译为英语。

- Amazon Polly 根据所提取文本合成音频文件。

完整的应用程序可使用 AWS CDK 进行部署。有关源代码和部署说明，请参阅中的项目 GitHub。以下摘录显示了在 Lambda 函数中 AWS SDK for JavaScript 是如何使用的。

```
import {
  ComprehendClient,
  DetectDominantLanguageCommand,
  DetectSentimentCommand,
} from "@aws-sdk/client-comprehend";

/**
 * Determine the language and sentiment of the extracted text.
 *
 * @param {{ source_text: string}} extractTextOutput
 */
export const handler = async (extractTextOutput) => {
  const comprehendClient = new ComprehendClient({});

  const detectDominantLanguageCommand = new DetectDominantLanguageCommand({
    Text: extractTextOutput.source_text,
  });

  // The source language is required for sentiment analysis and
  // translation in the next step.
  const { Languages } = await comprehendClient.send(
    detectDominantLanguageCommand,
  );

  const languageCode = Languages[0].LanguageCode;

  const detectSentimentCommand = new DetectSentimentCommand({
    Text: extractTextOutput.source_text,
    LanguageCode: languageCode,
  });

  const { Sentiment } = await comprehendClient.send(detectSentimentCommand);

  return {
    sentiment: Sentiment,
    language_code: languageCode,
  };
};
```

```
import {
  DetectDocumentTextCommand,
  TextractClient,
} from "@aws-sdk/client-textract";

/**
 * Fetch the S3 object from the event and analyze it using Amazon Textract.
 *
 * @param {import("@types/aws-lambda").EventBridgeEvent<"Object Created">}
 eventBridgeS3Event
 */
export const handler = async (eventBridgeS3Event) => {
  const textractClient = new TextractClient();

  const detectDocumentTextCommand = new DetectDocumentTextCommand({
    Document: {
      S3Object: {
        Bucket: eventBridgeS3Event.bucket,
        Name: eventBridgeS3Event.object,
      },
    },
  });

  // Textract returns a list of blocks. A block can be a line, a page, word, etc.
  // Each block also contains geometry of the detected text.
  // For more information on the Block type, see https://docs.aws.amazon.com/
textract/latest/dg/API_Block.html.
  const { Blocks } = await textractClient.send(detectDocumentTextCommand);

  // For the purpose of this example, we are only interested in words.
  const extractedWords = Blocks.filter((b) => b.BlockType === "WORD").map(
    (b) => b.Text,
  );

  return extractedWords.join(" ");
};
```

```
import { PollyClient, SynthesizeSpeechCommand } from "@aws-sdk/client-polly";
import { S3Client } from "@aws-sdk/client-s3";
import { Upload } from "@aws-sdk/lib-storage";

/**
 * Synthesize an audio file from text.
```

```
 *
 * @param {{ bucket: string, translated_text: string, object: string}}
 sourceDestinationConfig
 */
export const handler = async (sourceDestinationConfig) => {
  const pollyClient = new PollyClient({});

  const synthesizeSpeechCommand = new SynthesizeSpeechCommand({
    Engine: "neural",
    Text: sourceDestinationConfig.translated_text,
    VoiceId: "Ruth",
    OutputFormat: "mp3",
  });

  const { AudioStream } = await pollyClient.send(synthesizeSpeechCommand);

  const audioKey = `${sourceDestinationConfig.object}.mp3`;

  // Store the audio file in S3.
  const s3Client = new S3Client();
  const upload = new Upload({
    client: s3Client,
    params: {
      Bucket: sourceDestinationConfig.bucket,
      Key: audioKey,
      Body: AudioStream,
      ContentType: "audio/mp3",
    },
  });

  await upload.done();
  return audioKey;
};
```

```
import {
  TranslateClient,
  TranslateTextCommand,
} from "@aws-sdk/client-translate";

/**
 * Translate the extracted text to English.
 *
```

```
 * @param {{ extracted_text: string, source_language_code: string}}
 textAndSourceLanguage
 */
export const handler = async (textAndSourceLanguage) => {
  const translateClient = new TranslateClient({});

  const translateCommand = new TranslateTextCommand({
    SourceLanguageCode: textAndSourceLanguage.source_language_code,
    TargetLanguageCode: "en",
    Text: textAndSourceLanguage.extracted_text,
  });

  const { TranslatedText } = await translateClient.send(translateCommand);

  return { translated_text: TranslatedText };
};
```

本示例中使用的服务

- Amazon Comprehend
- Lambda
- Amazon Polly
- Amazon Textract
- Amazon Translate

Ruby

适用于 Ruby 的 SDK

此示例应用程序可分析并存储客户反馈卡。具体来说，它满足了纽约市一家虚构酒店的需求。酒店以实体意见卡的形式收集来自不同语种的客人的反馈。该反馈通过 Web 客户端上传到应用程序中。意见卡图片上传后，将执行以下步骤：

- 使用 Amazon Textract 从图片中提取文本。
- Amazon Comprehend 确定所提取文本的情绪及其语言。
- 使用 Amazon Translate 将所提取文本翻译为英语。
- Amazon Polly 根据所提取文本合成音频文件。

完整的应用程序可使用 AWS CDK 进行部署。有关源代码和部署说明，请参阅中的项目 GitHub。

本示例中使用的服务

- Amazon Comprehend
- Lambda
- Amazon Polly
- Amazon Textract
- Amazon Translate

有关 S AWS DK 开发者指南和代码示例的完整列表，请参阅将 Amazon Comprehend 与 SDK 配合 AWS 使用。本主题还包括有关入门的信息以及有关先前的 SDK 版本的详细信息。

使用 AWS SDK 检测从图像中提取的文本中的实体

以下代码示例显示了如何使用 Amazon Comprehend 检测 Amazon Textract 从存储在 Amazon S3 内的图像中提取的文本中的实体。

Python

SDK for Python (Boto3)

演示如何使用 Jupyter 笔记本 AWS SDK for Python (Boto3) 中的来检测从图像中提取的文本中的实体。此示例使用 Amazon Textract 从存储在 Amazon Simple Storage Service (Amazon S3) 内的图像中提取文本，并使用 Amazon Comprehend 检测提取文本中的实体。

此示例是 Jupyter 笔记本，必须在可以托管笔记本电脑的环境中运行。有关如何使用 Amazon 运行示例的说明 SageMaker，请参阅 TextractAndComprehendNotebook.ipyn b 中的说明。

有关如何设置和运行的完整源代码和说明，请参阅上的完整示例GitHub。

本示例中使用的服务

- Amazon Comprehend
- Amazon S3
- Amazon Textract

有关 S AWS DK 开发者指南和代码示例的完整列表，请参阅将 Amazon Comprehend 与 SDK 配合 AWS 使用。本主题还包括有关入门的信息以及有关先前的 SDK 版本的详细信息。

Amazon Comprehend 中的安全性

AWS 十分重视云安全性。作为 AWS 客户，您将从专为满足大多数安全敏感型企业的要求而打造的数据中心和网络架构中受益。

安全性是 AWS 和您的共同责任。责任共担模式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 合规性计划的一部分。要了解适用于 Amazon Comprehend 的合规性计划，请参阅按合规性计划提供的范围内的 AWS 服务。
- 云中的安全性：您的责任由您使用的 AWS 服务决定。您还需要对其他因素负责，包括数据的敏感性、公司的要求以及适用的法律法规。

本文档帮助您了解如何在使用 Amazon Comprehend 时应用责任共担模式。以下主题说明如何配置 Amazon Comprehend 以实现您的安全性和合规性目标。您还会了解如何使用其他 AWS 服务以帮助您监控和保护您的 Amazon Comprehend 资源。

主题

Amazon Comprehend 中的数据保护

AWS 责任共担模式应用于 Amazon Comprehend 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解答。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，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）。这包括当您通过控制台、API、AWS CLI 或 AWS SDK 使用 Amazon Comprehend 或其他 AWS 服务 时。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，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。

主题

Amazon Comprehend 中的 KMS 加密

Amazon Comprehend 与 AWS Key Management Service (AWS KMS) 配合使用，为您的数据提供增强的加密。Amazon S3 已经允许您在创建文本分析、主题建模或自定义 Amazon Comprehend 作业时对输入文档进行加密。AWS KMS 与集成后，您可以加密存储卷中用于“Start*”和“Create*”作业的数据，并使用您自己的 KMS 密钥对“Start*”作业的输出结果进行加密。

对于 AWS Management Console，Amazon Comprehend 使用自己的 KMS 密钥对自定义模型进行加密。对于 AWS CLI，Amazon Comprehend 可以使用自己的 KMS 密钥或提供的客户自主管理型密钥 (CMK) 对自定义模型进行加密。

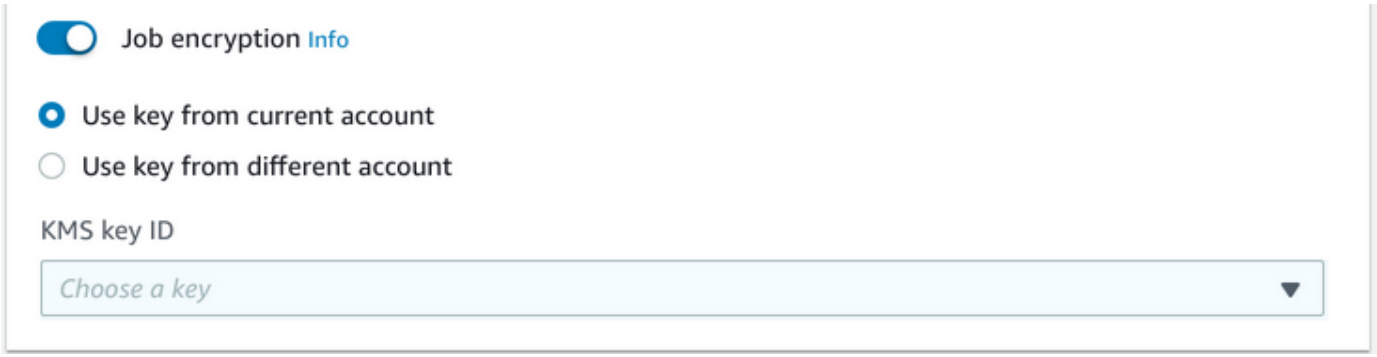
使用 AWS Management Console 的 KMS 加密

使用控制台时，有两个加密选项可用：

- 卷加密
- 输出结果加密

启用卷加密

1. 在作业设置下，选择作业加密选项。



2. 选择 KMS 客户自主管理型密钥 (CMK) 来自您当前使用的账户还是其他账户。如果您想要使用来自当前账户的密钥，请从 KMS 密钥 ID 选择密钥别名。如果您使用来自其他账户的密钥，则必须输入密钥的 ARN。

启用输出结果加密

1. 在输出设置下，选择加密选项。

2. 选择 客户自主管理型密钥 (CMK) 是来自您当前使用的账户还是其他账户。如果您要使用来自当前账户的密钥，请从 KMS 密钥 ID 选择密钥 ID。如果您使用来自其他账户的密钥，则必须输入密钥的 ARN。

如果您之前已在 S3 输入文档上使用 SSE-KMS 设置加密，则这可以为您提供额外的安全性。但是，如果您这样做，则所使用的 IAM 角色必须拥有加密输入文档的 KMS 密钥的 `kms:Decrypt` 权限。有关更多信息，请参阅 使用 KMS 加密所需的权限。

使用 API 操作进行 KMS 加密

所有 Amazon Comprehend `Start*` 和 `Create*` API 操作都支持 KMS 加密的输入文档。如果原始作业提供 `KmsKeyId` 作为输入，则 `Describe*` 和 `List*` API 操作会返回 `OutputDataConfig` 中的 `KmsKeyId`。如果未将其作为输入提供，则不会将其返回。

这可以在以下 AWS CLI 示例中使用StartEntitiesDetectionJob操作看出：

```
aws comprehend start-entities-detection-job \
     --region region \
     --data-access-role-arn "data access role arn" \
     --entity-recognizer-arn "entity recognizer arn" \
     --input-data-config "S3Uri=s3://Bucket Name/Bucket Path" \
     --job-name job name \
     --language-code en \
     --output-data-config "KmsKeyId=Output S3 KMS key ID" "S3Uri=s3://Bucket
 Name/Bucket Path/" \
     --volumekmskeyid "Volume KMS key ID"
```

> **Note**
>
> 此示例的格式适用于 Unix、Linux 和 macOS。对于 Windows，请将每行末尾的反斜杠 (\) Unix 行继续符替换为脱字号 (^)。

使用 API 操作进行客户自主管理型密钥 (CMK) 加密

Amazon Comprehend 自定义模型 API 操作 `CreateEntityRecognizer`、`CreateDocumentClassifier` 和 `CreateEndpoint`，支持通过 AWS CLI 使用客户自主管理型密钥进行加密。

您可以使用 IAM policy 来允许委托人使用或管理客户自主管理型密钥。这些密钥在策略语句的 `Resource` 元素中指定。最佳做法是，将客户自主管理型密钥限制为仅限委托人必须在您的策略语句中使用的密钥。

以下 AWS CLI 示例使用CreateEntityRecognizer操作创建具有模型加密功能的自定义实体识别器：

```
aws comprehend create-entity-recognizer \
     --recognizer-name name \
     --data-access-role-arn data access role arn \
     --language-code en \
     --model-kms-key-id Model KMS Key ID \
     --input-data-config file:///path/input-data-config.json
```

> **Note**
>
> 此示例的格式适用于 Unix、Linux 和 macOS。对于 Windows，请将每行末尾的反斜杠 (\) Unix 行继续符替换为脱字号 (^)。

防止跨服务混淆座席

混淆座席问题是一个安全性问题，即不具有操作执行权限的实体可能会迫使具有更高权限的实体执行该操作。在 AWS 中，跨服务模拟可能会导致混淆座席问题。一个服务（调用服务）调用另一项服务（被调用服务)时，可能会发生跨服务模拟。可以操纵调用服务，使用其权限以在其他情况下该服务不应有访问权限的方式对另一个客户的资源进行操作。为防止这种情况，AWS 提供可帮助您保护所有服务的数据的工具，而这些服务中的服务主体有权限访问账户中的资源。

我们建议在资源策略中使用 aws:SourceArn 和 aws:SourceAccount 全局条件上下文键，以限制 Amazon Comprehend 为其他服务提供的资源访问权限。如果使用两个全局条件上下文键，在同一策略语句中使用时，aws:SourceAccount 值和 aws:SourceArn 值中的账户必须使用相同的账户 ID。

防范混淆代理问题最有效的方法是使用 aws:SourceArn 全局条件上下文键和资源的完整 ARN。如果不知道资源的完整 ARN，或者正在指定多个资源，请针对 ARN 未知部分使用带有通配符 (*) 的 aws:SourceArn 全局上下文条件键。例如，arn:aws:*servicename*::*123456789012*:*。

使用源账户关联

以下示例演示如何使用 Amazon Comprehend 中的 aws:SourceAccount 全局条件上下文键。

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

```
    "Effect": "Allow",
    "Principal": {
          "Service": "comprehend.amazonaws.com"
    },
    "Action": "sts:AssumeRole",
    "Condition": {
        "StringEquals": {
             "aws:SourceAccount":"111122223333"
        }
    }
  }
}
```

加密模型终端节点的信任策略

您需要创建信任策略才能为加密模型创建或更新终端节点。将 aws:SourceAccount 值设置为您的账户 ID。如果您使用 ArnEquals 条件，请将 aws:SourceArn 值设置为终端节点的 ARN。

```
{
 "Version": "2012-10-17",
 "Statement": [
    {
        "Sid": "",
        "Effect": "Allow",
        "Principal": {
            "Service": "comprehend.amazonaws.com"
        },
        "Action": "sts:AssumeRole",
        "Condition": {
            "StringEquals": {
               "aws:SourceAccount": "111122223333"
            },
            "ArnEquals": {
               "aws:SourceArn": "arn:aws:comprehend:us-west-2:111122223333:document-
classifier-endpoint/endpoint-name"
            }
        }
    }
  ]
}
```

创建自定义模型

您需要创建信任策略才能创建自定义模型。将 aws:SourceAccount 值设置为您的账户 ID。如果您使用 ArnEquals 条件，请将 aws:SourceArn 值设置为自定义模型版本的 ARN。

```
{
 "Version": "2012-10-17",
  "Statement": [
      {
          "Sid": "",
          "Effect": "Allow",
          "Principal": {
              "Service": "comprehend.amazonaws.com"
          },
          "Action": "sts:AssumeRole",
          "Condition": {
              "StringEquals": {
                  "aws:SourceAccount": "111122223333"
              },
              "ArnEquals": {
                  "aws:SourceArn": "arn:aws:comprehend:us-west-2:111122223333:
                            document-classifier/smallest-classifier-test/
version/version-name"
              }
          }
      }
  ]
}
```

使用 Amazon 虚拟私有云保护工作

Amazon Comprehend 使用多种安全措施来确保您的数据在 Amazon Comprehend 使用期间存储在我们的任务容器中的安全。但是，任务容器会通过 Interne AWS t 访问资源，例如用于存储数据和模型工件的 Amazon S3 存储桶。

要控制对您的模型容器和数据的访问，我们建议您创建一个虚拟私有云 VPC，并配置该 VPC 以便无法通过互联网进行访问。有关创建和配置 VPC 的信息，请参阅《Amazon VPC 用户指南》中的 Amazon VPC 入门。使用 VPC 有助于保护您的数据，因为您可以配置 VPC 以使其不连接到互联网。使用 VPC，您还可以通过 VPC 流日志来监控进出任务容器的所有网络流量。有关更多信息，请参阅《Amazon VPC 用户指南》中的 VPC 流日志。

创建任务时，您通过指定子网和安全组来指定 VPC 配置。当您指定子网和安全组时，Amazon Comprehend 会创建与其中一个子网中的安全组关联的弹性网络接口 (ENI)。ENI 可将您的任务容器连接到 VPC 中的资源。有关 ENI 的信息，请参阅《Amazon VPC 用户指南》中的弹性网络接口。

> **Note**
>
> 对于任务，只能使用默认的租赁 VPC 配置子网，其中实例在共享硬件上运行。有关 VPC 租期属性的更多信息，请参阅 Amazon EC2 用户指南中的专用实例。

为 Amazon VPC 访问配置任务

要在 VPC 中指定子网和安全组，请使用适用 API 的 `VpcConfig` 请求参数，或者在 Amazon Comprehend 控制台中创建任务时提供此信息。Amazon Comprehend 使用此信息创建 ENI 并将其附加到任务容器。ENI 在未连接到互联网的 VPC 中为任务容器提供网络连接。

