Amazon Comprehend

Developer Guide

Amazon Comprehend: Developer Guide

Table of Contents

What Is Amazon Comprehend?

Amazon Comprehend uses natural language processing (NLP) to extract insights about the content of documents. Amazon Comprehend processes any text file in UTF-8 format. It develops insights by recognizing the entities, key phrases, language, sentiments, and other common elements in a document. Use Amazon Comprehend to create new products based on understanding the structure of documents. For example, using Amazon Comprehend you can search social networking feeds for mentions of products or scan an entire document repository for key phrases.

You work with one document at a time to detect entities, key phrases, languages, and sentiments. Each document is processed separately. Some of the insights that Amazon Comprehend develops about a document include:

- **Entities** – Amazon Comprehend returns a list of entities, such as people, places, and locations, identified in a document. For more information, see Detecting Entities (p. 3).
- **Key phrases** – Amazon Comprehend extracts key phrases that appear in a document. For example, a document about a basketball game might return the names of the teams, the name of the venue, and the final score. For more information, see Detecting Key Phrases (p. 4).
- **Language** – Amazon Comprehend identifies the dominant language in a document. Amazon Comprehend can identify 100 languages. For more information, see Detecting the Primary Language (p. 4).
- **Sentiment** – Amazon Comprehend determines the emotional sentiment of a document. Sentiment can be positive, neutral, negative, or mixed. For more information, see Detecting Sentiments (p. 6).

Topic Modelling

You can also use Amazon Comprehend to examine a corpus of documents to find the common themes contained within the corpus. Amazon Comprehend examines the documents in the corpus and then returns the most prominent topics and the documents that are associated with each topic.

Topic modeling is a asynchronous process, you submit a set of documents for processing and then later get the results when processing is complete. Amazon Comprehend does topic modeling on large document sets, for best results you should include at least 1,000 documents when you submit a topic modeling job. For more information, see Topic Modeling (p. 9).

Examples

The following examples show how you might use the Amazon Comprehend operations in your applications.

Example 1: Find documents about a subject

Find the documents about a particular subject using Amazon Comprehend topic modeling. Scan a set of documents to determine the topics discussed, and to find the documents associated with each topic. You can specify the number of topics that Amazon Comprehend should return from the document set.

Example 2: Find out how customers feel about your products

If your company publishes a catalog, let Amazon Comprehend tell you what customers think of your products. Send each customer comment to the `DetectSentiment` operation and it will tell you whether customers feel positive, negative, neutral, or mixed about a product.

Example 3: Discover what matters to your customers

Use Amazon Comprehend topic modeling to discover the topics that your customers are talking about on your forums and message boards, then use entity detection to determine the people, places, and things that they associate with the topic. Finally, use sentiment analysis to determine how your customers feel about a topic.

Benefits

Some of the benefits of using Amazon Comprehend include:

- **Integrate powerful natural language processing into your apps**—Amazon Comprehend removes the complexity of building text analysis capabilities into your applications by making powerful and accurate natural language processing available with a simple API. You don't need textual analysis expertise to take advantage of the insights that Amazon Comprehend produces.
- **Deep learning based natural language processing**—Amazon Comprehend uses deep learning technology to accurately analyze text. Our models are constantly trained with new data across multiple domains to improve accuracy.
- **Scalable natural language processing**—Amazon Comprehend enables you to analyze millions of documents so that you can discover the insights that they contain.
- **Integrate with other AWS services**—Amazon Comprehend is designed to work seamlessly with other AWS services like Amazon S3 and AWS Lambda. Store your documents in Amazon S3, or analyze real-time data with Kinesis Data Firehose. Support for AWS Identity and Access Management (IAM) makes it easy to securely control access to Amazon Comprehend operations. Using IAM, you can create and manage AWS users and groups to grant the appropriate access to your developers and end users.
- **Low cost**—With Amazon Comprehend, you only pay for the documents that you analyze. There are no minimum fees or upfront commitments.

Are You a First-time User of Amazon Comprehend?

If you are a first-time user of Amazon Comprehend, we recommend that you read the following sections in order:

1. **How It Works (p. 3)** – This section introduces Amazon Comprehend concepts.
2. **Getting Started with Amazon Comprehend (p. 13)** – In this section, you set up your account and test Amazon Comprehend.
3. **API Reference (p. 57)** – In this section you'll find reference documentation for Amazon Comprehend operations.

How It Works

Amazon Comprehend uses a pre-trained model to examine a document or set of documents to gather insights about the document set. The model is continuously trained on a large body of text so that there is no need for you to provide training data. Except for the DetectDominantLanguage (p. 72) operation, Amazon Comprehend can examine documents in these languages:

- English
- Spanish

Amazon Comprehend provides the following operations:

- DetectDominantLanguage (p. 72) – to detect the dominant language in a document. Amazon Comprehend can detect 101 different languages.
- DetectEntities (p. 75) – to detect the entities, such as persons or places, in the document.
- DetectKeyPhrases (p. 78) – to detect key noun phrases that are most indicative of the content.
- DetectSentiment (p. 81) – to detect the emotional sentiment, positive, negative, mixed, or neutral, of a document.
- StartTopicsDetectionJob (p. 87) – to detect topics in a set of documents.

All of the operations work on a single document. You can send up to 25 documents in a single batch using the batch operations. When you send a batch, Amazon Comprehend returns a list of responses, one for each document that you sent.