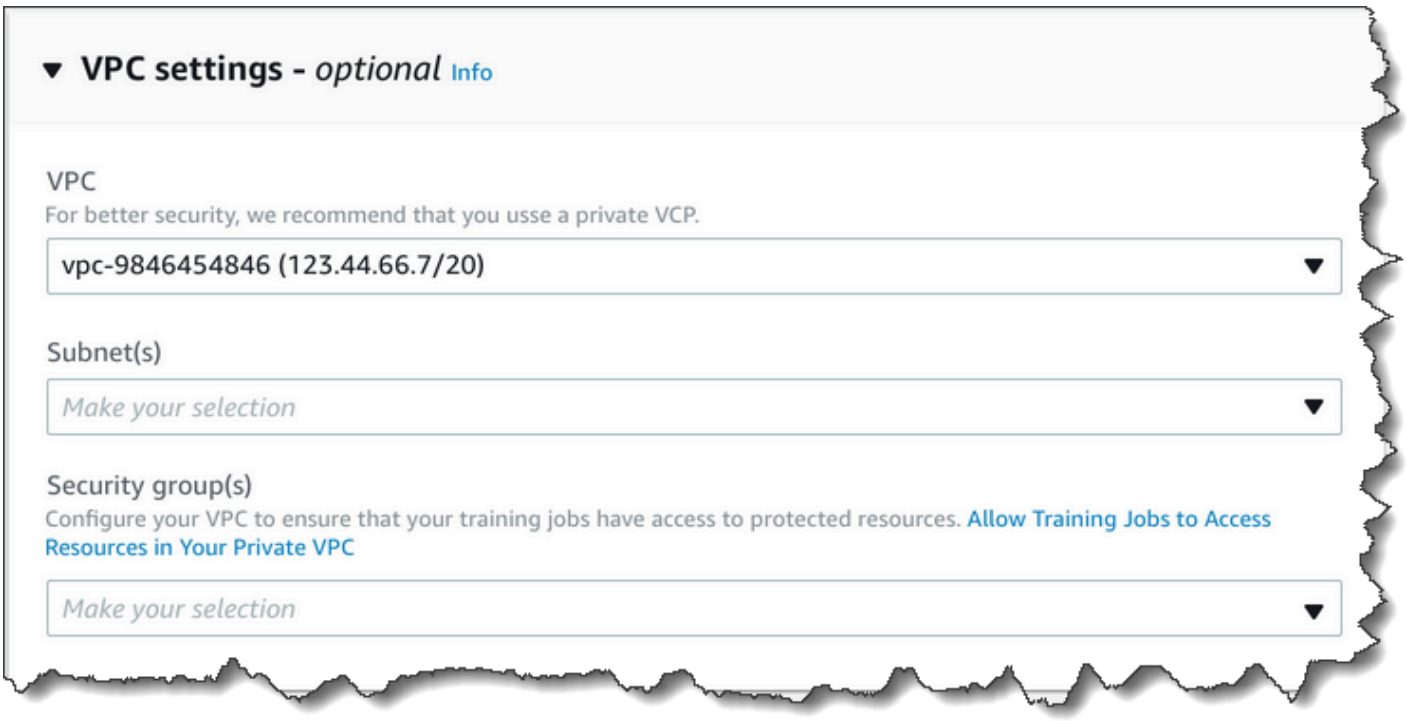
以下 API 包含 `VpcConfig` 请求参数：

- `Create*` API： CreateDocumentClassifier、 CreateEntityRecognizer
- `Start*` API： StartDocumentClassificationJob、StartDominantLanguageDetectionJob、 StartEntitiesDetectionJob、StartKeyPhrasesDetectionJob、 StartSentimentDetectionJob、StartTargetedSentimentDetectionJob、 StartTopicsDetectionJob

以下是您在 API 调用中包含的 VpcConfig 参数的示例：

```
"VpcConfig": {
      "SecurityGroupIds": [
          " sg-0123456789abcdef0"
          ],
      "Subnets": [
          "subnet-0123456789abcdef0",
          "subnet-0123456789abcdef1",
          "subnet-0123456789abcdef2"
          ]
      }
```

要从 Amazon Comprehend 控制台配置 VPC，请在创建任务时从可选的 VPC 设置部分中选择配置详细信息。



为 Amazon Comprehend 任务配置您的 VPC

为 Amazon Comprehend 任务配置 VPC 时，请使用以下指南。有关设置 VPC 的信息，请参阅《Amazon VPC 用户指南》中的使用 VPC 和子网。

确保子网拥有足够的 IP 地址

对于任务中的每个实例，您的 VPC 子网应至少具有两个私有 IP 地址。有关更多信息，请参阅《Amazon VPC 用户指南》中的针对 IPv4 的 VPC 和子网大小调整。

创建 Amazon S3 VPC 终端节点

如果您将 VPC 配置为使任务容器不具有互联网访问权限，则这些容器无法连接到包含数据的 Amazon S3 存储桶，除非您创建一个允许访问的 VPC 终端节点。通过创建 VPC 终端节点，您可以允许任务容器在训练和分析任务期间访问您的数据。

创建 VPC 终端节点时，请配置以下值：

- 选择服务类别作为AWS 服务
- 将服务指定为 `com.amazonaws.region.s3`
- 选择 “网关” 作为 VPC 终端节点类型

如果您使用 AWS CloudFormation 创建 VPC 终端节点，请按照 AWS CloudFormation vpcendPoint 文档进行操作。以下示例显示了模板中的 vpcendPoint 配置。 AWS CloudFormation

```
VpcEndpoint:
  Type: AWS::EC2::VPCEndpoint
  Properties:
    PolicyDocument:
      Version: '2012-10-17'
      Statement:
        - Action:
            - s3:GetObject
            - s3:PutObject
            - s3:ListBucket
            - s3:GetBucketLocation
            - s3:DeleteObject
            - s3:ListMultipartUploadParts
            - s3:AbortMultipartUpload
          Effect: Allow
          Resource:
            - "*"
          Principal: "*"
    RouteTableIds:
      - Ref: RouteTable
    ServiceName:
      Fn::Join:
        - ''
        - - com.amazonaws.
          - Ref: AWS::Region
          - ".s3"
    VpcId:
      Ref: VPC
```

我们还建议您创建自定义策略，只允许来自您的 VPC 的请求访问您的 S3 存储桶。有关更多信息，请参阅《Amazon VPC 用户指南》中的 Amazon S3 终端节点。

以下策略允许访问 S3 存储桶。编辑此策略，以便仅允许访问任务所需的资源。

```
{
    "Version": "2008-10-17",
    "Statement": [
        {
```

```
            "Effect": "Allow",
            "Principal": "*",
            "Action": [
                "s3:GetObject",
                "s3:PutObject",
                "s3:ListBucket",
                "s3:GetBucketLocation",
                "s3:DeleteObject",
                "s3:ListMultipartUploadParts",
                "s3:AbortMultipartUpload"
            ],
            "Resource": "*"
        }
    ]
}
```

对终端节点路由表使用默认的 DNS 设置，以便解析标准 Amazon S3 URL（例如 `http://s3-aws-region.amazonaws.com/DOC-EXAMPLE-BUCKET`）。如果您不使用默认 DNS 设置，则确保通过配置终端节点路由表来解析用于指定任务中数据位置的 URL。有关 VPC 终端节点路由表的信息，请参阅《Amazon VPC 用户指南》中的网关终端节点路由。

默认终端节点策略允许用户在任务容器中安装来自 Amazon Linux 和 Amazon Linux 2 存储库的包。如果您不希望用户安装来自该存储库的包，则创建一个自定义终端节点策略，明确拒绝访问 Amazon Linux 和 Amazon Linux 2 存储库。Comprehend 本身不需要任何此类软件包，因此不会对功能产生任何影响。以下是拒绝访问这些存储库的策略示例：

```
{
    "Statement": [
      {
        "Sid": "AmazonLinuxAMIRepositoryAccess",
        "Principal": "*",
        "Action": [
            "s3:GetObject"
        ],
        "Effect": "Deny",
        "Resource": [
            "arn:aws:s3:::packages.*.amazonaws.com/*",
            "arn:aws:s3:::repo.*.amazonaws.com/*"
        ]
      }
    ]
}
```

```
{
    "Statement": [
        { "Sid": "AmazonLinux2AMIRepositoryAccess",
          "Principal": "*",
          "Action": [
              "s3:GetObject"
              ],
          "Effect": "Deny",
          "Resource": [
              "arn:aws:s3:::amazonlinux.*.amazonaws.com/*"
              ]
         }
    ]
}
```

DataAccessRole 的权限

当您在分析任务中使用 VPC 时，DataAccessRole 用于 Create* 和 Start* 操作的用户必须拥有访问输入文档和输出存储桶的 VPC 的权限。

以下策略提供了用于 Create* 和 Start* 操作的 DataAccessRole 所需的访问权限。

```
{
    "Version": "2008-10-17",
    "Statement": [
        {
            "Effect": "Allow",
              "Action": [
                "ec2:CreateNetworkInterface",
                "ec2:CreateNetworkInterfacePermission",
                "ec2:DeleteNetworkInterface",
                "ec2:DeleteNetworkInterfacePermission",
                "ec2:DescribeNetworkInterfaces",
                "ec2:DescribeVpcs",
                "ec2:DescribeDhcpOptions",
                "ec2:DescribeSubnets",
                "ec2:DescribeSecurityGroups"
            ],
            "Resource": "*"
        }
    ]
}
```

配置 VPC 安全组

在分布式任务中，您必须允许同一任务中的不同任务容器之间进行通信。为此，请为您的安全组配置规则，以允许同一安全组的成员之间实现入站连接。有关信息，请参阅《Amazon VPC 用户指南》中的安全组规则。

连接到 VPC 之外的资源

如果您将 VPC 配置为不具有互联网访问权限，则使用该 VPC 的任务无权访问 VPC 之外的资源。如果您的任务需要访问您的 VPC 外部的资源，请使用以下选项之一以提供访问权限：

- 如果您的任务需要访问支持接口 VPC 终端节点的 AWS 服务，请创建一个终端节点来连接到该服务。有关支持接口端点的服务的列表，请参阅《Amazon VPC 用户指南》中的 VPC 端点。有关创建接口 VPC 终端节点的信息，请参阅 Amazon VPC 用户指南中的接口 VPC 终端节点 (AWS PrivateLink)。
- 如果您的任务需要访问不支持接口 VPC 终端节点的 AWS 服务或外部的资源 AWS，请创建 NAT 网关并将您的安全组配置为允许出站连接。有关为 VPC 设置 NAT 网关的信息，请参阅《Amazon VPC 用户指南》中的场景 2：带有公有子网和私有子网 (NAT) 的 VPC。

Amazon Comprehend 和接口 VPC 终端节点 (AWS PrivateLink)

您可以通过创建接口 VPC 终端节点在 VPC 和 Amazon Comprehend 之间建立私有连接。接口终端节点由 AWS PrivateLink 提供支持，该技术支持您通过私密方式访问 Amazon Comprehend API，而无需互联网网关、NAT 设备、VPN 连接或 AWS Direct Connect 连接。VPC 中的实例即使没有公有 IP 地址也可与 Amazon Comprehend API 进行通信。VPC 和 Amazon Comprehend 之间的流量不会脱离 Amazon 网络。

每个接口终端节点均由子网中的一个或多个弹性网络接口表示。

有关更多信息，请参阅《Amazon VPC 用户指南》中的接口 VPC 端点 (AWS PrivateLink)。

Amazon Comprehend VPC 终端节点的注意事项

在为 Amazon Comprehend 设置接口 VPC 终端节点之前，请务必查看《Amazon VPC 用户指南》中的接口终端节点属性和限制。

Amazon Comprehend 终端节点并非在一个地区的所有可用区域中都可用。在创建终端节点时，使用以下命令可列出可用区域。

```
aws ec2 describe-vpc-endpoint-services \
```

```
  --service-names com.amazonaws.us-west-2.comprehend
```

Amazon Comprehend 支持从 VPC 调用它的所有 API 操作。

为 Amazon Comprehend 创建接口 VPC 终端节点

您可以使用 Amazon VPC 控制台或 AWS Command Line Interface (AWS CLI) 为 Amazon Comprehend 服务创建 VPC 终端节点。有关更多信息，请参阅《Amazon VPC 用户指南》中的创建接口终端节点。

使用以下服务名称为 Amazon Comprehend 创建 VPC 终端节点：

- com.amazonaws.*region*.comprehend

如果为终端节点启用私有 DNS，则可以使用其默认 DNS 名称作为区域，向 Amazon Comprehend 发送 API 请求，例如 *comprehend.us-east-1.amazonaws.com*。

有关更多信息，请参阅《Amazon VPC 用户指南》中的通过接口终端节点访问服务。

为 Amazon Comprehend 创建 VPC 终端节点策略

您可以为 VPC 终端节点附加控制对 Amazon Comprehend 的访问的终端节点策略。该策略指定以下信息：

- 可执行操作的主体。
- 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《Amazon VPC 用户指南》中的使用 VPC 终端节点控制对服务的访问。

示例：Amazon Comprehend 操作的 VPC 终端节点策略

下面是用于 Amazon Comprehend 的终端节点策略示例。当附加到终端节点时，此策略会向所有资源上的所有主体授予对 Amazon Comprehend `DetectEntities` 操作的访问权限。

```
{
   "Statement":[
      {
         "Principal":"*",
         "Effect":"Allow",
```

```
            "Action":[
               "comprehend:DetectEntities"
            ],
            "Resource":"*"
         }
      ]
}
```

适用于 Amazon Comprehend 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 管理员控制谁可以通过身份验证（登录）和授权（具有权限）来使用 Amazon Comprehend 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 Comprehend 如何与 IAM 配合使用
- 适用于 Amazon Comprehend 的基于身份的策略示例
- Amazon Comprehend 的 AWS 托管策略
- Amazon Comprehend 身份和访问问题排查

受众

您的使用方式 AWS Identity and Access Management (IAM) 会有所不同，具体取决于您在 Amazon Comprehend 中所做的工作。

服务用户：如果您使用 Amazon Comprehend 服务来完成任务，则您的管理员会为您提供所需的凭证和权限。随着您使用更多 Amazon Comprehend 特征来完成任务，您可能需要额外权限。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 Amazon Comprehend 中的特征，请参阅 Amazon Comprehend 身份和访问问题排查。

服务管理员：如果您在公司负责管理 Amazon Comprehend 资源，则您可能具有 Amazon Comprehend 的完全访问权限。您有责任确定您的服务用户应访问哪些 Amazon Comprehend 特征和资源。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将 IAM 与 Amazon Comprehend 结合使用的更多信息，请参阅 Amazon Comprehend 如何与 IAM 配合使用。

IAM 管理员：如果您是 IAM 管理员，您可能希望了解有关如何编写策略以管理对 Amazon Comprehend 的访问权限的详细信息。要查看您可在 IAM 中使用的 Amazon Comprehend 基于身份的策略示例，请参阅 适用于 Amazon Comprehend 的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）。要了解用于跨账户访问的角色和基于资源的策略之间的差别，请参阅《IAM 用户指南》中的 IAM 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链接到的服务角色。 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 Comprehend 如何与 IAM 配合使用

在使用 IAM 管理对 Amazon Comprehend 的访问权限之前，您应该了解哪些 IAM 特征可用于 Amazon Comprehend。

可与 Amazon Comprehend 一起使用的 IAM 特征

IAM 功能	Amazon Comprehend 支持
基于身份的策略	是
基于资源的策略	是

IAM 功能	Amazon Comprehend 支持
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	是
服务相关角色	否

要全面了解 Amazon Compreh AWS end 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。

Amazon Comprehend 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。

适用于 Amazon Comprehend 的基于身份的策略示例

要查看 Amazon Comprehend 基于身份的策略示例，请参阅 适用于 Amazon Comprehend 的基于身份的策略示例。

Amazon Comprehend 内基于资源的策略

支持基于资源的策略	是

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。

Amazon Comprehend 服务仅支持一种类型的基于资源的策略（自定义模型策略），该策略将附加到自定义模型上。此策略定义了其他可以使用自定义模型的账户。

要了解如何将基于资源的策略附加到自定义模型上，请参阅 自定义模型的基于资源的策略。

Amazon Comprehend 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 Amazon Comprehend 操作的列表，请参阅服务授权参考中的 Amazon Comprehend 定义的操作。

Amazon Comprehend 中的策略操作在操作前使用以下前缀：

```
comprehend
```

要在单个语句中指定多项操作，请使用逗号将它们隔开。

```
"Action": [
      "comprehend:DetectSentiment",
      "comprehend:ClassifyDocument"
         ]
```

您也可以使用通配符（*）指定多个操作。例如，要指定以单词 Describe 开头的所有操作，包括以下操作：

```
"Action": "comprehend:Describe*"
```

请勿使用通配符来指定一个服务的所有操作。使用以下最佳实践：在策略中指定权限时，授予最低权限。

要查看 Amazon Comprehend 基于身份的策略示例，请参阅 适用于 Amazon Comprehend 的基于身份的策略示例。

Amazon Comprehend 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": "*"
```

有关 Amazon Comprehend 资源类型及其 ARN 的列表，请参阅服务授权参考中的 Amazon Comprehend 定义的资源。要了解您可以使用哪些操作指定每个资源的 ARN，请参阅 Amazon Comprehend 定义的操作。

Amazon Comprehend 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键。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 Amazon Comprehend 条件键的列表，请参阅服务授权参考中的 Amazon Comprehend 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 Comprehend 定义的操作。

要查看 Amazon Comprehend 基于身份的策略示例，请参阅 适用于 Amazon Comprehend 的基于身份的策略示例。

Amazon Comprehend 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。

ABAC 与 Amazon Comprehend

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）。

有关标记 Amazon Comprehend 资源的更多信息，请参阅 标记您的资源。

将临时凭证与 Amazon Comprehend 结合使用

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。

Amazon Comprehend 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。

Amazon Comprehend 的服务角色

支持服务角色	是

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 Amazon Comprehend 的功能。仅当 Amazon Comprehend 提供相关指导时才能编辑服务角色。

要使用 Amazon Comprehend 异步操作，您必须授予 Amazon Comprehend 访问包含文档集合的 Amazon S3 存储桶权限。为此，您可以在账户中创建一个数据访问角色，并使用信任策略来信任 Amazon Comprehend 服务的主体。

有关策略示例，请参阅 异步操作所需的基于角色的权限

Amazon Comprehend 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链接到的服务角色。 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。

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。

适用于 Amazon Comprehend 的基于身份的策略示例

原定设置情况下，用户和角色没有创建或修改 Amazon Comprehend 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 策略。然后，管理员可以向角色添加 IAM 策略，并且用户可以代入角色。

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 policy。

有关 Amazon Comprehend 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 Comprehend 的操作、资源和条件键。

主题

策略最佳实践

基于身份的策略确定某个人是否可以创建、访问或删除您账户中的 Amazon Comprehend 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，请遵循以下准则和建议：

- AWS 托管策略及转向最低权限许可入门 – 要开始向用户和工作负载授予权限，请使用 AWS 托管策略来为许多常见使用场景授予权限。您可以在 AWS 账户中找到这些策略。建议通过定义特定于您的使用场景的 AWS 客户管理型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 策略设置权限时，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 策略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。如果通过特定 AWS 服务（例如 AWS CloudFormation）使用服务操作，您还可以使用条件来授予对服务操作的访问权限。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 策略，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 策略语言 (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，有助于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 Access Analyzer 策略验证。
- 需要多重身份验证 (MFA) – 如果您所处的场景要求您的 AWS 账户 中有 IAM 用户或根用户，请启用 MFA 来提高安全性。要在调用 API 操作时要求 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 中的最佳实践的更多信息，请参阅《IAM 用户指南》中的 IAM 中的安全最佳实践。

使用 Amazon Comprehend 控制台

要访问 Amazon Comprehend 控制台，您必须具有一组最低的权限。这些权限必须允许您列出和查看有关您的 AWS 账户 中 Amazon Comprehend 资源的详细信息。如果创建比必需的最低权限更为严格的基于身份的策略，对于附加了该策略的实体（用户或角色），控制台将无法按预期正常运行。

对于只需要调用 AWS CLI 或 AWS API 的用户，您无需为其提供最低控制台权限。相反，只允许访问与其尝试执行的 API 操作相匹配的操作。

对于最低的 Amazon Comprehend 控制台权限，您可以 *ComprehendReadOnly* AWS 将托管策略附加到实体。有关更多信息，请参阅《IAM 用户指南》中的为用户添加权限。

要使用 Amazon Comprehend 控制台，您需要为如下策略中所示的操作授予权限：

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

```
  {
      "Action": [
          "iam:ListRoles",
          "iam:GetRole",
          "s3:ListAllMyBuckets",
          "s3:ListBucket",
          "s3:GetBucketLocation"
      ],
      "Effect": "Allow",
      "Resource": "*"
  }
  ]
  }
```

Amazon Comprehend 控制台出于以下原因需要上述其他权限：

- `iam` 列出您的账户可用 IAM 角色的权限。
- `s3` 访问包含主题建模数据的 Amazon S3 存储桶和对象的权限。

当您使用控制台创建异步批处理作业或主题建模作业时，您可以选择让控制台为您的作业创建 IAM 角色。要创建 IAM 角色，必须向用户授予以下附加权限，才能创建 IAM 角色和策略以及将策略附加到角色：

```
{
  "Version": "2012-10-17",
  "Statement":
    [
      {
        "Action":
          [
          "iam:CreateRole",
          "iam:CreatePolicy",
          "iam:AttachRolePolicy"
          ],
        "Effect": "Allow",
        "Resource": "*"
      },
      {
        "Action":
          [
          "iam:PassRole"
          ],
```