You can use the following batch operations:

- BatchDetectDominantLanguage (p. 58)
- BatchDetectEntities (p. 61)
- BatchDetectKeyPhrases (p. 64)
- BatchDetectSentiment (p. 67)

Topic Modeling

The `StartTopicsDetectionJob` operation starts an asynchronous operation that processes a set of documents stored in an Amazon S3 bucket to determine the topics in the document set. Amazon Comprehend trains its topic model on the corpus of documents that you supply, and then assigns documents and topics based on the insights that it discovered.

Detecting Entities

Use the DetectEntities (p. 75) and BatchDetectEntities (p. 61) operations to detect entities in a document. A *entity* is a textual reference to the unique name of a real-world object such as people, places, and commercial items, and to precise references to measures such as dates and quantities.

For example, in the text "John moved to 1313 Mockingbird Lane in 2012," "John" might be recognized as a `PERSON`, "1313 Mockingbird Lane" might be recognized as a `LOCATION`, and "2012" might be recognized as a `DATE`.

Each entity also has a score that indicates the level of confidence that Amazon Comprehend has that it correctly detected the entity type. You can filter out the entities with lower scores to reduce the risk of using incorrect detections.

The following table lists the entity types.

Type	Description
COMMERCIAL_ITEM	A branded product
DATE	A full date (for example, 11/25/2017), day (Tuesday), month (May), or time (8:30 a.m.)
EVENT	An event, such as a festival, concert, election, etc.
LOCATION	A specific location, such as a country, city, lake, building, etc.
ORGANIZATION	Large organizations, such as a government, company, religion, sports team, etc.
OTHER	Entities that don't fit into any of the other entity categories
PERSON	Individuals, groups of people, nicknames, fictional characters
QUANTITY	A quantified amount, such as currency, percentages, numbers, bytes, etc.
TITLE	An official name given to any creation or creative work, such as movies, books, songs, etc.

Detecting Key Phrases

You can use Amazon Comprehend operations to detect entities and key phrases in your document.

The DetectKeyPhrases (p. 78) and BatchDetectKeyPhrases (p. 64) operations detect the key phrases in a document. A *key phrase* is a string containing a noun phrase that describes a particular thing. It generally consists of a noun and the modifiers that distinguish it. For example, "day" is a noun; "a beautiful day" is a noun phrase that includes an article ("a") and an adjective ("beautiful"). Each key phrase includes a score that indicates the level of confidence that Amazon Comprehend has that the string is a noun phrase. You can use the score to determine if the detection has high enough confidence for your application.

Detecting the Primary Language

You can use the DetectDominantLanguage (p. 72) and BatchDetectDominantLanguage (p. 58) operations to examine text to determine the primary language. It identifies the language using identifiers from RFC 5646 — if there is a 2-letter ISO 639-1 identifier, with a regional subtag if necessary, it uses that. Otherwise, it uses the ISO 639-2 3-letter code. For more information about RFC 5646, see Tags for Identifying Languages on the *IETF Tools* web site.

The response includes a score that indicates the confidence level that Amazon Comprehend has that a particular language is the dominant language in the document. Each score is independent of the other scores — it does not indicate that a language makes up a particular percentage of a document.

If a long document, like a book, is written in multiple languages, you can break the long document into smaller pieces and run the `DetectDominantLanguage` operation on the individual pieces. You can then aggregate the results to determine the percentage of each language in the longer document.

Amazon Comprehend can detect the following languages.

Code	Language	Code	Language	Code	Language
af	Afrikaans	hy	Armenian	ps	Pushto
am	Amharic	ilo	Iloko	qu	Quechua
ar	Arabic	id	Indonesian	ro	Romanian
as	Assamese	is	Icelandic	ru	Russian
az	Azerbaijani	it	Italian	sa	Sanskrit
ba	Bashkir	jv	Javanese	si	Sinhala
be	Belarusian	ja	Japanese	sk	Slovak
bn	Bengali	kn	Kannada	sl	Slovenian
bs	Bosnian	ka	Georgian	sd	Sindhi
bg	Bulgarian	kk	Kazakh	so	Somali
ca	Catalan	km	Central Khmer	es	Spanish
ceb	Cebuano	ky	Kirghiz	sq	Albanian
cs	Czech	ko	Korean	sr	Serbian
cv	Chuvash	ku	Kurdish	su	Sundanese
cy	Welsh	la	Latin	sw	Swahili
da	Danish	lv	Latvian	sv	Swedish
de	German	lt	Lithuanian	ta	Tamil
el	Greek	lb	Luxembourgish	tt	Tatar
en	English	ml	Malayalam	te	Telugu
eo	Esperanto	mr	Marathi	tg	Tajik
et	Estonian	mk	Macedonian	tl	Tagalog
eu	Basque	mg	Malagasy	th	Thai
fa	Persian	mn	Mongolian	tk	Turkmen
fi	Finnish	ms	Malay	tr	Turkish
fr	French	my	Burmese	ug	Uighur
gd	Scottish Gaelic	ne	Nepali	uk	Ukrainian
ga	Irish	new	Newari	ur	Urdu
gl	Galician	nl	Dutch	uz	Uzbek
gu	Gujarati	no	Norwegian	vi	Vietnamese
ht	Haitian	or	Oriya	yi	Yiddish

Code	Language	Code	Language	Code	Language
he	Hebrew	pa	Punjabi	yo	Yoruba
hi	Hindi	pl	Polish	zh	Chinese (Simplified)
hr	Croatian	pt	Portuguese	zh-TW	Chinese (Traditional)
hu	Hungarian				

Detecting Sentiments

Use Amazon Comprehend to determine the sentiment of a document. You can determine if the sentiment is positive, negative, neutral, or mixed. For example, you can use sentiment analysis to determine the sentiments of comments on a blog posting to determine if your readers liked the post.

The DetectSentiment (p. 81) and BatchDetectSentiment (p. 67) operations return the most likely sentiment for the text as well as the scores for each of the sentiments. The score represents the likelihood that the sentiment was correctly detected. For example, in the first example below it is 95 percent likely that the text has a `Positive` sentiment. There is a less than 1 percent likelihood that the text has a `Negative` sentiment. You can use the `SentimentScore` to determine if the accuracy of the detection meets the needs of your application.

The following examples show the input text and output from the `DetectSentiment` operation. The examples are formatted for Unix, Linux, and macOS. For Windows, replace the backslash (\) Unix continuation character at the end of each line with a caret (^).

Positive sentiment:

```
aws comprehend detect-sentiment \
    --endpoint-url endpoint \
    --region region \
    --language-code "en" \
    --text "I feel great this morning."
{
    "SentimentScore": {
        "Mixed": 0.008924853056669235,
        "Positive": 0.9584325551986694,
        "Neutral": 0.026196759194135666,
        "Negative": 0.006445900071412325
    },
    "Sentiment": "POSITIVE",
}
```

Negative sentiment:

```
aws comprehend detect-sentiment \
    --endpoint-url endpoint \
    --region region \
    --language-code "en" \
    --text "This view is horrible."
{
    "SentimentScore": {
        "Mixed": 0.024891886860132217,
        "Positive": 0.050186172127723694,
        "Neutral": 0.1579618602991104,
        "Negative": 0.7669601440429688
    },
```

```
    "Sentiment": "NEGATIVE",
}
```

Neutral sentiment:

```
aws comprehend detect-sentiment \
    --endpoint-url endpoint \
    --region region \
    --language-code "en" \
    --text "Childhood is the time to play"
{
    "SentimentScore": {
        "Mixed": 0.020051531493663788,
        "Positive": 0.35540205240249634,
        "Neutral": 0.4203023612499237,
        "Negative": 0.20424406230449677
    },
    "Sentiment": "NEUTRAL",
}
```

Mixed sentiment:

```
aws comprehend detect-sentiment \
    --endpoint-url endpoint \
    --region region \
    --language-code "en" \
    --text "I love Seattle but the winter is too cold for me."
{
    "SentimentScore": {
        "Mixed": 0.5408428907394409,
        "Positive": 0.25948983430862427,
        "Neutral": 0.06789177656173706,
        "Negative": 0.13177546858787537
    },
    "Sentiment": "MIXED",
}
```

Batch Processing Documents

When you have multiple documents that you want to process, you can use batch operations to send more than one document to Amazon Comprehend at a time. You can send up to 25 documents in each batch. Amazon Comprehend sends back a list of responses, one for each document in the batch.

Using the batch operations is identical to calling the single document APIs for each of the documents in the request. Calling the batch APIs can result in better performance for your applications.

The input to the batch APIs is a JSON structure containing the documents to process. For all operations except `BatchDetectDominantLanguage`, you must set the input language. You can set only one input language for each batch. For example, the following is the input to the `BatchDetectEntities` operation. It contains two documents and is in English.

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

```
}
```

The response from a batch operation contains two lists, the `ResultList` and the `ErrorList`. The `ResultList` contains one record for each document that was successfully processed. The result for each document in the batch is identical to the result you would get if you ran a single document operation on the document. The results for each document are assigned an index based on the order of the documents in the input file. The response from the `BatchDetectEntities` operation is:

```
{
   "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": []
}
```

When an error occurs in the batch request the response contains an `ErrorList` that identifies the documents that contained an error. The document is identified by its index in the input list. For example, the following input to the `BatchDetectLanguage` operation contains a document that cannot be processed:

```
{
   "TextList": [
     "hello friend",
     "$$$$$$",
     "hola amigo"
```

```
    ]
}
```

The response from Amazon Comprehend includes an error list that identifies the document that contained an error:

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

Topic Modeling

You can use Amazon Comprehend to examine the content of a collection of documents to determine common themes. Amazon Comprehend can examine documents in English or Spanish. For example, you can give Amazon Comprehend a collection of news articles, and it will determine the subjects, such as sports, politics, or entertainment. The text in the documents doesn't need to be annotated.

Amazon Comprehend uses a Latent Dirichlet Allocation-based learning model to determine the topics in a set of documents. It examines each document to determine the context and meaning of a word. The set of words that frequently belong to the same context across the entire document set make up a topic.

A word is associated to a topic in a document based on how prevalent that topic is in a document and how much affinity the topic has to the word. The same word can be associated with different topics in different documents based on the topic distribution in a particular document.

For example, the word "glucose" in an article that talks predominantly about sports can be assigned to the topic "sports," while the same word in an article about "medicine" will be assigned to the topic "medicine."

Each word associated with a topic is given a weight that indicates how much the word helps define the topic. The weight is an indication of how many times the word occurs in the topic compared to other words in the topic, across the entire document set.

For the most accurate results you should provide Amazon Comprehend with the largest possible corpus to work with. For best results:

- You should use at least 1,000 documents in each topic modeling job.
- Each document should be at least 3 sentences long.
- If a document consists of mostly numeric data, you should remove it from the corpus.