```
        "Effect": "Allow",
        "Resource": "arn:aws:iam::*:role/*Comprehend*"
      }
    ]
  }
```

Amazon Comprehend 控制台出于以下原因需要上述其他权限：

- iam 创建角色和策略以及附加角色和策略的权限。iam:PassRole 操作使控制台能够将角色传递给 Amazon Comprehend。

允许用户查看他们自己的权限

该示例说明了您如何创建策略，以允许 IAM 用户查看附加到其用户身份的内联策略和托管式策略。此策略包括在控制台上完成此操作或者以编程方式使用 AWS CLI 或 AWS API 所需的权限。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "ViewOwnUserInfo",
            "Effect": "Allow",
            "Action": [
                "iam:GetUserPolicy",
                "iam:ListGroupsForUser",
                "iam:ListAttachedUserPolicies",
                "iam:ListUserPolicies",
                "iam:GetUser"
            ],
            "Resource": ["arn:aws:iam::*:user/${aws:username}"]
        },
        {
            "Sid": "NavigateInConsole",
            "Effect": "Allow",
            "Action": [
                "iam:GetGroupPolicy",
                "iam:GetPolicyVersion",
                "iam:GetPolicy",
                "iam:ListAttachedGroupPolicies",
                "iam:ListGroupPolicies",
                "iam:ListPolicyVersions",
                "iam:ListPolicies",
```

```
                "iam:ListUsers"
            ],
            "Resource": "*"
        }
    ]
}
```

执行文档分析操作所需的权

以下示例策略授予使用 Amazon Comprehend 文档分析操作的权限：

```
{
 "Version": "2012-10-17",
 "Statement": [{
    "Sid": "AllowDetectActions",
    "Effect": "Allow",
    "Action": [
              "comprehend:DetectEntities",
              "comprehend:DetectKeyPhrases",
              "comprehend:DetectDominantLanguage",
              "comprehend:DetectSentiment",
              "comprehend:DetectTargetedSentiment",
              "comprehend:DetectSyntax",
              "textract:DetectDocumentText",
              "textract:AnalyzeDocument"
           ],
    "Resource": "*"
    }
 ]
}
```

该策略中的一个语句授予使用 `DetectEntities`、`DetectKeyPhrases`、`DetectDominantLanguage`、`DetectTargetedSentiment` 和 `DetectSyntax` 操作的权限。该策略语句还授予使用两种 Amazon Textract API 方法的权限。Amazon Comprehend 调用这些方法从图像文件和扫描的 PDF 文档中提取文本。对于从未对这些类型的输入文件运行自定义推理的用户，您可以移除这些权限。

拥有此政策的用户将无法在您的账户中执行批处理操作或异步操作。

该策略不指定 `Principal` 元素，因为在基于身份的策略中，未指定获取权限的委托人。附加了策略的用户是隐式主体。向 IAM 角色附加权限策略后，该角色的信任策略中标识的主体将获取权限。

有关显示所有 Amazon Comprehend API 操作及它们适用的资源的表，请参阅服务授权参考中的 Amazon Comprehend 的操作、资源和条件键。

使用 KMS 加密所需的权限

要在异步作业中充分使用 Amazon 密钥管理服务 (KMS) 进行数据和作业加密，您需要授予以下策略中显示的操作权限：

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
        "Action": [
            "kms:CreateGrant"
        ],
        "Effect": "Allow",
        "Resource": "*"
    },
    {
        "Action": [
            "kms:Decrypt",
            "kms:GenerateDatakey"
        ],
        "Effect": "Allow",
        "Resource": "*",
        "Condition": {
            "StringEquals": {
                "kms:ViaService": [
                    "s3.region.amazonaws.com"
                ]
            }
        }
    }
  ]
}
```

当您使用 Amazon Comprehend 创建异步作业时，您将使用存储在 Amazon S3 上的输入数据。使用 S3，您可以选择加密存储的数据，这些数据由 S3 加密，而不是 Amazon Comprehend 加密的。如果您向 Amazon Comprehend 作业使用的数据访问角色提供对原始输入数据进行加密的密钥的 `kms:Decrypt` 权限，我们就可以解密和读取加密的输入数据。

您还可以选择使用 KMS 客户自主管理型密钥 (CMK) 对 S3 上的输出结果以及作业处理期间使用的存储卷进行加密。执行此操作时，您可以对两种类型的加密使用相同的 KMS 密钥，但这不是必需的。创建作业时，可以使用单独的字段来指定输出加密和卷加密的密钥，您甚至可以使用来自不同账户的 KMS 密钥。

使用 KMS 加密时，卷加密需要 `kms:CreateGrant` 权限，输出数据加密需要 `kms:GenerateDataKey` 权限。要读取加密的输入（如输入数据已由 Amazon S3 加密），需要 `kms:Decrypt` 权限。IAM 角色需要根据需要授予这些权限。但是，如果密钥来自与当前使用的账户不同的账户，则该 KMS 密钥的 KMS 密钥策略还必须将这些权限授予作业的数据访问角色。

适用于 Amazon Comprehend 的 AWS 托管（预定义）策略

AWS 通过提供由 AWS 创建和管理的独立 IAM policy 来满足许多常用案例的要求。这些 AWS 托管策略可针对常用案例授予必要的权限，使您免去调查所需权限的工作。有关更多信息，请参阅《IAM 用户指南》中的 AWS 托管式策略。

以下 AWS 托管策略（可附加到您账户中的用户）特定于 Amazon Comprehend：

- ComprehendFullAccess— 授予对 Amazon Comprehend 资源的完全访问权限，包括运行主题建模作业。包括列出和获取 IAM 角色的权限。
- ComprehendReadOnly— 授予运行所有 Amazon Comprehend `StartDominantLanguageDetectionJob` 操作的权限`StartKeyPhrasesDetectionJob`，`StartEntitiesDetectionJob`但、、、、`StartSentimentD`除外。`StartTargetedSentimentDetectionJob` `StartTopicsDetectionJob`

您需要对任何将使用 Amazon Comprehend 的任何用户应用以下附加策略：

```
{
    "Version": "2012-10-17",
    "Statement":
      [
        {
          "Action":
            [
              "iam:PassRole"
            ],
          "Effect": "Allow",
          "Resource": "arn:aws:iam::*:role/*Comprehend*"
        }
      ]
```

```
  }
```

您可以通过登录到 IAM 控制台并在该控制台中搜索特定策略来查看托管权限策略。

当您使用 AWS SDK 或 AWS CLI 时，这些策略会起作用。

此外，您还可以创建您自己的自定义 IAM policy，以授予 Amazon Comprehend 操作和资源的相关权限。您可以将这些自定义策略附加到需要这些权限的用户、组或角色。

异步操作所需的基于角色的权限

要使用 Amazon Comprehend 异步操作，您必须授予 Amazon Comprehend 访问包含文档集合的 Amazon S3 存储桶权限。为此，您可以在账户中创建一个数据访问角色，并使用信任策略来信任 Amazon Comprehend 服务的主体。有关创建角色的更多信息，请参阅《AWS 身份和访问管理用户指南》中的创建角色以向 AWS 服务委托权限。

以下显示了您创建的角色的信任策略示例。为了帮助避免混淆代理，您可以使用一个或多个全局条件上下文键来限制权限的范围。将 `aws:SourceAccount` 值设置为您的账户 ID。如果您使用 `ArnEquals` 条件，请将 `aws:SourceArn` 值设置为作业的 ARN。为 ARN 中的作业编号使用通配符，因为 Amazon Comprehend 会在创建作业时生成此编号。

```
{
    "Version": "2012-10-17",
    "Statement": [
      {
        "Effect": "Allow",
        "Principal": {
          "Service": "comprehend.amazonaws.com"
        },
        "Action": "sts:AssumeRole",
        "Condition": {
          "StringEquals": {
            "aws:SourceAccount": "111122223333"
          },
          "ArnEquals": {
            "aws:SourceArn": "arn:aws:comprehend:us-west-2:111122223333:pii-entities-
detection-job/*"
          }
        }
      }
    ]
 }
```

创建角色后，为该角色创建访问策略。这应该授予 Amazon S3 `GetObject` 权限和对包含您输入数据的 Amazon S3 存储桶的 `ListBucket` 权限，以及 Amazon S3 对您的 Amazon S3 输出数据存储桶的 `PutObject` 权限。

允许所有 Amazon Comprehend 操作的权限

在您注册 AWS 后，您可以创建管理员用户来管理您的帐户，包括创建用户和管理用户权限。

您可以选择创建具有所有 Amazon Comprehend 操作权限的用户（将该用户视为特定于服务的管理员），以便使用 Amazon Comprehend。您可以将以下权限策略附加到该用户。

```
{
  "Version": "2012-10-17",
  "Statement":
    [
      {
        "Sid": "AllowAllComprehendActions",
        "Effect": "Allow",
        "Action":
        [
            "comprehend:*",
            "iam:ListRoles",
            "iam:GetRole",
            "s3:ListAllMyBuckets",
            "s3:ListBucket",
            "s3:GetBucketLocation",
            "iam:CreateRole",
            "iam:CreatePolicy",
            "iam:AttachRolePolicy",
            "kms:CreateGrant",
            "kms:Decrypt",
            "kms:GenerateDatakey"
      ],
      "Resource": "*"
    },
    {
      "Action":
        [
          "iam:PassRole"
        ],
      "Effect": "Allow",
      "Resource": "arn:aws:iam::*:role/*Comprehend*"
    }
```

```
  ]
}
```

可通过以下方式对这些加密权限进行修改：

- 要使 Amazon Comprehend 能够分析存储在加密 S3 存储桶中的文档，IAM 角色必须拥有 `kms:Decrypt` 权限。
- 要使 Amazon Comprehend 能够加密存储在连接至处理分析任务的计算实例的存储卷上的文档，IAM 角色必须拥有 `kms:CreateGrant` 权限。
- 要使 Amazon Comprehend 能够加密其 S3 存储桶中的输出结果，IAM 角色必须拥有 `kms:GenerateDataKey` 权限。

允许主题建模操作的权限

以下权限策略向用户授予执行 Amazon Comprehend 主题建模操作的权限。

```
{
  "Version": "2012-10-17",
  "Statement": [{
  "Sid": "AllowTopicModelingActions",
  "Effect": "Allow",
  "Action": [
            "comprehend:DescribeTopicsDetectionJob",
            "comprehend:ListTopicsDetectionJobs",
            "comprehend:StartTopicsDetectionJob",
         ],
        "Resource": "*"
        ]
    }
  ]
  }
```

自定义异步分析任务所需的权限

> **Important**
>
> 如果您有限制模型访问权限的 IAM policy，则无法使用自定义模型完成推理作业。您的 IAM policy 应更新为拥有用于自定义异步分析任务的通配符资源。

如果您使用的是StartDocumentClassificationJob 和 StartEntitiesDetectionJobAPI，则需要更新您的 IAM 策略，除非您当前使用通配符作为资源。如果您使用的是使用预训练模型，这不会对您产生影响，您也无需进行任何更改。StartEntitiesDetectionJob

以下示例策略包含一个过时的参考。

```
{
    "Action": [
        "comprehend:StartDocumentClassificationJob",
        "comprehend:StartEntitiesDetectionJob",
    ],
    "Resource": [
        "arn:aws:comprehend:us-east-1:123456789012:document-classifier/myClassifier",
        "arn:aws:comprehend:us-east-1:123456789012:entity-recognizer/myRecognizer"
    ],
    "Effect": "Allow"
}
```

这是成功运行 StartDocumentClassificationJob 和需要使用的更新策略。 StartEntitiesDetectionJob

```
{
    "Action": [
        "comprehend:StartDocumentClassificationJob",
        "comprehend:StartEntitiesDetectionJob",
    ],
    "Resource": [
        "arn:aws:comprehend:us-east-1:123456789012:document-classifier/myClassifier",
        "arn:aws:comprehend:us-east-1:123456789012:document-classification-job/*",
        "arn:aws:comprehend:us-east-1:123456789012:entity-recognizer/myRecognizer",
        "arn:aws:comprehend:us-east-1:123456789012:entities-detection-job/*"
    ],
    "Effect": "Allow"
}
```

Amazon Comprehend 的 AWS 托管策略

要向用户、组和角色添加权限，与自己编写策略相比，使用 AWS 托管策略更简单。创建仅为团队提供所需权限的 IAM 客户管理型策略需要时间和专业知识。要快速入门，您可以使用我们的 AWS 托管策略。这些策略涵盖常见使用案例，可在您的 AWS 账户 中使用。有关 AWS 托管策略的更多信息，请参阅 《IAM 用户指南》中的AWS 托管策略。

AWS 服务负责维护和更新 AWS 托管式策略。您无法更改 AWS 托管式策略中的权限。服务偶尔会向 AWS 托管策略添加额外权限以支持新特征。此类更新会影响附加策略的所有身份（用户、组和角色）。当启动新特征或新操作可用时，服务最有可能会更新 AWS 托管策略。服务不会从 AWS 托管策略中删除权限，因此策略更新不会破坏您的现有权限。

此外，AWS 还支持跨多种服务的工作职能的托管式策略。例如，ReadOnlyAccess AWS 托管式策略提供对所有 AWS 服务和资源的只读访问权限。当服务启动新特征时，AWS 会为新操作和资源添加只读权限。有关工作职能策略的列表和说明，请参阅《IAM 用户指南》中的适用于工作职能的AWS托管策略。

AWS 托管策略： ComprehendFullAccess

该策略授予对 Amazon Comprehend 资源的完全访问权限，包括正在运行的主题建模作业。该策略还授予 Amazon S3 存储桶和 IAM 角色的列出和获取权限。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "comprehend:*",
                "iam:GetRole",
                "iam:ListRoles",
                "s3:GetBucketLocation",
                "s3:ListAllMyBuckets",
                "s3:ListBucket",
            ],
            "Resource": "*"
        }
    ]
}
```

AWS 托管式策略： ComprehendReadOnly

该策略授予运行所有 Amazon Comprehend 操作的只读权限，以下操作除外：

- `StartDominantLanguageDetectionJob`

- StartEntitiesDetectionJob
- StartKeyPhrasesDetectionJob
- StartSentimentDetectionJob
- StartTargetedSentimentDetectionJob
- StartTopicsDetectionJob

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Action": [
                "comprehend:BatchDetectDominantLanguage",
                "comprehend:BatchDetectEntities",
                "comprehend:BatchDetectKeyPhrases",
                "comprehend:BatchDetectSentiment",
                "comprehend:BatchDetectSyntax",
                "comprehend:ClassifyDocument",
                "comprehend:ContainsPiiEntities",
                "comprehend:DescribeDocumentClassificationJob",
                "comprehend:DescribeDocumentClassifier",
                "comprehend:DescribeDominantLanguageDetectionJob",
                "comprehend:DescribeEndpoint",
                "comprehend:DescribeEntitiesDetectionJob",
                "comprehend:DescribeEntityRecognizer",
                "comprehend:DescribeKeyPhrasesDetectionJob",
                "comprehend:DescribePiiEntitiesDetectionJob",
                "comprehend:DescribeResourcePolicy",
                "comprehend:DescribeSentimentDetectionJob",
                "comprehend:DescribeTargetedSentimentDetectionJob",
                "comprehend:DescribeTopicsDetectionJob",
                "comprehend:DetectDominantLanguage",
                "comprehend:DetectEntities",
                "comprehend:DetectKeyPhrases",
                "comprehend:DetectPiiEntities",
                "comprehend:DetectSentiment",
                "comprehend:DetectSyntax",
                "comprehend:ListDocumentClassificationJobs",
                "comprehend:ListDocumentClassifiers",
                "comprehend:ListDocumentClassifierSummaries",
                "comprehend:ListDominantLanguageDetectionJobs",
                "comprehend:ListEndpoints",
```

```
                "comprehend:ListEntitiesDetectionJobs",
                "comprehend:ListEntityRecognizers",
                "comprehend:ListEntityRecognizerSummaries",
                "comprehend:ListKeyPhrasesDetectionJobs",
                "comprehend:ListPiiEntitiesDetectionJobs",
                "comprehend:ListSentimentDetectionJobs",
                "comprehend:ListTargetedSentimentDetectionJobs",
                "comprehend:ListTagsForResource",
                "comprehend:ListTopicsDetectionJobs"
            ],
            "Effect": "Allow",
            "Resource": "*"
        }
    ]
}
```

Amazon Comprehend 更新了 AWS 托管策略

查看有关自此服务开始跟踪这些更改起，Amazon Comprehend 的 AWS 托管式策略更新的详细信息。有关此页面更改的自动提示，请订阅 Amazon Comprehend 文档历史记录页面上的 RSS 源。

更改	描述	日期
ComprehendReadOnly – 对现有策略的更新	Amazon Comprehend 现在`comprehend:DescribeTargetedSentimentDetectionJob` 允许`comprehend:ListTargetedSentimentDetectionJobs` 在策略中使用和操作 ComprehendReadOnly	2022 年 3 月 30 日
ComprehendReadOnly – 更新了现有策略	Amazon Comprehend 现在`comprehend:DescribeResourcePolicy` 允许在政策中执行该操作 ComprehendReadOnly	2022 年 2 月 2 日

更改	描述	日期
ComprehendReadOnly – 更新了现有策略	Amazon Comprehend 现在`ListDocumentClassifierSummaries` 允许`ListEntityRecognizerSummaries` 在策略中使用和操作 ComprehendReadOnly	2021 年 9 月 21 日
ComprehendReadOnly – 更新了现有策略	Amazon Comprehend 现在`ContainsPIIEntities` 允许在政策中执行该操作 ComprehendReadOnly	2021 年 3 月 26 日
Amazon Comprehend 已开启跟踪变更	Amazon Comprehend 开始跟踪其 AWS 托管策略的更改。	2021 年 3 月 1 日

Amazon Comprehend 身份和访问问题排查

使用以下信息帮助您诊断和修复在使用 Amazon Comprehend 和 IAM 时可能遇到的常见问题。

主题

- 我没有在 Amazon Comprehend 中执行操作的权限
- 我无权执行 iam：PassRole
- 我想允许我以外的人访问我的 Amazon Comprehend 资源 AWS 账户

我没有在 Amazon Comprehend 中执行操作的权限

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 `comprehend:`*`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:
 comprehend:GetWidget on resource: my-example-widget
```

在此情况下，Mateo 的策略必须更新以允许其使用 comprehend:*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，指明您无权执行 iam:PassRole 操作，则必须更新策略以允许您将角色传递给 Amazon Comprehend。

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 Amazon Comprehend 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 Amazon Comprehend 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 Amazon Comprehend 是否支持这些特征，请参阅 Amazon Comprehend 如何与 IAM 配合使用。
- 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。

使用 AWS CloudTrail 记录 Amazon Comprehend API 调用

Amazon Comprehend 与一项服务AWS CloudTrail集成，可记录用户、角色或服务在 Amazon Comprehend 中执行的操作。AWS CloudTrail 将 Amazon Comprehend 的 API 调用捕获为事件。捕获调用中包括通过 Amazon Comprehend 控制台的调用和对 Amazon Comprehend API 操作的代码调用。如果您创建了跟踪，则可以允许将 CloudTrail 事件持续传输到 Amazon S3 存储桶，包括针对 Amazon Comprehend 的事件。如果您未配置跟踪，您仍然可以在 CloudTrail 控制台的 “事件历史记录” 中查看最新的事件。通过收集的信息 CloudTrail，您可以确定向 Amazon Comprehend 发出的请求、发出请求的 IP 地址、谁提出请求、何时提出请求以及其他详细信息。

要了解更多信息 CloudTrail，包括如何配置和启用它，请参阅AWS CloudTrail用户指南。

亚马逊 Comprehend 中的信息 CloudTrail

CloudTrail 在您创建账户AWS 账户时已在您的账户上启用。当 Amazon Comprehend 中出现支持的事件活动时，该活动会与其他服务事件AWS一起记录 CloudTrail 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，请参阅使用事件历史查看 CloudTrail 事件。

要持续记录 AWS 账户 中的事件（包括 Amazon Comprehend 的事件），请创建跟踪。跟踪允许 CloudTrail 将日志文件传输到 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 AWS 区域。此跟踪记录在 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，以进一步分析和处理 CloudTrail 日志中收集的事件数据。有关更多信息，请参阅下列内容：