Topic modeling is an asynchronous process. You submit your list of documents to Amazon Comprehend from an Amazon S3 bucket using the StartTopicsDetectionJob (p. 87) operation. The response is sent to an Amazon S3 bucket. You can configure both the input and output buckets. Get a list of the topic modeling jobs that you have submitted using the ListTopicsDetectionJobs (p. 84) operation and view information about a job using the DescribeTopicsDetectionJob (p. 70) operation. Content delivered to Amazon S3 buckets might contain customer content. For more information about removing sensitive data, see How Do I Empty an S3 Bucket? or How Do I Delete an S3 Bucket?.

Documents must be in UTF-8 formatted text files. You can submit your documents two ways. The following table shows the options.

Format	Description
One document per file	Each file contains one input document. This is best for collections of large documents.
One document per line	The input is a single file. Each line in the file is considered a document. This is best for short documents, such as social media postings.

For more information, see the InputDataConfig (p. 99) data type.

After Amazon Comprehend processes your document collection, it returns a compressed archive containing two files, `topic-terms.csv` and `doc-topics.csv`. For more information about the output file, see OutputDataConfig (p. 101).

The first output file, `topic-terms.csv`, is a list of topics in the collection. For each topic, the list includes, by default, the top terms by topic according to their weight. For example, if you give Amazon Comprehend a collection of newspaper articles, it might return the following to describe the first two topics in the collection:

Topic	Term	Weight
000	team	0.118533
000	game	0.106072
000	player	0.031625
000	season	0.023633
000	play	0.021118
000	yard	0.024454
000	coach	0.016012
000	games	0.016191

Topic	Term	Weight
000	football	0.015049
000	quarterback	0.014239
001	cup	0.205236
001	food	0.040686
001	minutes	0.036062
001	add	0.029697
001	tablespoon	0.028789
001	oil	0.021254
001	pepper	0.022205
001	teaspoon	0.020040
001	wine	0.016588
001	sugar	0.015101

The weights represent a probability distribution over the words in a given topic. Since Amazon Comprehend returns only the top 10 words for each topic the weights won't sum to 1.0. In the rare cases where there are less than 10 words in a topic, the weights will sum to 1.0.

The words are sorted by their discriminative power by looking at their occurrence across all topics. Typically this is the same as their weight, but in some cases, such as the words "play" and "yard" in the table, this results in an order that is not the same as the weight.

You can specify the number of topics to return. For example, if you ask Amazon Comprehend to return 25 topics, it returns the 25 most prominent topics in the collection. Amazon Comprehend can detect up to 100 topics in a collection. Choose the number of topics based on your knowledge of the domain. It may take some experimentation to arrive at the correct number.

The second file, `doc-topics.csv`, lists the documents associated with a topic and the proportion of the document that is concerned with the topic. If you specified `ONE_DOC_PER_FILE` the document is identified by the file name. If you specified `ONE_DOC_PER_LINE` the document is identified by the file name and the 0-indexed line number within the file. For example, Amazon Comprehend might return the following for a collection of documents submitted with one document per file:

Document	Topic	Proportion
sample-doc1	000	0.999330137
sample-doc2	000	0.998532187
sample-doc3	000	0.998384574
...		
sample-docN	000	3.57E-04

Amazon Comprehend utilizes information from the *Lemmatization Lists Dataset by MBM*, which is made available here under the Open Database License (ODbL) v1.0.

Role-based Data Access

To use the Amazon Comprehend topic modeling operations, you must grant Amazon Comprehend access to the Amazon S3 bucket that contains your document collection. For more information, see Role-Based Permissions Required for Topic Detection (p. 52).

Getting Started with Amazon Comprehend

To get started using Amazon Comprehend, set up an AWS account and create an AWS Identity and Access Management (IAM) user. To use the AWS Command Line Interface (AWS CLI), download and configure it.

Topics

Step 1: Set Up an AWS Account and Create an Administrator User

Before you use Amazon Comprehend for the first time, complete the following tasks:

1. Sign Up for AWS (p. 13)
2. Create an IAM User (p. 14)

Sign Up for AWS

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

With Amazon Comprehend, you pay only for the resources that you use. If you are a new AWS customer, you can get started with Amazon Comprehend for free. For more information, see AWS Free Usage Tier.

If you already have an AWS account, skip to the next section.

To create an AWS account

1. Open https://aws.amazon.com/, and then choose **Create an AWS Account**.

   Note
   This might be unavailable in your browser if you previously signed into the AWS Management Console. In that case, choose **Sign in to a different account**, and then choose **Create a new AWS account**.

2. Follow the online instructions.

   Part of the sign-up procedure involves receiving a phone call and entering a PIN using the phone keypad.

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

Create an IAM User

Services in AWS, such as Amazon Comprehend, require that you provide credentials when you access them. This allows the service to determine whether you have permissions to access the service's resources.

We strongly recommend that you access AWS using AWS Identity and Access Management (IAM), not the credentials for your AWS account. To use IAM to access AWS, create an IAM user, add the user to an IAM group with administrative permissions, and then grant administrative permissions to the IAM user. You can then access AWS using a special URL and the IAM user's credentials.

The Getting Started exercises in this guide assume that you have a user with administrator privileges, `adminuser`.

To create an administrator user and sign in to the console

1. Create an administrator user called `adminuser` in your AWS account. For instructions, see Creating Your First IAM User and Administrators Group in the *IAM User Guide*.
2. Sign in to the AWS Management Console using a special URL. For more information, see How Users Sign In to Your Account in the *IAM User Guide*.