- 创建跟踪记录概览
- CloudTrail 支持的服务和集成
- 配置 Amazon SNS 通知 CloudTrail
- 接收来自多个地区的 CloudTrail 日志文件和接收来自多个账户的 CloudTrail 日志文件

Amazon Comprehend 支持将以下操作作为事件记录在日志文件中： CloudTrail

- BatchDetectDominantLanguage
- BatchDetectEntities

- BatchDetectKeyPhrases
- BatchDetectSentiment
- BatchDetectSyntax
- ClassifyDocument
- CreateDocumentClassifier
- CreateEndpoint
- CreateEntityRecognizer
- DeleteDocumentClassifier
- DeleteEndpoint
- DeleteEntityRecognizer
- DescribeDocumentClassificationJob
- DescribeDocumentClassifier
- DescribeDominantLanguageDetectionJob
- DescribeEndpoint
- DescribeEntitiesDetectionJob
- DescribeEntityRecognizer
- DescribeKeyPhrasesDetectionJob
- DescribePiiEntitiesDetectionJob
- DescribeSentimentDetectionJob
- DescribeTargetedSentimentDetectionJob
- DescribeTopicsDetectionJob
- DetectDominantLanguage
- DetectEntities
- DetectKeyPhrases
- DetectPiiEntities
- DetectSentiment
- DetectSyntax
- ListDocumentClassificationJobs
- ListDocumentClassifiers

- ListDominantLanguageDetectionJobs
- ListEndpoints
- ListEntitiesDetectionJobs
- ListEntityRecognizers
- ListKeyPhrasesDetectionJobs
- ListPiiEntitiesDetectionJobs
- ListSentimentDetectionJobs
- ListTargetedSentimentDetectionJobs
- ListTagsForResource
- ListTopicsDetectionJobs
- StartDocumentClassificationJob
- StartDominantLanguageDetectionJob
- StartEntitiesDetectionJob
- StartKeyPhrasesDetectionJob
- StartPiiEntitiesDetectionJob
- StartSentimentDetectionJob
- StartTargetedSentimentDetectionJob
- StartTopicsDetectionJob
- StopDominantLanguageDetectionJob
- StopEntitiesDetectionJob
- StopKeyPhrasesDetectionJob
- StopPiiEntitiesDetectionJob
- StopSentimentDetectionJob
- StopTargetedSentimentDetectionJob
- StopTrainingDocumentClassifier
- StopTrainingEntityRecognizer
- TagResource
- UntagResource
- UpdateEndpoint

每个事件或日记账条目都包含有关生成请求的人员信息。身份信息有助于您确定以下内容：

- 请求是否使用根用户凭证进行请求。
- 请求是使用角色还是联合用户的临时安全凭证发出的。
- 请求是否由其他 AWS 服务发出。

有关更多信息，请参阅 CloudTrail userIdentity 元素。

示例：Amazon Comprehend 日志文件条目

跟踪是一种配置，允许将事件作为日志文件传输到您指定的 Amazon S3 存储桶。 CloudTrail 日志文件包含一个或多个日志条目。事件代表来自任何来源的单个请求，包括有关请求的操作、操作的日期和时间、请求参数等的信息。 CloudTrail 日志文件不是公共 API 调用的有序堆栈跟踪，因此它们不会按任何特定的顺序出现。

以下示例显示了演示该`ClassifyDocument`操作的 CloudTrail 日志条目。


```
{
    "eventVersion": "1.08",
    "userIdentity": {
        "type": "IAMUser",
        "principalId": "AROAICFHPEXAMPLE",
        "arn": "arn:aws:iam::12345678910:user/myadmin2",
        "accountId": "12345678910",
        "accessKeyId": "ASIA3VZEXAMPLE",
        "sessionContext": {
            "sessionIssuer": {},
            "webIdFederationData": {},
            "attributes": {
                "creationDate": "2023-10-19T14:22:09Z",
                "mfaAuthenticated": "false"
            }
        }
    },
    "eventTime": "2023-10-19T17:31:20Z",
    "eventSource": "comprehend.amazonaws.com",
    "eventName": "ClassifyDocument",
    "awsRegion": "us-east-2",
    "sourceIPAddress": "3.21.185.237",
    "userAgent": "Mozilla/5.0 (Macintosh; Intel Mac OS X 10.15; rv:109.0)
 Gecko/20100101 Firefox/115.0",
```

```
    "requestParameters": null,
    "responseElements": null,
    "requestID": "fd916e66-caac-46c9-a1fc-81a0ef33e61b",
    "eventID": "535ca22b-b3a3-4c13-b2c5-bf51ab082794",
    "readOnly": false,
    "resources": [
        {
            "accountId": "12345678910",
            "type": "AWS::Comprehend::DocumentClassifierEndpoint",
            "ARN": "arn:aws:comprehend:us-east-2:12345678910:document-classifier-
endpoint/endpointExample"
        }
    ],
    "eventType": "AwsApiCall",
    "recipientAccountId": "12345678910"
}
```

Amazon Comprehend 的合规性验证

作为多个 AWS 合规性计划的一部分，第三方审计员将评估 Amazon Comprehend 的安全性和合规性。其中包括 PCI、FedRAMP、HIPAA 及其他计划。您可以使用 AWS Artifact 下载第三方审计报告。有关更多信息，请参阅下载 AWS Artifact 中的报告。

您使用 Amazon Comprehend 的合规性责任取决于您数据的敏感度、贵公司的合规性目标以及适用的法律法规。AWS 提供以下资源来帮助满足合规性：

- 安全性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 AWS 上部署基于安全性和合规性的基准环境的步骤。
- 设计符合 HIPAA 安全性和合规性要求的架构白皮书 — 此白皮书介绍公司如何使用 AWS 创建符合 HIPAA 标准的应用程序。
- AWS 合规性资源 – 此业务手册和指南集合可能适用于您的行业和位置。
- AWS Config – 此 AWS 服务评估您的资源配置对内部实践、行业指南和法规的遵循情况。
- AWS Security Hub – 此 AWS 服务提供了 AWS 中安全状态的全面视图，可帮助您检查是否符合安全行业标准和最佳实操。

有关特定合规性计划范围内的 AWS 服务的列表，请参阅按合规性计划提供的范围内 AWS 服务。有关一般信息，请参阅 AWS合规性计划。

Amazon Comprehend 中的弹性

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 Comprehend 中的基础设施安全性

作为一项托管式服务，Amazon Comprehend 受 AWS 全球网络安全保护。有关 AWS 安全服务以及 AWS 如何保护基础设施的信息，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，请参阅安全支柱 AWS 架构完善的框架中的基础设施保护。

您可以使用 AWS 发布的 API 调用通过网络访问 Amazon Comprehend。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 主体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。

指南和配额

除非另行指定，否则 Amazon Comprehend 的配额是按区域计算的。如果应用程序需要，您可以请求提高可调配额。有关配额以及如何请求增加配额的更多信息，请参阅 AWS 服务限额。

主题

支持的区域

Amazon Comprehend 在以下地区上市： AWS

- 美国东部（俄亥俄）
- 美国东部（弗吉尼亚州北部）
- US West（Oregon）
- 亚太地区（孟买）
- 亚太地区（首尔）
- 亚太地区（新加坡）
- 亚太地区（悉尼）
- 亚太地区（东京）
- 加拿大（中部）
- 欧洲地区（法兰克福）
- 欧洲地区（爱尔兰）
- 欧洲地区（伦敦）
- AWS GovCloud （美国西部）

默认情况下，Amazon Comprehend 在每个支持的区域提供所有 API 操作。有关例外情况，请参阅 文档处理。

有关 API 终端节点的信息，请参阅《Amazon Web Services 一般参考》中的 Amazon Comprehend 区域和终端节点。

要查看某个区域的当前配额或请求增加可调整配额的配额，请打开服务限额控制台。

内置模型的配额

Amazon Comprehend 提供内置模型供您分析 UTF-8 文本文档。Amazon Comprehend 提供使用内置模型的同步和异步操作。

主题

- 实时（同步）分析
- 异步分析

实时（同步）分析

本节介绍与使用内置模型进行实时分析相关的配额。

主题

- 单一文档操作
- 多个文档操作
- 对实时（同步）请求进行节流

单一文档操作

Amazon Comprehend API 提供将单个文档作为输入的操作。以下配额适用于这些操作。

单个文档操作的常规配额

以下配额适用于用于检测实体、关键短语或主要语言的实时分析。对于实体检测，这些配额适用于使用内置模型进行的检测。有关自定义实体检测，请参阅 自定义实体识别 中的配额。

描述	配额/指南
最大文档大小	100KB

单个文档操作的特定操作配额

以下配额适用于检测情绪、目标情绪和语法的实时分析。

描述	配额/指南
最大文档大小	5 KB

多个文档操作

Amazon Comprehend API 提供批处理操作，只需一个 API 请求即可处理多个文档。以下配额适用于批处理操作。

描述	配额/指南
最大文档大小	5 KB
每次请求的最大文件数	25

有关使用批处理文档操作的更多信息，请参阅 多文档同步处理。

对实时（同步）请求进行节流

Amazon Comprehend 对同步请求应用动态节流。如果系统处理带宽可用，Amazon Comprehend 会逐渐增加它处理的请求数量。为了控制您的应用程序对同步 API 操作的使用，我们建议您在应用程序中开启账单提醒或实施速率限制。

异步分析

本节介绍与使用内置模型进行异步分析相关的配额。

每个异步 API 操作最多支持 10 个活动作业。要查看每个 API 操作的配额，请参阅《Amazon Web Services 一般参考》中 Amazon Comprehend 终端节点和配额中的服务限额表。

对于可调整配额，您可以使用服务限额控制台请求增加配额。

主题

- 异步操作的常规配额
- 异步作业的特定操作配额

- 对异步请求进行节流

异步操作的常规配额

您可以使用控制台或任何 API `Start*` 操作运行异步分析作业。有关何时使用异步操作的信息，请参阅 异步批处理。以下配额适用于内置模型的大多数 API `Start*` 操作。有关例外情况，请参阅 异步作业的特定操作配额。

描述	配额/指南
检测实体、关键短语、PII 和语言的作业中每个文档的最大大小	1 MB
请求中所有文件的最大总大小	5 GB
请求中所有文件的最小总大小	500 字节
最大文件数，每个文件一个文档	1000000
最大总行数，每行一个文档	1000000

异步作业的特定操作配额

本节介绍特定异步操作的配额。如果下表中未指定配额，则适用常规配额值。

主题

- 情绪
- 目标情绪
- 事件
- 主题建模

情绪

您使用该操作创建的异步情绪StartSentimentDetectionJob作业具有以下配额。

描述	配额/指南
每个输入文档的最大大小	5 KB

目标情绪

您通过该操作创建的异步定向情绪StartTargetedSentimentDetectionJob作业具有以下配额。

描述	配额/指南
支持的文档格式	UTF-8
作业中每个文档的最大大小	10 KB
作业中所有文档的最大大小	300 MB
最大文件数，每个文件一个文档	30000
最大总行数，每行一个文档（适用于请求中的所有文件）	30000

事件

您通过该StartEventsDetectionJob操作创建的异步事件检测任务具有以下配额。

描述	配额
字符编码	UTF-8
作业中所有文件的总大小	50 MB
作业中每个文档的最大大小	10 KB
最大文件数，每个文件一个文档	5000
最大总行数，每行一个文档（适用于请求中的所有文件）	5000

主题建模

您使用操作创建的异步主题建模StartTopicsDetectionJob作业具有以下配额。

描述	配额/指南
字符编码	UTF-8

描述	配额/指南
返回主题的最大数量	100
一个文件的最大文件大小，每个文件一个文档	100 MB

有关更多信息，请参阅 主题建模。

对异步请求进行节流

每个异步 API 操作支持每秒的最大请求数（每个区域、每个账户），还支持最多 10 个活动作业。要查看每个 API 操作的配额，请参阅《Amazon Web Services 一般参考》中 Amazon Comprehend 终端节点和配额中的服务限额表。

对于可调整配额，您可以使用服务限额控制台请求增加配额。

自定义模型的配额

您可以使用 Amazon Comprehend 构建自己的自定义模型，用于自定义分类和自定义实体识别。本节提供与训练和使用自定义模型相关的指南和配额。有关自定义模型的更多信息，请参阅 Amazon Comprehend 自定义。

主题

- 常规配额
- 终端节点配额
- 文档分类
- 自定义实体识别

常规配额

Amazon Comprehend 为每种类型的输入文档设置了常规大小配额，您可以使用自定义模型进行分析。有关实时分析配额，请参阅 用于实时分析的最大文档大小。有关异步分析配额，请参阅 异步自定义分析的输入。

每个异步 API 操作支持每秒的最大请求数（每个区域、每个账户），还支持最多 10 个活动作业。要查看每个 API 操作的配额，请参阅《Amazon Web Services 一般参考》中 Amazon Comprehend 终端节点和配额中的服务限额表。

对于可调整配额，您可以使用服务限额控制台请求增加配额。

终端节点配额

您可以创建终端节点以使用自定义模型运行实时分析。有关终端节点的信息，请参阅 管理 Amazon Comprehend 终端节点。

以下配额适用于终端节点。有关请求增加配额的更多信息，请参阅 AWS 服务限额。

描述	配额/指南
每个账户每个区域的活动终端节点的最大数量	20
每个账户每个区域的推理单元的最大数量	200
每个区域每个终端节点推理单元的最大数量	50
每个推理单元的最大吞吐量（字符）	每秒 100 个
每个推理单元的最大吞吐量（文档）	每秒 2 个

文档分类

本节介绍以下文档分类操作的指南和配额：

- 从CreateDocumentClassifier操作开始的分类器训练作业。
- 您从操作开始执行的异步文档分类StartDocumentClassificationJob作业。
- 使用该ClassifyDocument操作的同步文档分类请求。

文件分类的常规配额

下表描述了与训练自定义分类器相关的常规配额。

描述	配额/指南
用户名的最大长度	5000 个字符
类数（多类模型）	2 - 1000

描述	配额/指南
类数（多标签模式）	2 - 100
注释格式	
每类的最小注释数量（多类模式）	10
每类的最小注释数量（多标签模式）	10
最小注释数量（多标签模式）	50
CSV 文件格式	
每类最小训练文档数量（多类模式）	50
每类最小训练文档数量（多标签模式）	10
最小训练文档数量（多标签模式）	50

纯文本文档的分类

您可以使用纯文本输入文档创建和训练纯文本模型。Amazon Comprehend 提供实时和异步操作，使用纯文本模型对纯文本文档进行分类。

训练

下表描述了与使用纯文本文档训练自定义分类器相关的配额。

描述	配额/指南
训练作业中所有文件的总大小	5 GB
用于训练自定义分类器的增强清单文件的最大数量	5
每个增强的清单文件的最大属性名称数量	5
属性名称的最大长度	63 个字符

实时（同步）分析

下表描述了与纯文本文档实时分类相关的配额。

描述	配额/指南
每个同步请求的最大文档数	1
最大文本文档大小（UTF-8 编码）	10 KB

异步分析

下表描述了与纯文本文档异步分类相关的配额。

描述	配额/指南
异步作业中所有文件的总大小	5 GB
一个文件的最大文件大小，每个文件一个文档	10MB
最大文件数，每个文件一个文档	1000000
最大总行数，每行一个文档（适用于请求中的所有文件）	1000000

半结构化文档的分类

本节介绍半结构化文档的文档分类指南和配额。要对半结构化文档进行分类，请使用使用原生输入文档训练过的原生文档模型。

使用半结构化文档训练原生文档模型

下表描述了与使用半结构化文档（例如 PDF 文档、Word 文档和图像文件）训练自定义分类器相关的配额。

描述	配额/指南
所有文档的最大页数	10000

描述	配额/指南
最大注释文件大小（所有 CSV 文件大小总和）	5MB
文档语料库大小（训练和测试文档）	10 GB
训练和测试文件的文件大小	
图像文件大小（JPG、PNG、TIFF）。	1 字节 - 10 MB。 TIFF 文件：最多一页。
PDF 文档的页面大小	1 字节 - 10 MB
Word 文档的页面大小	1 字节 - 10 MB
Amazon Textract API 输出 JSON 大小	1 字节 - 1 MB

实时（同步）分析

本节介绍与半结构化文档的实时分类相关的配额。

下表显示输入文档的最大文件大小。对于所有输入文档类型，输入文件的最大值为一页，不超过 10000 个字符。

文件类型	最大大小 (API)	最大大小（控制台）
UTF-8 文本文档	10 KB	10 KB
PDF 文档	10MB	5MB
Word 文档	10MB	5MB
图像文件	10MB	5MB
Amazon Textract API 输出大小	1 MB	不适用

异步分析

下表描述了与半结构化文档异步分类相关的配额。

描述	配额/指南
作业所有输入文档的最大页数	25000
文档语料库大小	25 GB
图像文件大小（JPG、PNG 或 TIFF）	1 字节 - 10 MB。 TIFF 文件：最多一页。
PDF 文档的页面大小	1 字节 - 10 MB
Word 文档的页面大小	1 字节 - 10 MB
Textract API 输出 JSON 大小	1 字节 - 1 MB。

自定义实体识别

本节介绍自定义实体识别的以下操作的指南和配额：

- 实体识别器训练作业从CreateEntityRecognizer操作开始。
- 异步实体识别作业从该StartEntitiesDetectionJob操作开始。
- 使用DetectEntities操作同步实体识别请求。

纯文本文档的自定义实体识别

Amazon Comprehend 提供异步和同步操作，可使用自定义实体识别器分析纯文本文档。

训练

本节介绍与训练自定义实体识别器分析纯文本文档相关的配额。要训练模型，您可以提供实体列表或一组带注释的文本文档。

下表描述了与使用实体列表训练模型相关的配额。

描述	配额/指南
每个模型的实体数量	1 - 25

描述	配额/指南
文件大小 (UTF-8)	1 - 5000 字节
实体列表中的项目数	1 - 1 百万
条目列表中单个条目（去除格式后）的长度	1 - 5000
实体列表语料库大小（所有文档合并为纯文本）	5 KB - 200 MB

下表描述了与使用注释文本文档训练模型相关的配额。

描述	配额/指南
每个模型/自定义实体识别器的实体数量	1 - 25
文件大小 (UTF-8)	1 - 5000 字节
文档数量（参见纯文本注释）	3 - 200000
文档语料库大小（所有文档合并为纯文本）	5 KB - 200 MB
每个实体的最小注释数量	25

实时（同步）分析

下表描述了与纯文本文档实时分析相关的配额。

描述	配额/指南
每个同步请求的最大文档数	1
最大文本文档大小（UTF-8 编码）	5 KB

异步分析

下表描述了与纯文本文档的异步实体识别相关的配额。

描述	配额/指南
文件大小 (UTF-8)	1 字节 - 1 MB
最大文件数，每个文件一个文档	1000000
最大总行数，每行一个文档（适用于请求中的所有文件）	1000000
文档语料库大小（所有文档合并为纯文本）	1 字节 - 5 GB

半结构化文档的自定义实体识别

Amazon Comprehend 提供异步和同步操作，可使用自定义实体识别器分析半结构后化文档。您必须使用带注释的 PDF 文档训练模型。

训练

下表描述了与训练自定义实体识别器 (CreateEntityRecognizer) 以分析半结构化文档相关的配额。

描述	配额/指南
每个模型/自定义实体识别器的实体数量	1 - 25
最大注释文件大小 (UTF-8 JSON)	5MB
文档数量	250 - 10000
文档语料库大小（所有文档合并为纯文本）	5 KB - 1 GB
每个实体的最小注释数量	100
用于训练自定义实体识别器的增强清单文件的最大数量	5
每个增强的清单文件的最大属性名称数量	5
属性名称的最大长度	63 个字符

实时（同步）分析

本节介绍与半结构化文档实时分析相关的配额。

下表显示输入文档的最大文件大小。对于所有输入文档类型，输入文件的最大值为一页，不超过 10000 个字符。

文件类型	最大大小 (API)	最大大小（控制台）
UTF-8 文本文档	10 KB	10 KB
PDF 文档	10MB	5MB
Word 文档	10MB	5MB
图像文件	10MB	5MB
Textract 输出文件	1 MB	不适用