For more information about IAM, see the following:

- AWS Identity and Access Management (IAM)
- Getting Started
- IAM User Guide

Next Step

Step 2: Set Up the AWS Command Line Interface (AWS CLI) (p. 14)

Step 2: Set Up the AWS Command Line Interface (AWS CLI)

You don't need the AWS CLI to perform the steps in the Getting Started exercises. However, some of the other exercises in this guide do require it. If you prefer, you can skip this step and go to Step 3: Getting Started Using the Amazon Comprehend Console (p. 15), and set up the AWS CLI later.

To set up the AWS CLI

1. Download and configure the AWS CLI. For instructions, see the following topics in the *AWS Command Line Interface User Guide*:
   - Getting Set Up with the AWS Command Line Interface
   - Configuring the AWS Command Line Interface
2. In the AWS CLI config file, add a named profile for the administrator user:.

```
[profile adminuser]
```

```
aws_access_key_id = adminuser access key ID
aws_secret_access_key = adminuser secret access key
region = aws-region
```

You use this profile when executing the AWS CLI commands. For more information about named profiles, see Named Profiles in the *AWS Command Line Interface User Guide*. For a list of AWS Regions, see Regions and Endpoints in the *Amazon Web Services General Reference*.

3. Verify the setup by typing the following help command at the command prompt:

```
aws help
```

Next Step

Step 3: Getting Started Using the Amazon Comprehend Console (p. 15)

Step 3: Getting Started Using the Amazon Comprehend Console

The easiest way to get started using Amazon Comprehend is to use the console to analyze a short text file. If you haven't reviewed the concepts and terminology in How It Works (p. 3), we recommend that you do that before proceeding.

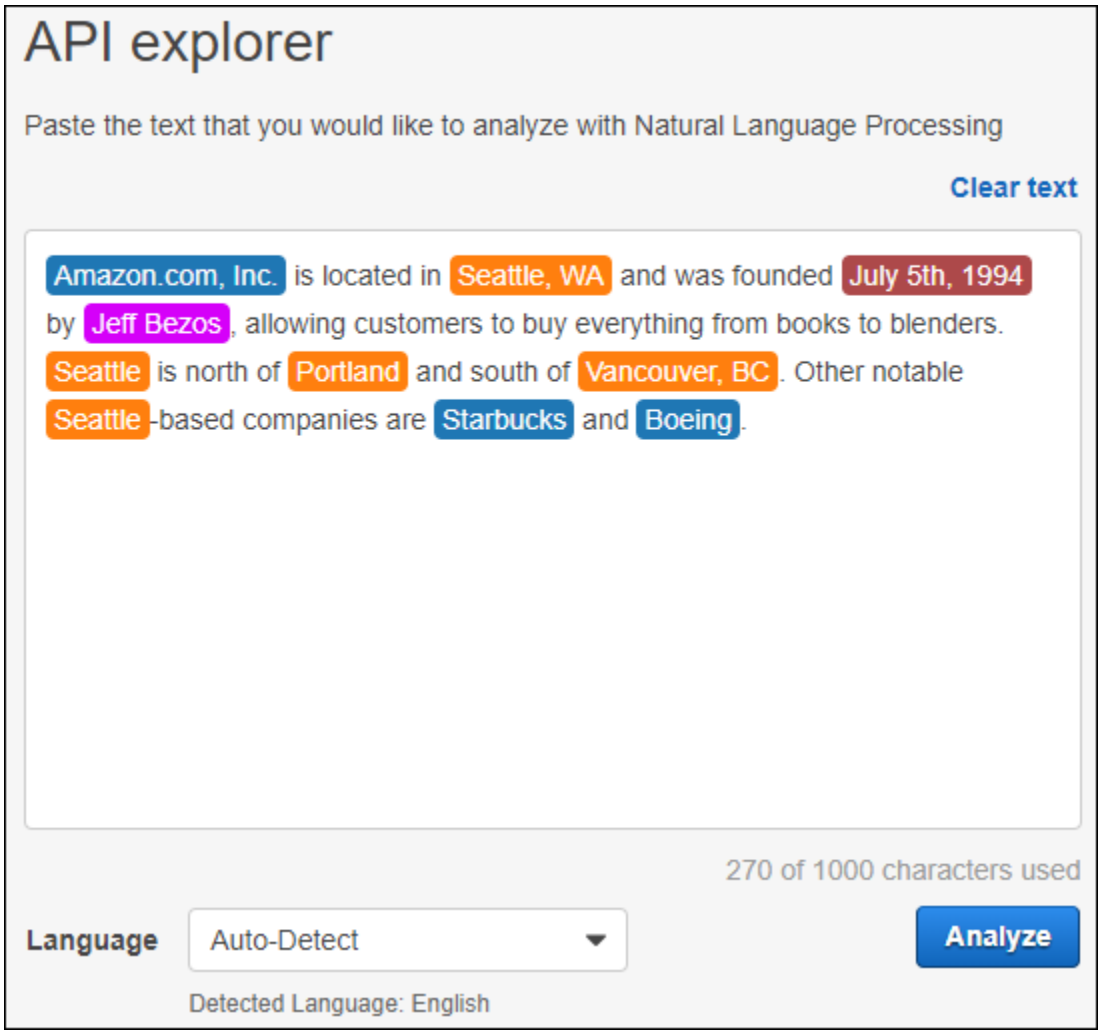
Topics

Analyzing Documents Using the Console

The Amazon Comprehend console enables you to analyze the contents of documents up to 1,000 characters long. The results are shown in the console so that you can review the analysis.