异步分析

本节介绍半结构化文档异步分析的配额。

描述	配额/指南
图像尺寸（JPG 或 PNG）	1 字节 - 10 MB
图像尺寸 (TIFF)	1 字节 - 10 MB。最多一页。
文档大小 (PDF)	1 字节 - 50 MB
文档大小 (Docx)	1 字节 - 5 MB
文件大小 (UTF-8)	1 字节 - 1 MB
最大文件数量，每个文件一个文档（图像文件或 PDF/Word 文档不允许每行一个文档）	500
PDF 或 Docx 文件的最大页数	100
文本提取后的文档语料库大小（纯文本，所有文件合并）	1 字节 - 5 GB

有关图像限制的更多信息，请参阅 Amazon Textract 中的硬限制

飞轮配额

使用飞轮管理自定义模型版本的训练和跟踪，以进行自定义分类和自定义实体识别。有关飞轮的更多信息，请参阅 飞轮。

飞轮的常规配额

以下配额适用于飞轮和飞轮迭代。

描述	配额/指南
最大飞轮数量	50
处于“创建”状态的飞轮的最大数量	10
每个飞轮训练数据集的最大数量	50
每个飞轮测试数据集的最大数量	50
处于“提取”状态的数据集的最大数量	10
每个账户正在进行的飞轮迭代的最大数量	10

自定义分类模型的数据集配额

当您为与自定义分类模型关联的飞轮摄取数据集时，适用以下配额。

描述	配额/指南
每类最小训练文档数量（多标签模式）	50
最大训练文档数	1000000
最小数据集大小	500 字节
最大数据集大小	5 GB
一个文件的最大文件大小，每个文件一个文档	10MB

自定义实体识别模型的数据集配额

当您为与自定义实体识别模型关联的飞轮提取数据集时，适用以下配额。

描述	配额/指南
最大文档大小	5 KB
最小训练文档数	3
最大训练文档数	200,000
每个实体的最小注释数量	25
最大数据集大小	200 MB

教程和其他资源

Amazon Comprehend 的教程和其他资源。

主题

- 教程：使用 Amazon Comprehend 分析来自客户评论的见解
- 使用 Amazon S3 对象 Lambda 接入点获取个人身份信息 (PII)
- 解决方案：使用 Amazon Comprehend 分析文本和 OpenSearch

教程：使用 Amazon Comprehend 分析来自客户评论的见解

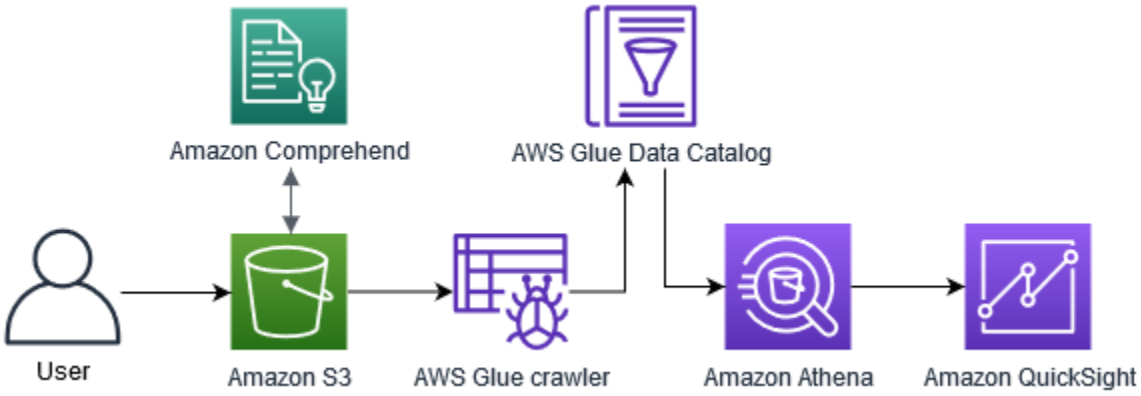
本教程介绍如何将 Amazon Comp rehend 与亚马逊简单AWS Glue存储服务Amazon Athena、QuickSight和亚马逊一起使用，以获得对文档的宝贵见解。Amazon Comprehend 可以从非结构化文本中提取情绪（文档的情绪）和实体（人员、组织、事件、日期、产品、地点、数量和标题）。

例如，您可以从客户评论中获得可操作的见解。在本教程中，您将分析客户对小说的评论样本数据集。您可以使用 Amazon Comprehend 情绪分析来确定客户对这部小说的看法是积极还是消极。您还可以使用 Amazon Comprehend 实体分析来发现提及的重要实体，例如相关小说或作者。完成本教程后，您可能会发现超过 50% 的评论是积极的。您可能还会发现，客户会比较作者并表示对其他经典小说的兴趣。

在本教程中，您完成了以下任务：

- 将评论样本数据集存储在 Amazon Simple Storage Service (Amazon S3) 中。Amazon Simple Storage Service 是一种对象存储服务。
- 使用 Amazon Comprehend 分析评论文档中的情绪和实体。
- 使用 AWS Glue 爬网程序将分析结果存储在数据库中。AWS Glue 是提取、转换和加载 (ETL) 服务，允许您编目和清理数据进行分析。
- 运行 Amazon Athena 查询以清理您的数据。Amazon Athena 是一种无服务器交互式查询服务。
- 在 Amazon QuickSight 中使用您的数据创建可视化效果。Amazon QuickSight 是一款无服务器商业智能工具，用于从您的数据中提取见解。

图表显示了以下工作流程。



完成本教程的预计时间：1 小时

预计费用：本教程中的某些操作会向您的 AWS 账户 收取费用。有关每项服务的费用信息，请参阅以下定价页面。

- Amazon S3 定价
- Amazon Comprehend 定价
- AWS Glue 定价
- Amazon Athena 定价
- 亚马逊 QuickSight定价

主题

先决条件

要完成本教程，您需要：

- AWS 账户。有关设置 AWS 账户 的信息，请参阅 设置。

- IAM 实体（用户、组或角色）。要了解如何为您的账户设置用户和群组，请参阅《IAM 用户指南》中的入门教程。
- 将以下权限策略附加到用户、组或角色。该策略授予完成本教程所需的部分权限。下一个先决条件描述了您需要的其他权限。

```
{
  "Version": "2012-10-17",
  "Statement":
  [
    {
      "Sid": "VisualEditor0",
      "Effect": "Allow",
      "Action":
      [
        "comprehend:*",
        "ds:AuthorizeApplication",
        "ds:CheckAlias",
        "ds:CreateAlias",
        "ds:CreateIdentityPoolDirectory",
        "ds:DeleteDirectory",
        "ds:DescribeDirectories",
        "ds:DescribeTrusts",
        "ds:UnauthorizeApplication",
        "iam:AttachRolePolicy",
        "iam:CreatePolicy",
        "iam:CreatePolicyVersion",
        "iam:CreateRole",
        "iam:DeletePolicyVersion",
        "iam:DeleteRole",
        "iam:DetachRolePolicy",
        "iam:GetPolicy",
        "iam:GetPolicyVersion",
        "iam:GetRole",
        "iam:ListAccountAliases",
        "iam:ListAttachedRolePolicies",
        "iam:ListEntitiesForPolicy",
        "iam:ListPolicies",
        "iam:ListPolicyVersions",
        "iam:ListRoles",
        "quicksight:*",
        "s3:*",
        "tag:GetResources"
      ],
```

```
      "Resource": "*"
    },
    {
      "Action":
      [
        "iam:PassRole"
      ],
      "Effect": "Allow",
      "Resource":
      [
        "arn:aws:iam::*:role/*Comprehend*"
      ]
    }
  ]
}
```

使用之前的策略创建 IAM policy 并将其附加到您的组或用户。有关创建 IAM policy 的更多信息，请参阅《IAM 用户指南》中的创建 IAM policy。有关附加 IAM policy 的信息，请参阅《IAM 用户指南》中的添加和删除 IAM 身份权限。

- 附加到 IAM 组或用户的托管策略。除了之前的策略外，您还必须将以下 AWS 托管策略附加到您的组或用户：
  - `AWSGlueConsoleFullAccess`
  - `AWSQuicksightAthenaAccess`

  这些托管政策允许您使用AWS GlueAmazon Athena、和 Amazon QuickSight。有关附加 IAM policy 的信息，请参阅《IAM 用户指南》中的添加和删除 IAM 身份权限。

步骤 1：向 Amazon S3 添加文档

在开始 Amazon Comprehend 分析作业之前，您需要在 Amazon Simple Storage Service (Amazon S3) 中存储客户评论的示例数据集。Amazon S3 将您的数据托管在名为存储桶的容器中。Amazon Comprehend 可以分析存储在存储桶中的文档，并将分析结果发送到存储桶。在此步骤中，您将创建一个 S3 存储桶，在该存储桶中创建输入和输出文件夹，并将示例数据集上传到该存储桶。

主题

- 先决条件
- 下载示例数据
- 创建 Amazon S3 存储桶

- （仅限控制台）创建文件夹
- 上传输入数据

先决条件

在开始之前，请查看 教程：使用 Amazon Comprehend 分析来自客户评论的见解 并完成先决条件。

下载示例数据

以下示例数据集包含来自较大数据集“Amazon 评论——完整”的 Amazon 评论，该数据集与文章《用于文本分类的字符级卷积网络》（Xiang Zhang等人，2015年）一起发表。将数据集下载到您的计算机中。

获取示例数据

1. 将 zip 文件 tutorial-reviews-data.zip 下载到您的计算机上。
2. 将 zip 文件提取到您的计算机上。有两个文件。文件 `THIRD_PARTY_LICENSES.txt` 是 Xiang Zhang 等人发布的数据集的开源许可证。文件 `amazon-reviews.csv` 是您在教程中分析的数据集。

创建 Amazon S3 存储桶

下载示例数据集后，创建一个 Amazon S3 存储桶以存储您的输入和输出数据。您可以使用 Amazon S3 控制台或 AWS Command Line Interface (AWS CLI) 创建 S3 存储桶。

创建 Amazon S3 存储桶（控制台）

在 Amazon S3 控制台中，您可以创建一个存储桶，其名称在所有 AWS中都是唯一的。

创建 S3 存储桶（控制台）

1. 登录 AWS Management Console 并打开 Amazon S3 控制台，网址为 https://console.aws.amazon.com/s3/。
2. 在存储桶中，选择创建存储桶。
3. 对于存储桶名称，请输入全局唯一名称，以描述存储桶用途。
4. 对于区域，选择要在其中创建存储桶的 AWS 区域。您选择的区域必须支持 Amazon Comprehend。要减少延迟，请选择 Amazon Comprehend 支持的离您的地理位置最近的 AWS 区域。有关支持 Amazon Comprehend 的区域列表，请参阅《全球基础设施指南》中的区域表。

5. 保留对象所有权、阻止公共访问的存储桶设置、存储桶版本控制和标签的默认设置。
6. 对于默认加密，请选择禁用。

> **Tip**
>
> 虽然本教程不使用加密，但您可能需要在分析重要数据时使用加密。要进行 end-to-end 加密，您可以加密存储桶中的静态数据，也可以在运行分析任务时对数据进行加密。有关使用加密的更多信息 AWS，请参阅什么是 AWS Key Management Service？ 在《AWS Key Management Service 开发人员指南》中。

7. 查看您的存储桶配置，然后选择创建存储桶。

创建 Amazon S3 存储桶 (AWS CLI)

打开后 AWS CLI，您可以运行`create-bucket`命令来创建用于存储输入和输出数据的存储桶。

创建 Amazon S3 存储桶 (AWS CLI)

1. 请在 AWS CLI中运行以下命令来创建存储桶。将 DOC-EXAMPLE-BUCKET 替换为所有存储桶中唯一的名称。 AWS

```
aws s3api create-bucket --bucket DOC-EXAMPLE-BUCKET
```

默认情况下，该`create-bucket`命令在`us-east-1` AWS 区域中创建存储桶。要在以 `us-east-1` 外的 AWS 区域 中创建存储桶，请添加 `LocationConstraint` 参数以指定您的区域。例如，以下命令在 `us-west-2` 区域中创建一个存储桶。

```
aws s3api create-bucket --bucket DOC-EXAMPLE-BUCKET
--region us-west-2 --create-bucket-configuration LocationConstraint=us-west-2
```

请注意，只有某些区域支持 Amazon Comprehend。有关支持 Amazon Comprehend 的区域列表，请参阅《全球基础设施指南》中的区域表。

2. 要确保成功创建存储桶，请运行以下命令。该命令列出与您的账户关联的所有 S3 存储桶。

```
aws s3 ls
```

（仅限控制台）创建文件夹

接下来，在您的 S3 存储桶中创建两个文件夹。第一个文件夹用于存储输入数据。第二个文件夹是 Amazon Comprehend 发送分析结果的地方。如果您使用 Amazon S3 控制台，则必须手动创建文件夹。如果您使用 AWS CLI，则可以在上传示例数据集或运行分析作业时创建文件夹。因此，我们提供了仅为控制台用户创建文件夹的程序。如果您使用的是 AWS CLI，则将在 上传输入数据 中和 步骤 3：在 Amazon S3 中对文档运行分析作业 中创建文件夹。

在 S3 存储桶中创建文件夹（控制台）

1. 打开 Amazon S3 控制台，网址为：https://console.aws.amazon.com/s3/。
2. 在存储桶中，从存储桶列表中选择您的存储桶。
3. 在概述选项卡中，选择创建文件夹。
4. 对于新文件夹名称，输入 `input`。
5. 对于加密设置，请选择无（使用存储桶设置）。
6. 选择保存。
7. 重复步骤 3 到 6，为分析作业的输出创建另一个文件夹，但在步骤 4 中，输入新的文件夹名称 `output`。

上传输入数据

现在您已经有了存储桶，请上传示例数据集 `amazon-reviews.csv`。您可以使用 Amazon S3 控制台或 AWS CLI将数据上传到 S3 存储桶。

将示例文档上传到存储桶（控制台）

在 Amazon S3 控制台中，将示例数据集文件上传到输入文件夹。

上传示例文档（控制台）

1. 打开 Amazon S3 控制台，网址为：https://console.aws.amazon.com/s3/。
2. 在存储桶中，从存储桶列表中选择您的存储桶。
3. 选择 `input` 文件夹，然后选择上传。
4. 选择添加文件，然后在计算机上选择 `amazon-reviews.csv` 文件。
5. 将其他设置保留为默认值。
6. 选择上传。

将示例文档上传到存储桶 (AWS CLI)

在 S3 存储桶中创建输入文件夹，然后使用 cp 命令将数据集文件上传到新文件夹。

上传示例文档 (AWS CLI)

1. 要将amazon-reviews.csv文件上传到存储桶中的新文件夹，请运行以下 AWS CLI 命令。将 DOC-EXAMPLE-BUCKET 替换为您的存储桶的名称。通过在末尾添加路径 /input/，Amazon S3 会自动在您的存储桶中创建一个名为 input 的新文件夹，并将数据集文件上传到该文件夹。

```
aws s3 cp amazon-reviews.csv s3://DOC-EXAMPLE-BUCKET/input/
```

2. 要确保成功上传文件，请运行以下命令。该命令列出了您的存储桶 input 文件夹的内容。

```
aws s3 ls s3://DOC-EXAMPLE-BUCKET/input/
```

现在，您有一个 S3 存储桶，其中 amazon-reviews.csv 文件位于名为 input 的文件夹中。如果您使用控制台，则存储桶中还有一个 output 文件夹。如果您使用了 AWS CLI，则将在运行 Amazon Comprehend 分析任务时创建输出文件夹。

步骤 2：（仅限 CLI）为 Amazon Comprehend 创建 IAM 角色

只有在使用 AWS Command Line Interface (AWS CLI) 完成本教程时，才需要执行此步骤。如果您正在使用 Amazon Comprehend 控制台运行分析作业，请跳至 步骤 3：在 Amazon S3 中对文档运行分析作业。

要运行分析作业，Amazon Comprehend 需要访问包含示例数据集并包含作业输出的 Amazon S3 存储桶。IAM 角色允许您控制 AWS 服务或用户的权限。在此步骤中，您会为 Amazon Comprehend 创建一个 IAM 角色。然后，您可以创建一项基于资源的策略并将其附加到该角色，该策略授予 Amazon Comprehend 访问您的 S3 存储桶的权限。在此步骤结束时，Amazon Comprehend 将拥有访问您的输入数据、存储输出以及运行情绪和实体分析作业的必要权限。

有关将 IAM 与 Amazon Comprehend 搭配使用的更多信息，请参阅 Amazon Comprehend 如何与 IAM 配合使用。

主题

- 将 IAM policy 附加到 IAM 角色

先决条件

开始之前，请执行以下操作：

- 完成 步骤 1：向 Amazon S3 添加文档。
- 使用代码或文本编辑器来保存 JSON 策略并跟踪您的 Amazon 资源名称（ARN)。

创建 IAM 角色

要访问您的亚马逊简单存储服务 (Amazon S3) 存储桶，Amazon Comprehend 需要担任 (IAM) 角色。AWS Identity and Access Management IAM 角色声明 Amazon Comprehend 为可信实体。在 Amazon Comprehend 担任该角色并成为可信实体后，您可以向 Amazon Comprehend 授予存储桶访问权限。在此步骤中，您需要创建一个角色，将 Amazon Comprehend 标记为可信实体。您可以使用 AWS CLI 或 Amazon Comprehend 控制台创建角色。要使用控制台，请跳至 步骤 3：在 Amazon S3 中对文档运行分析作业。

Amazon Comprehend 控制台允许您选择角色名称包含“Comprehend”且信任策略包含 comprehend.amazonaws.com 的角色。如果您希望控制台显示您的 CLI 创建的角色，请将其配置为符合这些标准。

为亚马逊 Comp AWS rehend (CLI) 创建 IAM 角色

1. 在计算机上的代码或文本编辑器中，将以下信任策略保存为名为 `comprehend-trust-policy.json` 的 JSON 文档。该信任策略宣布 Amazon Comprehend 为可信实体，并允许其担任 IAM 角色。

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

```
}
```

2. 要创建 IAM 角色，请运行以下 AWS CLI 命令。该命令会创建名为 AmazonComprehendServiceRole-access-role 的 IAM 角色并将信任策略附加到该角色。将 *path/* 替换为本地计算机的 JSON 文档路径。

```
aws iam create-role --role-name AmazonComprehendServiceRole-access-role
--assume-role-policy-document file://path/comprehend-trust-policy.json
```

> **Tip**
>
> 如果您收到解析参数时出错消息，则说明您的 JSON 信任策略文件的路径可能不正确。根据您的主目录提供文件的相对路径。

3. 复制 Amazon 资源名称 (ARN) 并将其保存在文本编辑器中。ARN 具有类似于 *arn:aws:iam::123456789012:role/AmazonComprehendServiceRole-access-role* 的格式。您需要这个 ARN 才能运行 Amazon Comprehend 分析作业。

将 IAM policy 附加到 IAM 角色

要访问您的 Amazon S3 存储桶，Amazon Comprehend 需要列出、读取和写入权限。要向 Amazon Comprehend 授予所需的权限，请创建一个 IAM policy 并将其附加到您的 IAM 角色。IAM policy 允许 Amazon Comprehend 从您的存储桶中检索输入数据并将分析结果写入存储桶。创建策略后，将其附加到 IAM 角色。

创建 IAM policy (AWS CLI)

1. 在本地将以下策略另存为名为 comprehend-access-policy.json 的 JSON 文档。它会向 Amazon Comprehend 授予对指定 S3 存储桶的访问权限。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Action": [
                "s3:GetObject"
            ],
            "Resource": [
                "arn:aws:s3:::DOC-EXAMPLE-BUCKET/*"
            ],
```

```
            "Effect": "Allow"
        },
        {
            "Action": [
                "s3:ListBucket"
            ],
            "Resource": [
                "arn:aws:s3:::DOC-EXAMPLE-BUCKET"
            ],
            "Effect": "Allow"
        },
        {
            "Action": [
                "s3:PutObject"
            ],
            "Resource": [
                "arn:aws:s3:::DOC-EXAMPLE-BUCKET/*"
            ],
            "Effect": "Allow"
        }
    ]
}
```

2. 要创建 S3 存储桶访问策略，请运行以下 AWS CLI 命令。将 *path/* 替换为本地计算机的 JSON 文档路径。