To start analyzing documents, sign in to the AWS Management Console and open the Amazon Comprehend console.

The console displays sample text and the analysis of that text:



You can replace the sample text with your own text in English or Spanish and then choose **Analyze** to get an analysis of your text.

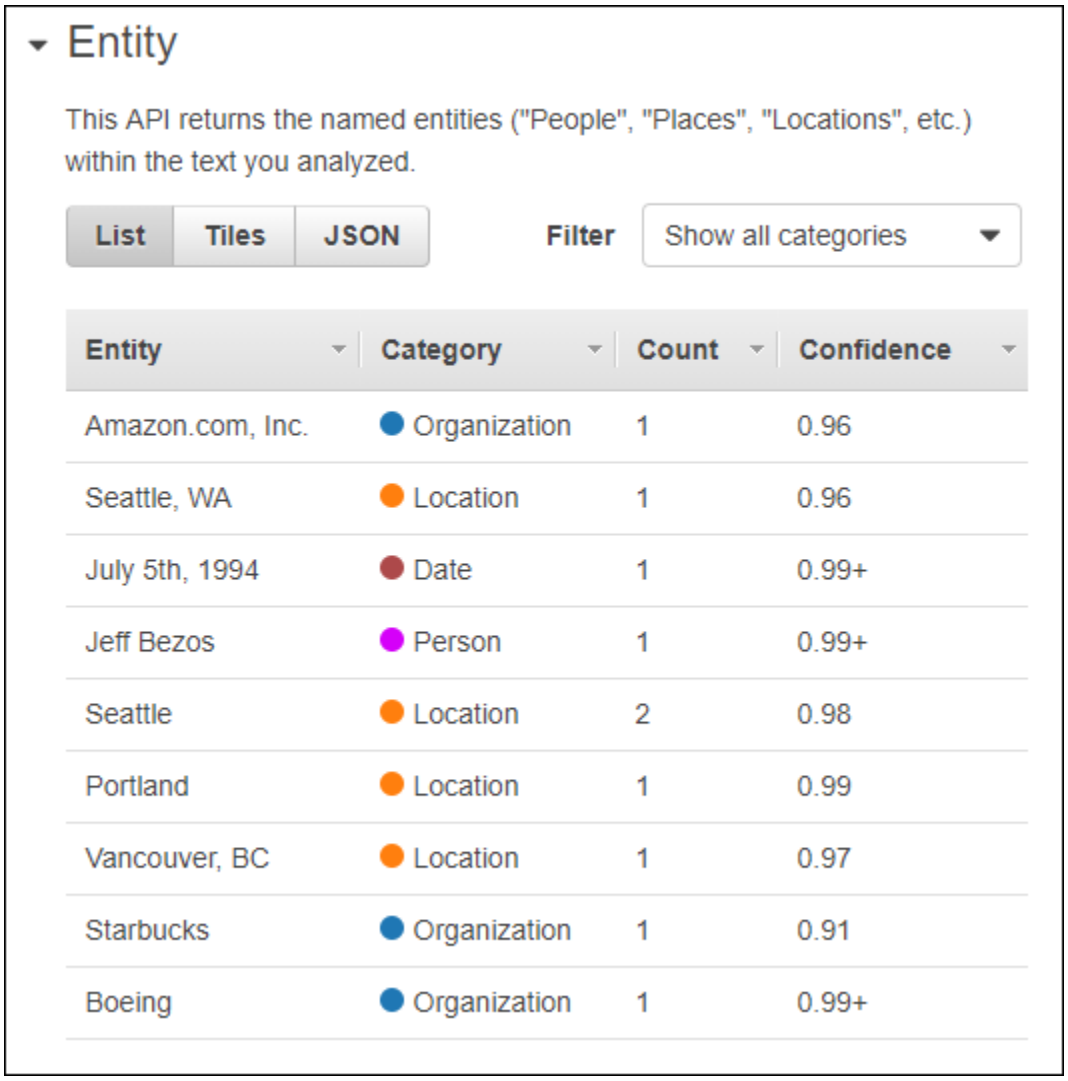
The text is color-coded to indicate the entity type of significant words:

- Orange tags identify locations.
- Brown tags identify dates.
- Magenta tags identify persons.
- Blue tags identify organizations.
- Black tags identify other entities that don't fit into any of the other entity categories.

For more information, see Detecting Entities (p. 3)