```
aws iam create-policy --policy-name comprehend-access-policy
--policy-document file://path/comprehend-access-policy.json
```

3. 复制访问策略 ARN 并将其保存在文本编辑器中。ARN 具有类似于 *arn:aws:iam::123456789012:policy/comprehend-access-policy* 的格式。您需要此 ARN 将访问策略附加到您的 IAM 角色。

将 IAM policy 附加到您的 IAM 角色 (AWS CLI)

- 运行以下命令。将 *policy-arn* 替换为您在上一步中复制的访问策略 ARN。

```
aws iam attach-role-policy --policy-arn policy-arn
--role-name AmazonComprehendServiceRole-access-role
```

现在，您有一个名为 AmazonComprehendServiceRole-access-role 的 IAM 角色，该角色具有 Amazon Comprehend 的信任策略和授予 Amazon Comprehend 访问您的 S3 存储桶的访问权限的访问策略。您还可以将 IAM 角色的 ARN 复制到文本编辑器中。

步骤 3：在 Amazon S3 中对文档运行分析作业

将数据存储到 Amazon S3 以后，您可以开始运行 Amazon Comprehend 分析作业。情绪分析工作确定文档的整体情绪（积极、消极、中性或混合的）。实体分析作业从文档中提取真实世界对象的名称。这些对象包括人物、地点、标题、事件、日期、数量、产品和组织。在此步骤中，您将运行两个 Amazon Comprehend 分析作业，从示例数据集中提取情绪和实体。

主题

先决条件

开始之前，请执行以下操作：

- 完成 步骤 1：向 Amazon S3 添加文档。
- （可选）如果您正在使用 AWS CLI，请填写步骤 2：（仅限 CLI）为 Amazon Comprehend 创建 IAM 角色并准备好您的 IAM 角色 ARN。

分析情绪和实体

您运行的第一项作业将分析示例数据集中每条客户评论的情绪。第二项工作提取每条客户评论中的实体。您可以使用 Amazon Comprehend 控制台或 AWS CLI。

> **Tip**
>
> 请确保您所在的 AWS 地区支持亚马逊 Comprehend。有关更多信息，请参阅在《全球基础设施指南》中的区域表。

分析情绪和实体（控制台）

使用 Amazon Comprehend 控制台时，您一次只能创建一个作业。您需要重复以下步骤才能同时运行情绪和实体分析作业。请注意，对于第一个作业，您需要创建一个 IAM 角色，但对于第二个作业，您

可以重复使用第一个作业的 IAM 角色。只要您使用相同的 S3 存储桶和文件夹，就可以重复使用 IAM 角色。

运行情绪和实体分析作业（控制台）

1. 请确保您所在的区域与 Amazon Simple Storage Service (Amazon S3) 存储桶的创建区域相同。如果您在其他区域，请在导航栏中，从 AWS 区域选择器中选择您创建 S3 存储桶的区域。
2. 打开 Amazon Comprehend 控制台，网址：https://console.aws.amazon.com/comprehend/
3. 选择启动 Amazon Comprehend。
4. 在导航窗格中，选择分析作业。
5. 请选择创建作业。
6. 在作业设置部分，执行以下操作：
   a. 对于名称，请输入 `reviews-sentiment-analysis`。
   b. 在分析类型中，选择情绪。
   c. 在语言中，选择英语。
   d. 将作业加密设置保留为禁用。
7. 请在输入数据部分，执行以下操作：
   a. 在数据来源中，选择我的文档。
   b. 在 S3 位置，选择浏览 S3，然后从存储桶列表中选择您的存储桶。
   c. 在您的 S3 存储桶中，对于对象，选择您的 `input` 文件夹。
   d. 在 `input` 文件夹中，选择示例数据集 `amazon-reviews.csv`，然后选择选择。
   e. 对于输入格式，选择每行一个文档。
8. 请在输出数据部分，执行以下操作：
   a. 在 S3 位置，选择浏览 S3，然后从存储桶列表中选择您的存储桶。
   b. 在您的 S3 存储桶中，对于对象，选择 `output` 文件夹，然后选择选择。
   c. 将加密保持关闭状态。
9. 在访问权限部分，执行以下操作：
   a. 对于 IAM 角色，选择创建 IAM 角色。
   b. 在访问权限中，请选择输入和输出 S3 存储桶。
   c. 在名称后缀中，输入 `comprehend-access-role`。该角色提供对 Amazon S3 存储桶的访问权限。

10. 请选择创建作业。
11. 重复步骤 1 - 10 创建实体分析作业。进行以下更改：

    a. 在作业设置中，对于名称中输入 reviews-entities-analysis。

    b. 在作业设置中，对于分析类型，选择实体。

    c. 在访问权限下，选择使用现有 IAM 角色。在角色名称中，选择AmazonComprehendServiceRole-comprehend-access-role（这与您为情绪作业创建的角色相同）。

分析情绪和实体 (AWS CLI)

您可以使用 start-sentiment-detection-job 和 start-entities-detection-job 命令来运行情绪和实体分析作业。运行每条命令后，会 AWS CLI 显示一个 JSON 对象，其JobId值允许您访问有关任务的详细信息，包括输出 S3 位置。

运行情绪和实体分析作业 (AWS CLI)

1. 通过在 AWS CLI中运行以下命令启动情绪分析作业。将 *arn:aws:iam::123456789012:role/comprehend-access-role* 替换为您之前复制到文本编辑器的 IAM 角色 ARN。如果您的默认 AWS CLI 区域与您创建 Amazon S3 存储桶时所在的区域不同，请添加--region参数并*us-east-1*替换为存储桶所在的区域。

```
aws comprehend start-sentiment-detection-job
--input-data-config S3Uri=s3://DOC-EXAMPLE-BUCKET/input/
--output-data-config S3Uri=s3://DOC-EXAMPLE-BUCKET/output/
--data-access-role-arn arn:aws:iam::123456789012:role/comprehend-access-role
--job-name reviews-sentiment-analysis
--language-code en
[--region us-east-1]
```

2. 提交作业后，复制 JobId 并保存到文本编辑器中。您需要使用 JobId 来查找分析作业的输出文件。
3. 通过运行以下命令启动实体分析作业。

```
aws comprehend start-entities-detection-job
--input-data-config S3Uri=s3://DOC-EXAMPLE-BUCKET/input/
--output-data-config S3Uri=s3://DOC-EXAMPLE-BUCKET/output/
--data-access-role-arn arn:aws:iam::123456789012:role/comprehend-access-role
--job-name reviews-entities-analysis
```

```
--language-code en
[--region us-east-1]
```

4. 提交作业后，复制 JobId 并保存到文本编辑器中。
5. 检查您的作业的状态。您可以通过跟踪其 JobId 来查看作业的进度。

   要跟踪您的情绪分析作业，请运行以下命令。将 *sentiment-job-id* 替换为在运行情绪分析后复制的 JobId。

   ```
   aws comprehend describe-sentiment-detection-job
   --job-id sentiment-job-id
   ```

   要跟踪您的实体分析作业，请运行以下命令。将 *entities-job-id* 替换为在运行实体分析后复制的 JobId。

   ```
   aws comprehend describe-entities-detection-job
   --job-id entities-job-id
   ```

   JobStatus 需要几分钟才能显示为 COMPLETED。

您已经完成了情绪和实体分析作业。应先完成这两项作业，然后再继续下一步。可能需要几分钟时间完成作业。

步骤 4：准备用于数据可视化的 Amazon Comprehend 输出

要准备用于创建数据可视化效果的情绪和实体分析作业的结果，请使用 AWS Glue 和 Amazon Athena。在此步骤中，您将提取到 Amazon Comprehend 结果文件。然后，您可以创建一个 AWS Glue 爬网程序，用于浏览您的数据并自动将其编入 AWS Glue Data Catalog中的表格中。之后，您可以使用无服务器的交互式查询服务访问和转换这些表。 Amazon Athena完成此步骤后，您的 Amazon Comprehend 结果就干净了，可以进行可视化了。

对于 PII 实体检测作业，输出文件是纯文本，而不是压缩存档。输出文件名与输入文件名相同，并在末尾附上 .out。您不需要提取输出文件的步骤。跳到[将数据加载到 AWS Glue Data Catalog](#).

主题

- [先决条件](#)
- [下载输出结果](#)

- 提取输出文件
- 上传提取的文件
- 将数据加载到 AWS Glue Data Catalog
- 准备数据以供分析

先决条件

在开始之前，请完成步骤 3：在 Amazon S3 中对文档运行分析作业。

下载输出结果

Amazon Comprehend 使用 Gzip 压缩来压缩输出文件并将其保存为 tar 存档。提取输出文件的最简单方法是在本地下载 `output.tar.gz` 存档。

在此步骤中，您将下载情绪和实体输出档案。

下载输出文件（控制台）

要查找每个作业的输出文件，请返回 Amazon Comprehend 控制台中的分析作业。分析作业提供输出的 S3 位置，您可以从中下载输出文件。

下载输出文件（控制台）

1. 在 Amazon Comprehend 控制台的导航窗格中，返回分析作业。
2. 选择您的情绪分析作业 `reviews-sentiment-analysis`。
3. 在输出下，选择输出数据位置旁边显示的链接。这会将您重定向到 S3 存储桶中的 `output.tar.gz` 存档。
4. 在概述选项卡中，选择下载。
5. 在您的计算机上，将存档重命名为 `sentiment-output.tar.gz`。由于所有输出文件都具有相同的名称，因此这可以帮助您跟踪情绪和实体文件。
6. 重复步骤 1 - 4，查找并下载 `reviews-entities-analysis` 作业的输出。在您的计算机上，将存档重命名为 `entities-output.tar.gz`。

下载输出文件 (AWS CLI)

要查找每个作业的输出文件，请使用分析作业中的 `JobId` 来查找输出的 S3 位置。然后，使用 `cp` 命令将输出文件下载到您的计算机。

下载输出文件 (AWS CLI)

1. 要列出有关您的情绪分析作业的详细信息，请运行以下命令。替换 *sentiment-job-id* 为您保存的情绪 JobId。

```
aws comprehend describe-sentiment-detection-job --job-id sentiment-job-id
```

如果您不知道您的 JobId，则可以运行以下命令来列出所有情绪作业，并按名称筛选您的作业。

```
aws comprehend list-sentiment-detection-jobs
--filter JobName="reviews-sentiment-analysis"
```

2. 在 OutputDataConfig 对象中，找到 S3Uri 值。S3Uri 值应类似于以下格式：*s3://DOC-EXAMPLE-BUCKET/.../output/output.tar.gz*。将此值复制到文本编辑器。
3. 要将情绪输出存档下载到本地目录，请运行以下命令。将 S3 存储桶路径替换为您在上一步中复制的 S3Uri。将 *path/* 替换为本地目录的文件夹路径。名称 sentiment-output.tar.gz 替换了原始存档名称，以帮助您跟踪情绪和实体文件。

```
aws s3 cp s3://DOC-EXAMPLE-BUCKET/.../output/output.tar.gz
path/sentiment-output.tar.gz
```

4. 要列出有关您的实体分析作业的详细信息，请运行以下命令。

```
aws comprehend describe-entities-detection-job
--job-id entities-job-id
```

如果您不知道您的 JobId，运行以下命令来列出所有实体作业，并按名称筛选您的作业。

```
aws comprehend list-entities-detection-jobs
--filter JobName="reviews-entities-analysis"
```

5. 从实体作业描述中的 OutputDataConfig 对象中，复制 S3Uri 值。
6. 要将实体输出存档下载到本地目录，请运行以下命令。将 S3 存储桶路径替换为您在上一步中复制的 S3Uri。将 *path/* 替换为本地目录的文件夹路径。名称 entities-output.tar.gz 取代了原始存档名称。

```
aws s3 cp s3://DOC-EXAMPLE-BUCKET/.../output/output.tar.gz
path/entities-output.tar.gz
```

提取输出文件

在访问 Amazon Comprehend 结果之前，请先解压情绪和实体存档。您可以使用本地文件系统或终端来解压缩存档。

提取输出文件（GUI 文件系统）

如果您使用的是 macOS，请双击 GUI 文件系统中的存档文件以从存档中提取输出文件。

如果您使用 Windows，则可以使用第三方工具（例如 7-Zip）在 GUI 文件系统中提取输出文件。在 Windows 中，必须执行两个步骤才能访问存档中的输出文件。首先解压存档，然后提取存档。

将情绪文件重命名为 `sentiment-output`，将实体文件重命名为 `entities-output`，以区分输出文件。

提取输出文件（终端）

如果您使用的是 Linux 或 macOS，则可以使用标准终端。如果您使用 Windows，则必须有权访问 Unix 风格的环境（例如 Cygwin）才能运行 tar 命令。

要从情绪存档中提取情绪输出文件，请在本地终端中运行以下命令。

```
tar -xvf sentiment-output.tar.gz --transform 's,^,sentiment-,'
```

请注意，`--transform` 参数将前缀 `sentiment-` 添加到存档内的输出文件中，将文件重命名为 `sentiment-output`。这使您可以区分情绪和实体输出文件并防止覆盖。

要从实体存档中提取实体输出文件，请在本地终端中运行以下命令。

```
tar -xvf entities-output.tar.gz --transform 's,^,entities-,'
```

`--transform` 参数将前缀 `entities-` 添加到输出文件名中。

> **Tip**
>
> 为了节省 Amazon S3 中的存储成本，您可以在上传文件之前使用 Gzip 再次压缩文件。解压缩和解压缩原始存档很重要，因为 AWS Glue 无法自动从 tar 存档中读取数据。但是， AWS Glue 可以从 Gzip 格式的文件中读取。

上传提取的文件

解压文件后，将其上传到您的存储桶。为了正确读取数据，必须将情绪和实体输出文件存储在单独的文件夹中。 AWS Glue 在您的存储桶中，为提取的情绪结果创建一个文件夹，为提取的实体结果创建第二个文件夹。您可以使用 Amazon S3 控制台来创建文件夹，也可以使用 AWS CLI。

将提取后的文件上传到 Amazon S3（控制台）

在您的 S3 存储桶中，为提取的情绪结果创建一个文件夹，为实体结果文件创建一个文件夹。然后，将提取的结果文件上传到各自的文件夹中。

将提取后的文件上传到 Amazon S3（控制台）

1. 打开 Amazon S3 控制台，网址为：https://console.aws.amazon.com/s3/。
2. 在存储桶中，选择您的存储桶，然后选择创建文件夹。
3. 对于新文件夹的名称，输入 `sentiment-results` 并选择保存。此文件夹将包含提取的情绪输出文件。
4. 在存储桶的概述选项卡中，从存储桶内容列表中选择新文件夹 `sentiment-results`。选择上传。
5. 选择添加文件，从本地计算机中选择 `sentiment-output` 文件，然后选择下一步。
6. 将 “管理用户”、“其他 AWS 账户用户访问权限” 和 “管理公共权限” 选项保留为默认值。选择下一步。
7. 对于存储类别，选择标准。将加密、元数据和标记选项保留为默认值。选择下一步。
8. 查看上传选项，然后选择上传。
9. 重复步骤 1 - 8，创建一个名为 `entities-results` 的文件夹，并将 `entities-output` 文件上传到该文件夹。

将提取后的文件上传到 Amazon S3 (AWS CLI)

使用 `cp` 命令上传文件时，您可以在 S3 存储桶中创建文件夹。

将提取后的文件上传到 Amazon S3 (AWS CLI)

1. 通过运行以下命令创建一个情绪文件夹，并将您的情绪文件上传到该文件夹。*`path/`* 替换为提取的情绪输出文件的本地路径。

```
aws s3 cp path/sentiment-output s3://DOC-EXAMPLE-BUCKET/sentiment-results/
```

2. 通过运行以下命令创建一个实体输出文件夹，并将您的实体文件上传到该文件夹。*path/* 替换为提取的实体输出文件的本地路径。

```
aws s3 cp path/entities-output s3://DOC-EXAMPLE-BUCKET/entities-results/
```

将数据加载到 AWS Glue Data Catalog

要将结果导入数据库，你可以使用 AWS Glue 爬虫。 AWS Glue 爬虫会扫描文件并发现数据的架构。然后，它将数据排列在 AWS Glue Data Catalog （无服务器数据库）中的表中。您可以使用 AWS Glue 控制台或. 创建爬虫。 AWS CLI

将数据加载到 AWS Glue Data Catalog （控制台）

创建可分别扫描您的`sentiment-results`和`entities-results`文件夹的 AWS Glue 抓取工具。AWS Glue 的新 IAM 角色授予爬虫程序访问您的 S3 存储桶的权限。您在设置爬网程序时创建此 IAM 角色。

将数据加载到 AWS Glue Data Catalog （控制台）

1. 确保您所在的地区支持 AWS Glue。如果您在另一区域，请在导航栏中，从区域选择器中选择选择支持的区域。有关支持的区域列表 AWS Glue，请参阅《全球基础设施指南》中的区域表。
2. 打开 AWS Glue 控制台，网址为 https://console.aws.amazon.com/glue/。
3. 在导航窗格中，选择爬网程序，然后选择添加爬网程序。
4. 对于爬网程序名称，输入 `comprehend-analysis-crawler`，然后选择下一步。
5. 对于爬网程序源类型，选择数据存储，然后选择下一步。
6. 对于添加数据存储，请执行以下操作：
   a. 对于选择数据存储，请选择 S3。
   b. 将连接留空。
   c. 对于爬网数据位于选项，选择在我的账户中指定路径。
   d. 在包含路径中，输入情绪输出文件夹的完整 S3 路径：`s3://DOC-EXAMPLE-BUCKET/sentiment-results`。
   e. 选择下一步。
7. 对于添加其他数据存储，请依次选择是、下一步。重复步骤 6，但输入实体输出文件夹的完整 S3 路径：`s3://DOC-EXAMPLE-BUCKET/entities-results`。
8. 对于添加其他数据存储，请依次选择是、下一步。

9. 对于选择 IAM 角色，请执行以下操作之一：

   a. 选择创建 IAM 角色。

   b. 对于 IAM 角色，输入 `glue-access-role`，然后选择下一步。

10. 在为此爬网程序创建计划中，选择按需运行，然后选择下一步。

11. 对于配置爬网程序的输出，请执行以下操作：

    a. 对于数据库，选择添加数据库。

    b. 对于 Database name (数据库名称)，请输入 `comprehend-results`。该数据库将存储您的 Amazon Comprehend 输出表。

    c. 将其他选项保留为默认设置，然后选择下一步。

12. 检查爬网程序信息，然后选择完成。

13. 在 Glue 控制台的爬网程序中，选择 `comprehend-analysis-crawler` 并选择运行爬网程序。爬网程序可能需要几分钟时间来完成。

将数据加载到 AWS Glue Data Catalog (AWS CLI)

为创建一个 IAM 角色 AWS Glue 以提供访问您的 S3 存储桶的权限。然后，在 AWS Glue Data Catalog中创建数据库。最后，创建并运行一个爬网程序，将您的数据加载到数据库的表中。

将数据加载到 AWS Glue Data Catalog (AWS CLI)

1. 要为创建 IAM 角色 AWS Glue，请执行以下操作：

   a. 将以下信任策略另存为计算机上名为 `glue-trust-policy.json` 的 JSON 文档。

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

b. 运行以下命令以创建 IAM 角色。将 *path/* 替换为本地计算机的 JSON 文档路径。

```
aws iam create-role --role-name glue-access-role
--assume-role-policy-document file://path/glue-trust-policy.json
```

c. AWS CLI 列出新角色的 Amazon 资源编号 (ARN) 时，将其复制并保存到文本编辑器中。

d. 将以下 IAM policy 另存为计算机上名为 `glue-access-policy.json` 的 JSON 文档。该策略授予对您的结果文件夹进行爬网的 AWS Glue 权限。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "s3:GetObject",
                "s3:PutObject"
            ],
            "Resource": [
                "arn:aws:s3:::DOC-EXAMPLE-BUCKET/sentiment-results*",
                "arn:aws:s3:::DOC-EXAMPLE-BUCKET/entities-results*"
            ]
        }
    ]
}
```

e. 要创建 IAM policy，请运行以下命令。将 *path/* 替换为本地计算机的 JSON 文档路径。

```
aws iam create-policy --policy-name glue-access-policy
--policy-document file://path/glue-access-policy.json
```

f. AWS CLI 列出访问策略的 ARN 后，将其复制并保存到文本编辑器中。

g. 使用以下命令将新策略附加到 IAM 角色。将 *policy-arn* 替换为您在上一步中复制的 IAM policy ARN。

```
aws iam attach-role-policy --policy-arn policy-arn
--role-name glue-access-role
```

h. 通过运行以下命令将 AWS 托管策略附加`AWSGlueServiceRole`到您的 IAM 角色。

```
aws iam attach-role-policy --policy-arn
```

```
arn:aws:iam::aws:policy/service-role/AWSGlueServiceRole
--role-name glue-access-role
```

2. 通过运行以下命令创建 AWS Glue 数据库。

```
aws glue create-database
--database-input Name="comprehend-results"
```

3. 通过运行以下命令创建新的 AWS Glue 爬虫。*glue-iam-role-arn*替换为您的 IA AWS Glue M 角色的 ARN。

```
aws glue create-crawler
--name comprehend-analysis-crawler
--role glue-iam-role-arn
--targets S3Targets=[
{Path="s3://DOC-EXAMPLE-BUCKET/sentiment-results"},
{Path="s3://DOC-EXAMPLE-BUCKET/entities-results"}]
--database-name comprehend-results
```

4. 通过运行以下命令启动爬网程序。

```
aws glue start-crawler --name comprehend-analysis-crawler
```

爬网程序可能需要几分钟时间来完成。

准备数据以供分析

现在，您的数据库中填充了 Amazon Comprehend 结果。但是，结果是嵌套的。要解除它们的嵌套，你需要在中运行几条 SQL 语句。 Amazon Athena Amazon Athena 是一项交互式查询服务，可使用标准 SQL 轻松分析 Amazon S3 中的数据。Athena 是无服务器的，因此无需管理基础架构，而且它有定价模式。 pay-per-query 在此步骤中，您将创建用于分析和可视化的已清理数据的新表。您可以使用 Athena 控制台来准备数据。

准备数据

1. 从 https://console.aws.amazon.com/athena/ 打开 Athena 控制台。
2. 在查询编辑器中，选择 Settings（设置），然后选择 Manage（管理）。
3. 在查询结果的位置中，输入 `s3://DOC-EXAMPLE-BUCKET/query-results/`。这将在您的存储桶`query-results`中创建一个名为的新文件夹，用于存储您运行的 Amazon Athena 查询的输出。选择保存。

4. 在查询编辑器中，选择编辑器。
5. 对于数据库，选择您创建 AWS Glue 的数据comprehend-results库。
6. 在表部分中，应该有两个名为 sentiment_results 和 entities_results 的表。预览表格，确保爬网程序加载了数据。在每个表格的选项（表格名称旁边的三个点）中，选择预览表格。自动运行简短查询。检查结果窗格以确保表中包含数据。

> **Tip**
>
> 如果表中没有任何数据，请尝试检查 S3 存储桶中的文件夹。确保有一个用于存储实体结果的文件夹和一个用于存储情绪结果的文件夹。然后，尝试运行新的 AWS Glue 爬虫。

7. 要取消嵌套 sentiment_results 表，请在查询编辑器中输入以下查询，然后选择运行。

```
CREATE TABLE sentiment_results_final AS
SELECT file, line, sentiment,
sentimentscore.mixed AS mixed,
sentimentscore.negative AS negative,
sentimentscore.neutral AS neutral,
sentimentscore.positive AS positive
FROM sentiment_results
```

8. 要开始取消嵌套实体表，请在查询编辑器中输入以下查询，然后选择运行。

```
CREATE TABLE entities_results_1 AS
SELECT file, line, nested FROM entities_results
CROSS JOIN UNNEST(entities) as t(nested)
```

9. 要完成取消嵌套实体表，请在查询编辑器中输入以下查询，然后选择运行查询。

```
CREATE TABLE entities_results_final AS
SELECT file, line,
nested.beginoffset AS beginoffset,
nested.endoffset AS endoffset,
nested.score AS score,
nested.text AS entity,
nested.type AS category
FROM entities_results_1
```

您的 sentiment_results_final 表格应如下所示，列名分别为文件、行、情绪、混合、消极、中性和积极。该表的每个单元格应有一个值。情绪列描述了某条评论中最有可能的总体情绪。混合、消极、中性、积极的列给出了每种情绪的分数。

Results

	file	line	sentiment	mixed	negative	neutral	positive
1	amazon-reviews.csv	6	MIXED	0.9862896203994751	0.0015502438182011247	1.6660270921420306E-4	0.0119935879483
2	amazon-reviews.csv	8	POSITIVE	0.0012987082591280341	0.01186690479516983	0.174478679895401	0.8123556375503
3	amazon-reviews.csv	11	POSITIVE	6.536858109029708 4E-6	0.0013866390800103545	0.007405391428619623	0.9912014007568
4	amazon-reviews.csv	13	POSITIVE	4.7155481297522783E-4	0.24615342915058136	0.017713148146867752	0.7356618046760
5	amazon-reviews.csv	14	POSITIVE	1.5821871784282848E-5	0.06828905642032623	0.014075091108679771	0.9176200628280
6	amazon-reviews.csv	16	MIXED	0.9864791035652161	8.548551704734564E-4	1.0789262159960344E-4	0.0125581491738
7	amazon-reviews.csv	20	NEGATIVE	1.1621621524682269E-4	0.9815887212753296	0.004688907880336046	0.0136061981320
8	amazon-reviews.csv	21	POSITIVE	4.663573781726882E-5	0.009533549658954144	0.0015825830632820725	0.9888372421264
9	amazon-reviews.csv	23	POSITIVE	1.7699007003102452E-4	0.40269607305526733	0.0018250439316034317	0.5953019261360
10	amazon-reviews.csv	25	POSITIVE	1.8434448065818287E-6	1.15832663141191E-4	0.0010993879986926913	0.9987829327583

您 entities_results_final 表应如下所示，列名分别为文件、行、开始偏移量、结束偏移量、分数、实体 和 类别。该表的每个单元格应有一个值。分数列表示 Amazon Comprehend 对其检测到的实体的置信度。该类别表示 Comprehend 检测到的是哪种实体。

Results

	file	line	beginoffset	endoffset	score	entity	category
1	amazon-reviews.csv	0	15	22	0.9885989378545348	English	OTHER
2	amazon-reviews.csv	2	24	28	0.9699371997593782	2 me	QUANTITY
3	amazon-reviews.csv	2	94	95	0.6523066984191679	2	QUANTITY
4	amazon-reviews.csv	2	125	126	0.713791396412543	2	QUANTITY
5	amazon-reviews.csv	4	30	36	0.9957169942979278	kindle	COMMERCIAL_ITEM
6	amazon-reviews.csv	5	1	10	0.9979111763962706	Hawthorne	PERSON
7	amazon-reviews.csv	5	135	142	0.5065408081314243	Puritan	OTHER
8	amazon-reviews.csv	5	143	148	0.7702269458801602	Salem	LOCATION
9	amazon-reviews.csv	5	211	229	0.999675563687763	The Scarlet Letter	TITLE
10	amazon-reviews.csv	5	233	236	0.8944631322676461	one	QUANTITY

现在，您已将 Amazon Comprehend 结果加载到表格中，您可以对数据进行可视化并从中提取有意义的见解。

第 5 步：在亚马逊中可视化 Amazon Comprehend 的输出 QuickSight

将 Amazon Comprehend 结果存储在表格中后，您可以使用亚马逊连接数据并对其进行可视化。QuickSightAmazon QuickSight 是一款用于可视化数据的 AWS 托管商业智能 (BI) 工具。借 QuickSight 助 Amazon，您可以轻松连接到您的数据源并创建强大的视觉效果。在此步骤中，您将Amazon QuickSight 连接到您的数据，创建从数据中提取见解的可视化效果，并发布可视化控制面板。

主题

先决条件

在开始之前，请完成步骤 4：准备用于数据可视化的 Amazon Comprehend 输出。

授予亚马逊 QuickSight 访问权限

要导入数据，亚马逊 QuickSight 需要访问您的亚马逊简单存储服务 (Amazon S3) 存储桶 Amazon Athena 和表。要让 Amazon QuickSight 访问您的数据，您必须以 QuickSight 管理员身份登录并有权编辑资源权限。如果您无法完成以下步骤，请从概述页面 教程：使用 Amazon Comprehend 分析来自客户评论的见解 查看 IAM 先决条件。

让 Amazon QuickSight 访问您的数据

1. 打开 Amazon QuickSight 控制台。
2. 如果这是您首次使用 Amazon QuickSight，控制台会提示您通过提供电子邮件地址来创建新的管理员用户。对于电子邮件地址，输入您的 AWS 账户相同的电子邮件地址。选择继续。
3. 登录后，在导航栏中选择您的个人资料名称，然后选择管理 QuickSight。您必须以管理员身份登录才能查看 “管理 QuickSight” 选项。
4. 选择安全性和权限。
5. 要QuickSight 访问 AWS 服务，请选择 “添加” 或 “删除”。

6. 选择 Amazon S3。
7. 从选择 Amazon S3 存储桶中，为 S3 存储桶和 Athena 工作组的写入权限选择您的 S3 存储桶。
8. 选择完成。
9. 选择更新。

导入数据集

在创建可视化之前，您必须将情绪和实体数据集添加到 Amazon QuickSight。您可以使用 Amazon QuickSight 控制台执行此操作。您可以从中导入未嵌套的情绪和未嵌套的实体表。 Amazon Athena

导入您的数据集

1. 打开 Amazon QuickSight 控制台。
2. 在导航栏中的数据集中，选择新建数据集。
3. 在创建数据集中，选择 Athena。
4. 在数据来源名称中，输入 `reviews-sentiment-analysis` 然后选择创建数据来源。
5. 对于 Database (数据库)，选择 `comprehend-results` 数据库。
6. 对于表，选择情绪表 `sentiment_results_final`，然后选择选择。
7. 选择导入到 SPICE 以加快分析速度，然后选择可视化。SPICE 是 QuickSight内存计算引擎，在创建可视化效果时，它提供的分析速度比直接查询更快。
8. 返回 Amazon QuickSight 控制台并选择数据集。重复步骤 1-7 创建实体数据集，但要进行以下更改：

    a. 对于数据来源名称，输入 `reviews-entities-analysis`。

    b. 对于表，选择实体表 `entities_results_final`。

创建情绪可视化

现在您可以在 Amazon 中访问您的数据了 QuickSight，您可以开始创建可视化效果了。您可以使用 Amazon Comprehend 情绪数据创建饼图。饼图显示了积极、中性、混合和消极的评论的比例。

可视化情绪数据

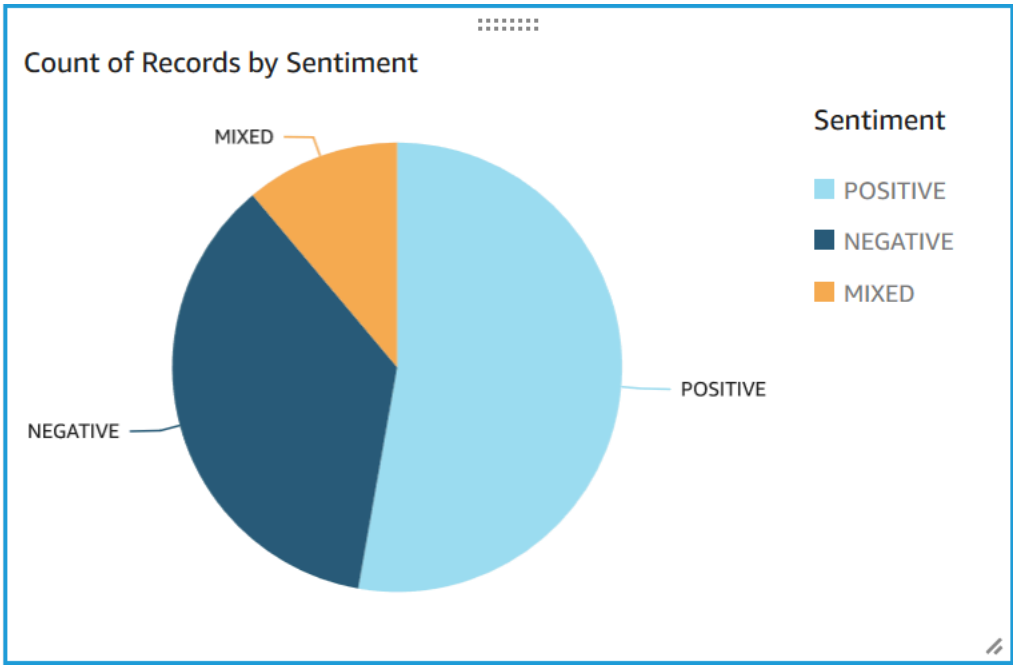
1. 在 Amazon QuickSight 控制台中，选择 “分析”，然后选择 “新建分析”。
2. 从您的数据集中，选择情绪数据集 `sentiment_results_final`，然后选择创建分析。

3. 在可视化编辑器的字段列表中，选择情绪。

> **Note**
>
> 字段列表中的值取决于您在 Amazon Athena中创建表时使用的列名。如果您在 SQL 查询中更改了提供的列名，则字段列表名称将与这些可视化示例中使用的名称不同。

4. 对于视觉类型，请选择饼图。

将显示与下图类似的饼图，包括积极、中性、混合和消极部分。要查看某个部分的数量和百分比，请将鼠标悬停在该部分上。



创建实体可视化

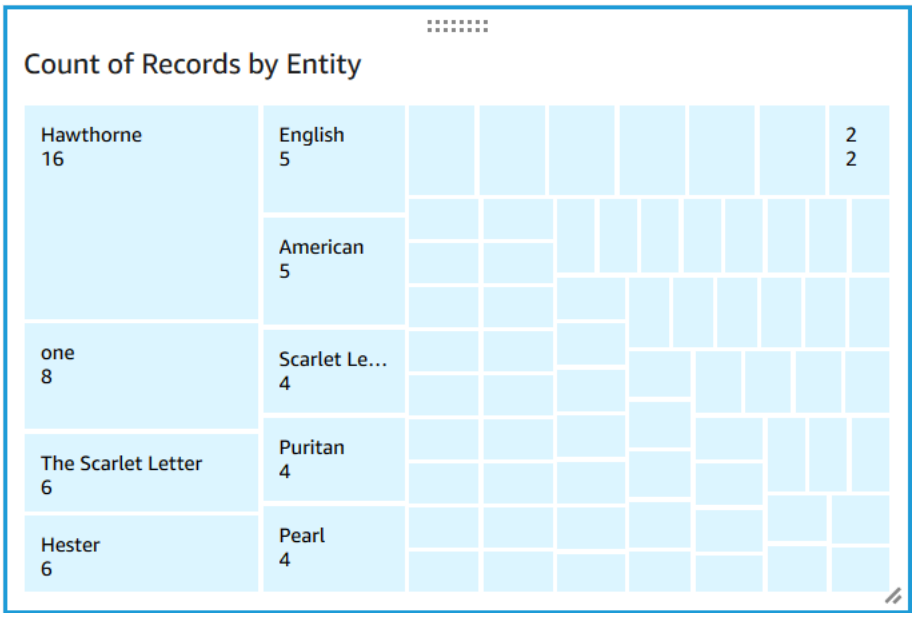
现在使用实体数据集创建第二个可视化。您可以创建数据中不同实体的树形图。树形图中的每个块代表一个实体，块的大小与该实体在数据集中出现的次数相关。

可视化实体数据

1. 在可视化控制窗格中，选择数据集旁边的添加、编辑、替换和删除数据集图标。
2. 选择添加数据集。
3. 在选择要添加的数据集中，从数据集列表中选择您的实体数据集 `entities_results_final`，然后选择选择。

4. 在可视化控制窗格中，选择数据集下拉菜单，然后选择实体数据集 `entities_results_final`。
5. 在字段列表中，选择实体。
6. 对于视觉类型，请选择树形图。

饼图旁边会显示类似于以下内容的树形图。要查看特定实体的数量，请将鼠标悬停在块上。



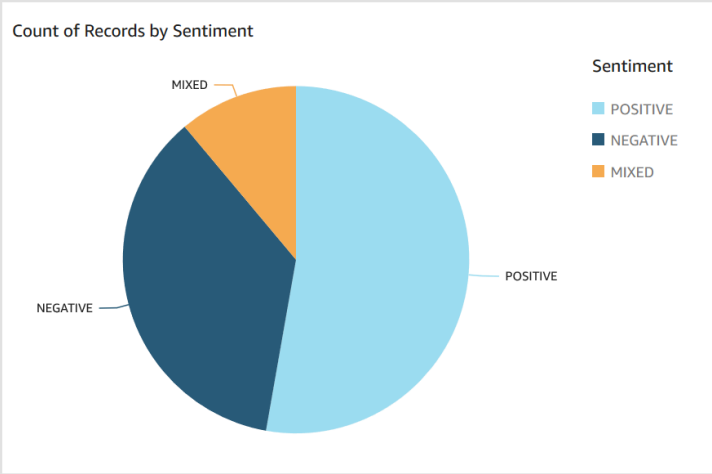
发布控制面板

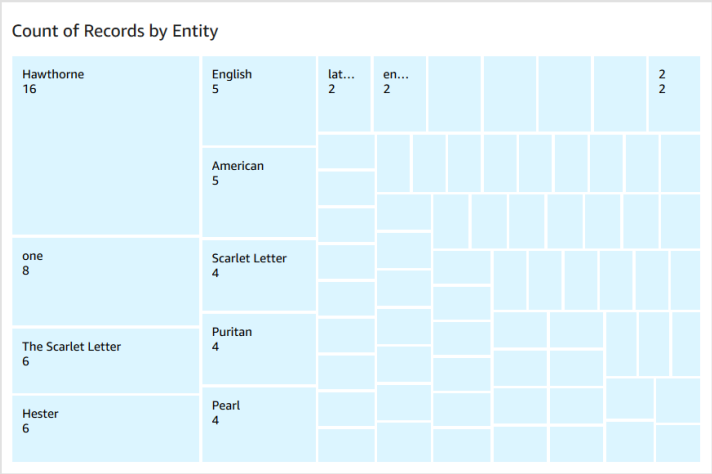
创建可视化效果后，您可以将其发布为控制面板。您可以使用控制面板执行各种任务，例如与您的用户共享 AWS 账户、将其另存为 PDF 或将其作为报告通过电子邮件发送（仅限于 Amazon 的企业版 QuickSight）。在此步骤中，您将可视化作为控制面板发布到您的账户中。

发布控制面板

1. 在导航栏中，选择共享。
2. 选择 Publish dashboard（发布控制面板）。
3. 选择发布新控制面板为，然后输入控制面板的名称 `comprehend-analysis-reviews`。
4. 选择 Publish dashboard（发布控制面板）。
5. 选择右上角的关闭按钮，关闭与用户共享控制面板窗格。
6. 在 Amazon QuickSight 控制台的导航窗格中，选择控制面板。新控制面板的缩略图 `comprehend-analysis-reviews` 应显示在控制面板下。选择仪表板进行查看。

现在，您有一个包含情绪和实体可视化的控制面板，与以下示例类似。





> **Tip**
>
> 如果要编辑控制面板中的可视化效果，请返回分析并编辑要更新的可视化效果。然后，将控制面板作为新控制面板或替换现有控制面板再次发布。

清理

完成本教程后，您可能需要清理所有不想再使用的 AWS 资源。活跃 AWS 资源可能会继续在您的账户中产生费用。

以下操作可以帮助避免产生持续费用：

- 取消您的 Amazon QuickSight 订阅。Amazon QuickSight 是一项按月订阅服务。要取消订阅，请参阅《亚马逊 QuickSight 用户指南》中的 “取消订阅”。
- 删除您的 Amazon S3 存储桶。Amazon S3 会向您收取存储费用。要清理您的 Amazon S3 资源，请删除您的存储桶。有关删除存储桶的信息，请参阅《Amazon Simple Storage Service 用户指南》中的如何删除 S3 存储桶？ 在删除存储桶之前，请务必保存所有重要文件。
- 清除您的 AWS Glue Data Catalog。按月向您 AWS Glue Data Catalog 收取存储费用。您可以删除您的数据库，以免产生持续的费用。有关管理 AWS Glue Data Catalog 数据库的信息，请参阅《AWS Glue 开发人员指南》中的在 AWS Glue 控制台上使用数据库。在清除任何数据库或表之前，请务必导出数据。