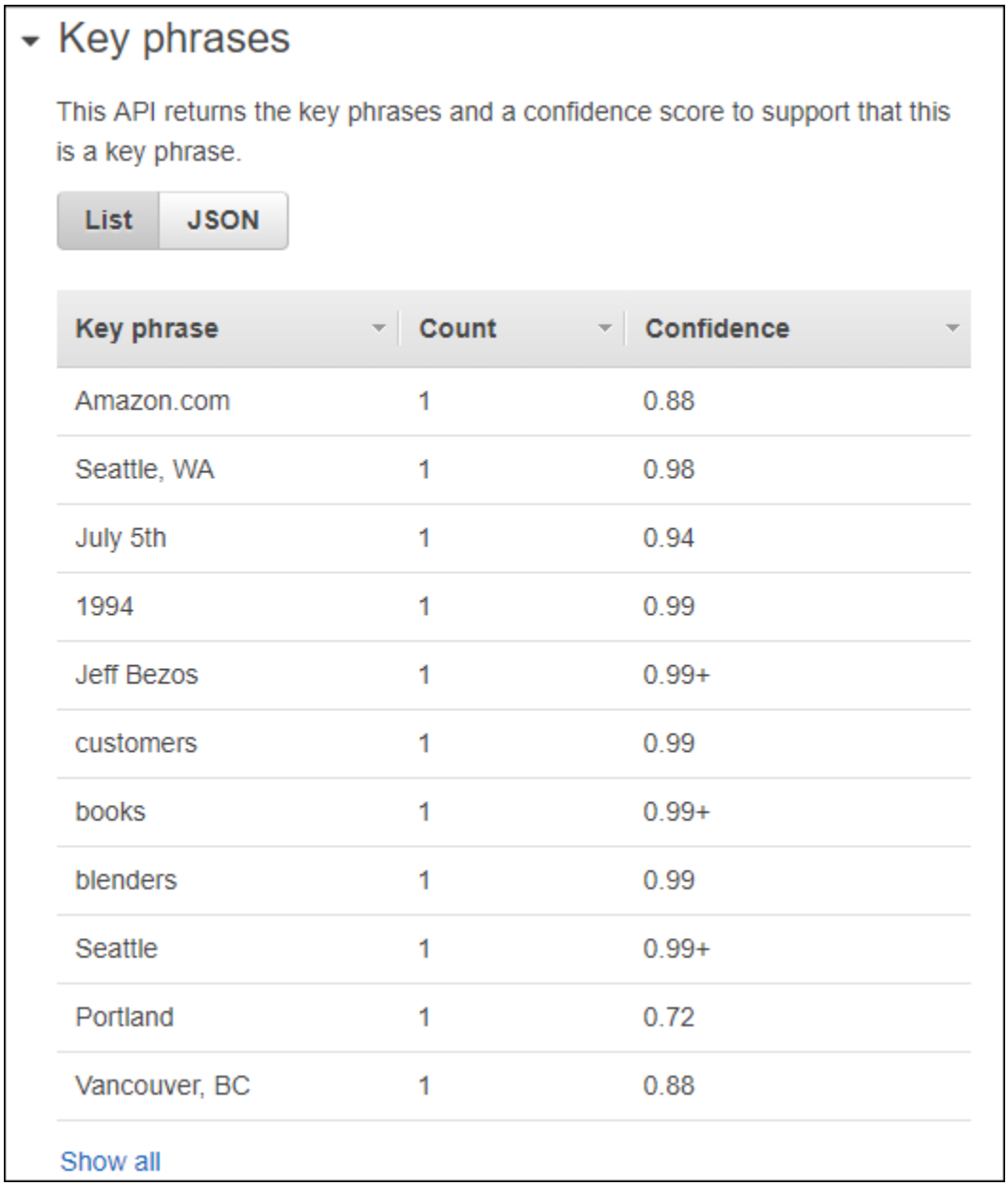
On the right side of the console, the **Analysis** pane shows more information about the text.

The **Entity** section displays cards for the entities found in the text:



Each card shows the text and its entity type. To see a list of all of the entities in the text, choose **List**. For a JSON structure of the results, choose **JSON**. The JSON structure is the same as the structure returned by the DetectEntities (p. 75) operation.

The **Key phrases** section of the **Analysis** pane lists key noun phrases that Amazon Comprehend detected in the input text. For the sample input text, the **Key phrases** section looks like this:



Key phrases

This API returns the key phrases and a confidence score to support that this is a key phrase.

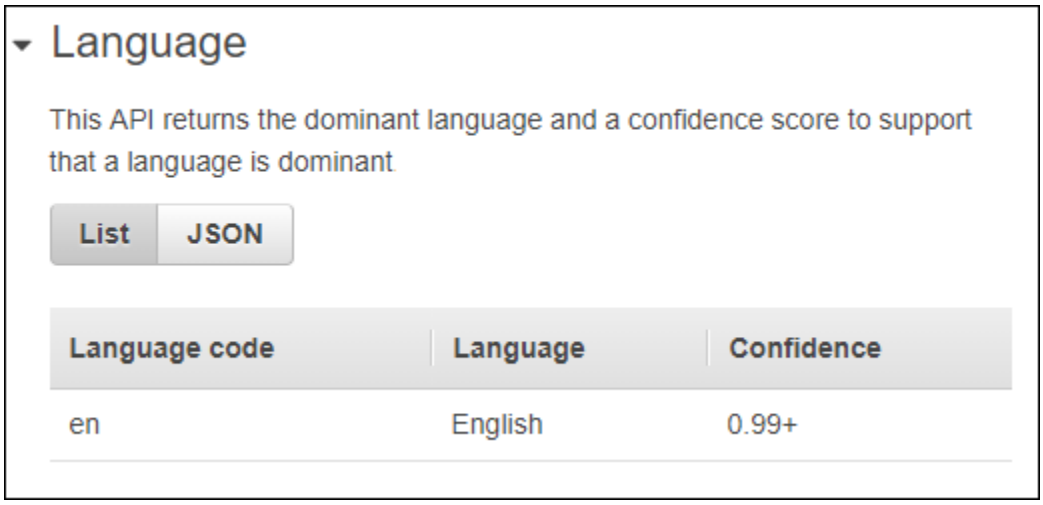
List JSON

Key phrase	Count	Confidence
Amazon.com	1	0.88
Seattle, WA	1	0.98
July 5th	1	0.94
1994	1	0.99
Jeff Bezos	1	0.99+
customers	1	0.99
books	1	0.99+
blenders	1	0.99
Seattle	1	0.99+
Portland	1	0.72
Vancouver, BC	1	0.88