使用 Amazon S3 对象 Lambda 接入点获取个人身份信息 (PII)

使用 Amazon S3 对象 Lambda 接入点来配置从您的 Amazon S3 存储桶中检索包含个人身份信息 (PII) 的文档。您可以控制对包含 PII 的文档的访问权限，并编辑文档中的 PII。有关 Amazon Comprehend 如何检测文档中的 PII 的更多信息，请参阅 检测 PII 实体。Amazon S3 对象 Lambda 接入点使用 AWS Lambda 函数自动转换标准 Amazon S3 GET 请求的输出。有关更多信息，请参阅《Amazon Simple Storage Service 用户指南》中的使用 S3 对象 Lambda 转换对象。

当您为 PII 创建 Amazon S3 对象 Lambda 接入点时，将使用 Amazon Comprehend Lambda 函数处理文档，以控制对包含 PII 的文档的访问权限并从文档中编辑 PII。

当您为 PII 创建 Amazon S3 对象 Lambda 接入点时，使用以下 Amazon Comprehend Lambda 函数处理文档：

- `ComprehendPiiAccessControlS3ObjectLambda`：控制对 S3 存储桶中存储有 PII 的文档的访问。有关此 Lambda 函数的更多信息，请登录查看中的 ComprehendPiiAccessControlS3 ObjectLambda 函数。AWS Management Console AWS Serverless Application Repository
- `ComprehendPiiRedactionS3ObjectLambda`：编辑您的 Amazon S3 存储桶中的文档中的 PII。有关此 Lambda 函数的更多信息，请登录查看中的 ComprehendPiiRedactionS3 ObjectLambda 函数。AWS Management Console AWS Serverless Application Repository

有关如何从 AWS Serverless Application Repository 部署无服务器应用程序的信息，请参阅《AWS 无服务器应用程序开发人员指南》中的部署应用程序。

主题

- 控制对包含个人身份信息 (PII) 的文档的访问权限
- 编辑文档中的个人身份信息 (PII)

控制对包含个人身份信息 (PII) 的文档的访问权限

您可以使用 Amazon S3 对象 Lambda 接入点来控制对包含个人身份信息 (PII) 的文档的访问权限。

为确保只有经过授权的用户才能访问存储在您的 Amazon S3 存储桶中的包含 PII 的文档，您可以使用 `ComprehendPiiAccessControlS3ObjectLambda` 函数。此 Lambda 函数在处理针对文档对象的标准 Amazon S3 GET 请求时使用该ContainsPiiEntities操作。

例如，如果您的 S3 存储桶中的文档包含 PII（如信用卡号或银行账户信息），则您可以配置 ComprehendPiiAccessControlS3ObjectLambda 函数检测这些 PII 实体类型并限制未授权用户进行访问。有关支持的 PII 实体类型的更多信息，请参阅 PII 通用实体类型。

有关此 Lambda 函数的更多信息，请登录查看中的 ComprehendPiiAccessControlS3 ObjectLambda 函数。AWS Management Console AWS Serverless Application Repository

创建 Amazon S3 对象 Lambda 接入点以控制对文档的访问权限

以下示例创建 Amazon S3 对象 Lambda 接入点以控制对包含社会保险号码的文档的访问权限。

使用 AWS Command Line Interface 创建 Amazon S3 对象 Lambda 接入点

创建 Amazon S3 对象 Lambda 接入点配置并将该配置保存在名为 config.json 的文件中。

```
{
    "SupportingAccessPoint": "s3-default-access-point-name-arn",
    "TransformationConfigurations": [
        {
            "Actions": [
                "s3:GetObject"
            ],
            "ContentTransformation": {
                "AwsLambda": {
                    "FunctionArn": "comprehend-pii-access-control-s3-object-lambda-
arn",
                    "FunctionPayload": "{\"pii_entities_types\": \"SSN\"}"
                }
            }
        }
    ]
}
```

以下示例根据 config.json 文件中定义的配置创建 Amazon S3 对象 Lambda 接入点。

此示例的格式适用于 Unix、Linux 和 macOS。对于 Windows，请将每行末尾的反斜杠 (\) Unix 行继续符替换为脱字号 (^)。

```
aws s3control create-banner-access-point \
    --region region \
    --account-id account-id \
    --name s3-object-lambda-access-point \
```

```
    --configuration file://config.json
```

调用 Amazon S3 对象 Lambda 接入点来控制对文档的访问权限

以下示例调用 Amazon S3 对象 Lambda 接入点来控制对文档的访问权限

使用 AWS Command Line Interface 调用 Amazon S3 对象 Lambda 接入点

以下示例使用 AWS CLI 调用 Amazon S3 对象 Lambda 接入点。

此示例的格式适用于 Unix、Linux 和 macOS。对于 Windows，请将每行末尾的反斜杠 (\) Unix 行继续符替换为脱字号 (^)。

```
aws s3api get-object \
    --region region \
    --bucket s3-object-lambda-access-point-name-arn \
    --key object-prefix-key output-file-name
```

编辑文档中的个人身份信息 (PII)

您可以使用 Amazon S3 对象 Lambda 接入点来编辑文档中的个人身份信息 (PII)。

要编辑存储在 S3 存储桶中的文档中的 PII 实体类型，请使用函数 `ComprehendPiiRedactionS3ObjectLambda`。此 Lambda 函数在处理针对文档对象的标准 Amazon S3 GET 请求时使用ContainsPiiEntities和DetectPiiEntities操作。

例如，如果您的 S3 存储桶中的文档包含 PII（如信用卡号或银行账户信息），则您可以配置 `ComprehendPiiRedactionS3ObjectLambda` 函数来检测 PII，然后返回这些文档的副本，其中的 PII 实体类型已被编辑。有关支持的 PII 实体类型的更多信息，请参阅 PII 通用实体类型。

有关此 Lambda 函数的更多信息，请登录查看中的 ComprehendPiiRedactionS3 ObjectLambda 函数。AWS Management Console AWS Serverless Application Repository

创建 Amazon S3 对象 Lambda 接入点以编辑文档中的 PII

以下示例创建 Amazon S3 对象 Lambda 接入点，用于编辑文档中的信用卡号。

使用 AWS Command Line Interface 创建 Amazon S3 对象 Lambda 接入点

创建 Amazon S3 对象 Lambda 接入点配置，并将配置保存到名为 `config.json` 的文件中。

```
{
    "SupportingAccessPoint": "s3-default-access-point-name-arn",
    "TransformationConfigurations": [
        {
            "Actions": [
                "s3:GetObject"
            ],
            "ContentTransformation": {
                "AwsLambda": {
                    "FunctionArn": "comprehend-pii-redaction-s3-object-lambda-arn",
                    "FunctionPayload": "{\"pii_entities_types\": \"CREDIT_DEBIT_NUMBER
\"}"
                }
            }
        }
    ]
}
```

以下示例根据 config.json 中定义的配置创建 Amazon S3 对象 Lambda 接入点。

此示例的格式适用于 Unix、Linux 和 macOS。对于 Windows，请将每行末尾的反斜杠 (\) Unix 行继续符替换为脱字号 (^)。

```
aws s3control create-access-point-for-object-lambda \
    --region region \
    --account-id account-id \
    --name s3-object-lambda-access-point \
    --configuration file://config.json
```

调用 Amazon S3 对象 Lambda 接入点来编辑文档中的 PII

以下示例调用了 Amazon S3 对象 Lambda 接入点来编辑文档中的 PII

使用 AWS Command Line Interface 调用 Amazon S3 对象 Lambda 接入点

以下示例使用 AWS CLI 调用 Amazon S3 对象 Lambda 接入点。

此示例的格式适用于 Unix、Linux 和 macOS。对于 Windows，请将每行末尾的反斜杠 (\) Unix 行继续符替换为脱字号 (^)。

```
aws s3api get-object \
    --region region \
```

```
    --bucket s3-object-lambda-access-point-name-arn \
    --key object-prefix-key output-file-name
```

解决方案：使用 Amazon Comprehend 分析文本和 OpenSearch

AWS提供了使用 Amazon Compre OpenSearch hend 和该服务进行文本分析的参考实现。Amazon Comprehend 提供文本 OpenSearch 分析并提供文档索引、搜索和可视化。

有关更多信息，请参阅使用 OpenSearch 和 Amazon Comprehend 分析文本。

API 参考

API 参考现在是一个单独的文档。有关更多信息，请参阅 Amazon Comprehend API 参考。

Amazon Comprehend 文档历史记录

下表介绍了此版本的 Amazon Comprehend 的文档。

变更	说明	日期
使用原生文档进行自定义分类器训练	Amazon Comprehend 现在支持使用原生文档进行自定义分类器训练。有关更多信息，请参阅 Amazon Comprehend 中的训练分类模型。	2023 年 4 月 19 日
用于管理自定义模型的飞轮	Amazon Comprehend 现在支持飞轮，可帮助您管理自定义模型的模型版本的训练和跟踪。有关更多信息，请参阅飞轮中的 Amazon Comprehend。	2023 年 2 月 28 日
更新了 IAM 安全主题	更新了 IAM 安全主题以包括联合身份验证。有关更多信息，请参阅 Amazon Comprehend 的身份和访问权限管理。	2022 年 12 月 22 日
使用自定义模型进行推理的单步处理	现在，在运行自定义分类或自定义实体识别之前，Amazon Comprehend 会自动对图像、PDF 或 Word 输入文档执行文本提取。有关更多信息，请参阅 Amazon Comprehend 中的文档处理。	2022 年 12 月 1 日
针对目标情绪的同步 API	Amazon Comprehend 现在支持针对目标情绪的同步 API 和控制台实时分析。目标情绪确定与文档中特定实体相关的情绪。有关更多信息，请参阅	2022 年 9 月 21 日

	Amazon Comprehend 中的目标情绪。	
降低训练识别器的最低注释量	Amazon Comprehend 降低了使用纯文本 CSV 注释文件训练识别器的最低要求。您现在可以构建一个自定义实体识别模型，其中包含最少三个带注释的文档，每种实体类型至少有 25 个注释。有关更多信息，请参阅准备训练数据。	2022 年 8 月 3 日
增加了实时 API 的输入文档大小	对于大多数实时 API，Amazon Comprehend 现在最多支持 100KB 的输入文档。有关更多信息，请参阅指南和配额。	2022 年 7 月 18 日
其他 PII 实体类型	Amazon Comprehend 现在可以检测到其他 PII 实体类型。有关更多信息，请参阅在 Amazon Comprehend 中检测 PII 实体。	2022 年 5 月 20 日
目录重组	重组了 Amazon Comprehend 开发人员指南目录，以便于导航。有关更多信息，请参阅 Amazon Comprehend 是什么。	2022 年 4 月 7 日
目标情绪	Amazon Comprehend 现在支持目标情绪分析，可确定与文档中特定实体相关的情绪。有关更多信息，请参阅 Amazon Comprehend 中的目标情绪。	2022 年 3 月 9 日

新特征	Amazon Comprehend 现在允许您分析图像以进行自定义实体识别。有关更多信息，请参阅在 Amazon Comprehend 中检测自定义实体。	2022 年 2 月 28 日
新特征	现在，你可以在 AWS 账户 之间复制经过训练的自定义模型。有关更多信息，请参阅在 Amazon Comprehend 中的账户之间复制自定义模型。	2022 年 2 月 2 日
新特征	您现在可以使用 AWS Trusted Advisor 查看建议，这些建议可以帮助您优化 Amazon Comprehend 终端节点的成本和安全性。有关更多信息，请参阅将 Trusted Advisor 与 Amazon Comprehend 结合使用。	2021 年 9 月 29 日
新特征	Amazon Comprehend 已为 Comprehend 自定义推出了一套特征，这些特征使您能够创建新的模型版本、持续测试特定测试集以及向新模型终端节点进行实时迁移，从而实现模型的持续改进。	2021 年 9 月 21 日
新特征	Amazon Comprehend 现在允许您分析 PDF 和 Word 文档以进行自定义实体识别。使用 PDF 和 Word 格式，您可以从包含标题、列表和表格的文档中提取信息。	2021 年 9 月 14 日

新特征	Amazon Comprehend 推出了一项新的终端节点概述特征，可为您提供终端节点的全局视图。在终端节点概述页面上，您可以在一个地方查看所有终端节点，以了解您的终端节点使用情况与实际资源使用情况。	2021 年 8 月 24 日
新特征	Amazon Comprehend Medical 现在允许您通过创建接口 VPC 终端节点与虚拟私有云 (VPC) 建立专用连接。有关更多信息，请参阅 VPC 终端节点 (PrivateLink)。	2021 年 6 月 13 日
语言扩展	Amazon Comprehend 为主要语言特征增加了四种语言：豪萨语 (ha)、老挝语 (lo)、马耳他语 (mt) 和奥罗莫语 (om)。有关更多信息，请参阅 Amazon Comprehend 支持的语言。	2021 年 5 月 10 日
新特征	借助 Amazon Comprehend，您现在可以使用客户自主管理型密钥 (CMK) 对自定义模型进行加密。有关更多信息，请参阅 Amazon Comprehend 中的 KMS 加密。	2021 年 3 月 31 日

新特征	您现在可以使用 Amazon S3 对象 Lambda 接入点来配置如何从您的 Amazon S3 存储桶中检索包含个人身份信息 (PII) 的文档。您可以控制对包含 PII 的文档的访问权限，并编辑文档中的 PII。有关更多信息，请参阅使用 Amazon S3 对象 Lambda 接入点获取个人身份信息 (PII)。	2021 年 3 月 18 日
新特征	现在，您可以用个人身份信息（PII）标记文档。Amazon Comprehend 可以分析您的文档中是否存在 PII，并返回已识别的 PII 实体类型的标签，例如姓名、地址、银行账号或电话号码。有关更多信息，请参阅使用 PII 标记文档。	2021 年 3 月 11 日
新特征	借助 Amazon Comprehend，您现在可以在一组文档中检测事件。当您创建异步事件检测作业时，Amazon Comprehend 可以检测支持的财务事件类型。有关更多信息，请参阅检测事件。	2020 年 11 月 24 日
新特征	Amazon Comprehend 现在允许您对自定义实体识别器终端节点使用自动扩缩。借助自动扩缩，您可以自动设置终端节点配置以满足您的容量需求。有关更多信息，请参阅 使用终端节点自动扩缩。	2020 年 9 月 28 日

新特征	要训练自定义分类器或实体识别器，您现在可以提供增强的清单文件，这些文件是由 Amazon G SageMaker round Truth 生成的带标签的数据集。有关这些文件的更多信息以及示例，请参阅多类模式、多标签模式和注释。	2020 年 9 月 22 日
新教程	Amazon Comprehend 现在有一个教程，可引导您完成分析客户评论和可视化分析结果的多服务工作流程。有关更多信息，请参阅教程：分析评论中的见解。	2020 年 9 月 17 日
新特征	借助 Amazon Comprehend，您现在可以检测文本中包含个人身份信息 (PII) 的实体，例如地址、银行账号或电话号码。Amazon Comprehend 可以在您的文本中提供每个 PII 实体的位置，也可以提供一份对 PII 进行了编辑的文本副本。有关更多信息，请参阅检测个人身份信息 (PII)。	2020 年 9 月 17 日
新特征	以前，您最多只能在 12 个自定义实体上训练模型。现在，Amazon Comprehend 允许您一次在多达 25 个自定义实体上训练模型。有关更多信息，请参阅自定义实体识别。	2020 年 8 月 12 日

语言扩展	Amazon Comprehend 为自定义实体识别特征新增了五种语言：德语 (de)、西班牙语 (es)、法语 (fr)、意大利语 (it) 和葡萄牙语 (pt)。有关更多信息，请参阅 Amazon Comprehend 支持的语言。	2020 年 8 月 12 日
新特征	Amazon Comprehend 现在允许您通过创建接口 VPC 终端节点来与虚拟私有云 (VPC) 建立专用连接。有关更多信息，请参阅 VPC 终端节点 (AWS PrivateLink)。	2020 年 8 月 11 日
新特征	借助 Amazon Comprehend，您现在可以通过运行实时分析来快速检测单个文本文档中的自定义实体。有关更多信息，请参阅在 Amazon Comprehend 中实时检测自定义实体。	2020 年 7 月 9 日
增加了新特征	Amazon Comprehend 现在支持文档异步自定义分类中的第二种模式，在将自定义类应用于文档时提供了更大的灵活性。虽然多类模式只能为每个文档关联一个类，但新的多标签模式可以关联多个类。例如，一部电影可以同时归类为科幻小说和动作片。有关更多信息，请参阅自定义分类中的多类和多标签模式。	2019 年 12 月 19 日

增加了新特征	Amazon Comprehend 现在支持对包含非结构化文本的文档进行实时自定义分类。客户可以使用实时自定义分类，根据自己的业务规则同步理解、标记和路由信息。有关更多信息，请参阅使用自定义分类进行实时分析。	2019 年 11 月 25 日
增加了新语言	Amazon Comprehend 为其多项特征增加了六种语言：阿拉伯语 (ar)、印地语 (hi)、日语 (ja)、韩语 (ko)、简体中文 (zh) 和繁体中文 (zh-TW)。这些新语言仅支持确定情绪、检测关键短语和非自定义检测实体操作。有关更多信息，请参阅支持的语言。	2019 年 11 月 6 日
新特征	以前，您只能在单个自定义实体上训练模型。因此，您只能通过实体识别操作搜索该实体。Amazon Comprehend 改变了这一状况，现在您可以同时在多达 12 个自定义实体上训练模型。有关更多信息，请参阅自定义实体识别	2019 年 7 月 9 日
新特征	Amazon Comprehend 现在提供了多类混淆矩阵，以便在训练自定义分类器时提高分析指标的能力。目前仅通过 API 支持。有关更多信息，请参阅在 Amazon Comprehend 中标记资源	2019 年 4 月 5 日

新特征	Amazon Comprehend 为自定义分类器和自定义实体识别器提供标签，这些标签可用作元数据，使您能够以比以往任何时候都更精细的控制级别来组织、筛选和控制对资源的访问。有关更多信息，请参阅在 Amazon Comprehend 中标记资源	2019 年 4 月 3 日
新特征	Amazon S3 已经允许您加密输入文档，而 Amazon Comprehend 将这一功能进一步扩展。通过使用您自己的 KMS 密钥，您不仅可以加密作业的输出结果，还可以加密附加到处理分析作业的计算实例的存储卷上的数据。结果是 end-to-end安全性。有关更多信息，请参阅 Amazon Comprehend 中的 KMS 加密	2019 年 3 月 28 日
新特征	自定义实体识别扩展了 Amazon Comprehend 的功能，它使您能够识别不支持作为预设通用实体类型之一的新实体类型。这意味着您可以分析文档并提取符合您特定需求的实体，例如产品代码或业务特定实体。有关更多信息，请参阅自定义实体识别	2018 年 11 月 16 日
新特征	您可以使用 Amazon Comprehend 构建自己的自定义分类模型，将文档分配给一个类或一个类别。有关详细信息，请参阅文档分类。	2018 年 11 月 15 日

区域扩展	Amazon Comprehend 现已在欧洲地区（法兰克福）(eu-central-1) 提供服务。	2018 年 10 月 10 日
语言扩展	除了英语和西班牙语之外，Amazon Comprehend 现在还可以检查法语、德语、意大利语和葡萄牙语的文档。有关更多信息，请参阅 Amazon Comprehend 支持的语言。	2018 年 10 月 10 日
区域扩展	Amazon Comprehend 现已在亚太地区（悉尼）(ap-southeast-2) 提供服务。	2018 年 8 月 15 日
新特征	Amazon Comprehend 现在可以解析文档，以发现文档的语法和每个单词的词性。有关更多信息，请参阅语法。	2018 年 7 月 17 日
新特征	Amazon Comprehend 现在支持异步批处理语言、关键短语、实体和情绪检测。有关更多信息，请参阅异步批处理。	2018 年 6 月 27 日
新指南	本指南是 Amazon Comprehend 开发人员指南的首次发布。	2017 年 11 月 29 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。