Show all

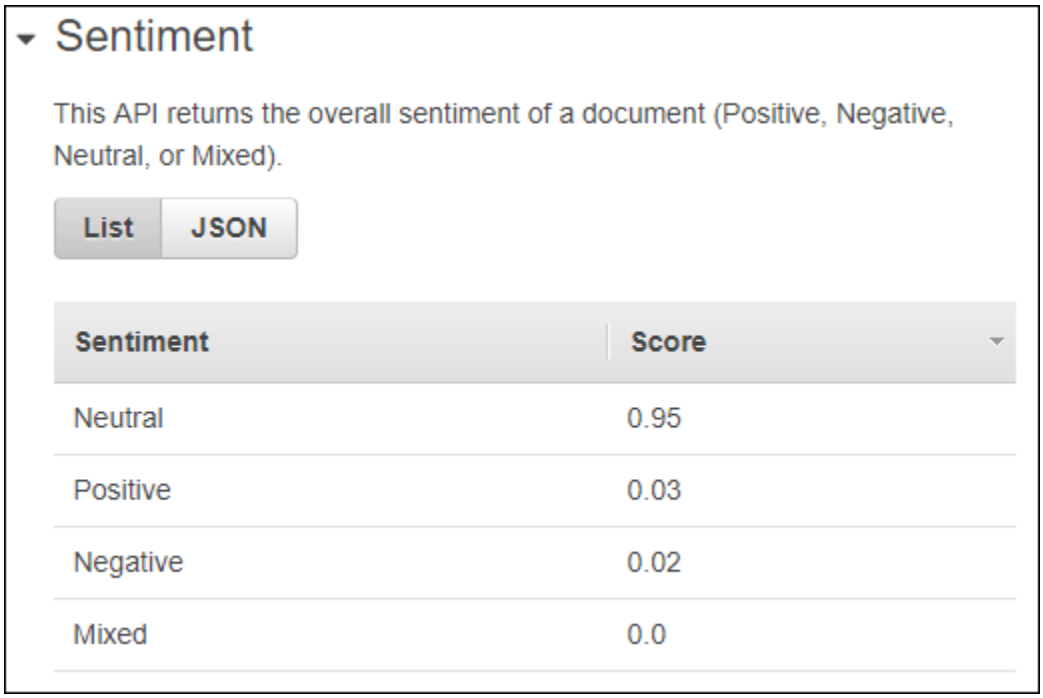
For an alternative view of the results, choose **List** or **JSON** . The JSON structure is the same as the one returned by the DetectKeyPhrases (p. 78) operation.

The **Language** section shows the dominant language for the sample text and the Confidence score. The Confidence score represents the level of confidence that Amazon Comprehend has that it's detected the dominant language correctly. Amazon Comprehend can recognize 100 languages. For more information, see Detecting the Primary Language (p. 4).



As with the other sections, you can choose **List** or **JSON** to get another view of the results. The JSON structure is the same as the one returned by the DetectDominantLanguage (p. 72) operation.

The **Sentiment** section of the **Analysis** pane shows the overall emotional sentiment of the text.



The score represents the confidence that Amazon Comprehend has that it has correctly detected the emotional sentiment expressed in the text. Sentiment can be rated positive, neutral, mixed, or negative.

Creating a Topic Modeling Job Using the Console

You can use the Amazon Comprehend console to create and manage asynchronous topic detection jobs.

To create a topic detection job

1. Sign in to the AWS Management Console and open the Amazon Comprehend console.

2. From the left menu, choose **Topic Modeling** and then choose **Create**.
3. Choose the data source to use. You can use either sample data or you can analyze your own data stored in an Amazon S3 bucket. If you use the sample dataset, the topic modeling job analyzes text from this collection of articles: https://s3.amazonaws.com/public-sample-attributions/Attributions.txt
4. If you chose to use your own data, provide the following information in the **Choose input data** section:
   - **S3 data location** – An Amazon S3 data bucket that contains the documents to analyze. You can choose the folder icon to browse to the location of your data. The bucket must be in the same region as the API that you are calling.
   - **Input format** – Choose whether in input data is contained in one document per file, or if there is one document per line in a file.
   - **Number of topics** – The number of topics to return.
   - **Job name** – A name that identifies the particular job.
5. In the **Choose output location** section, provide the following:
   - **S3 data location** – an Amazon S3 data bucket where the results of the analysis will be written. The bucket must be in the same region as the API that your are calling.
6. In the **Choose an IAM role** section, either select an existing IAM role or create a new one.
   - **Choose an existing IAM role** – Choose this option if you already have an IAM role with permissions to access the input and output Amazon S3 buckets.
   - **Create a new IAM role** – Choose this option when you want to create a new IAM role with the proper permissions for Amazon Comprehend to access the input and output buckets. For more information about the permissions given to the IAM role, see Role-Based Permissions Required for Topic Detection (p. 52).
7. When you have finished filling out the form, choose **Create job** to create and start the topic detection job.

Select input data

Please select the topic modeling data you would like to analyze. Our text a
1,000 documents of at least 100 words each, but can be applied to docum
millions of documents.

- (•) Topic modeling sample data
- ( ) My data (S3)

S3 data location s3://public-sample-us-west-2

Input format One document per line

Number of topics 10

Job Name <job name>

Any characters; length between 1-256

Select output location

Select the preferred output format for your analysis. S3 data output locatio

S3 data location s3://<output bucket>

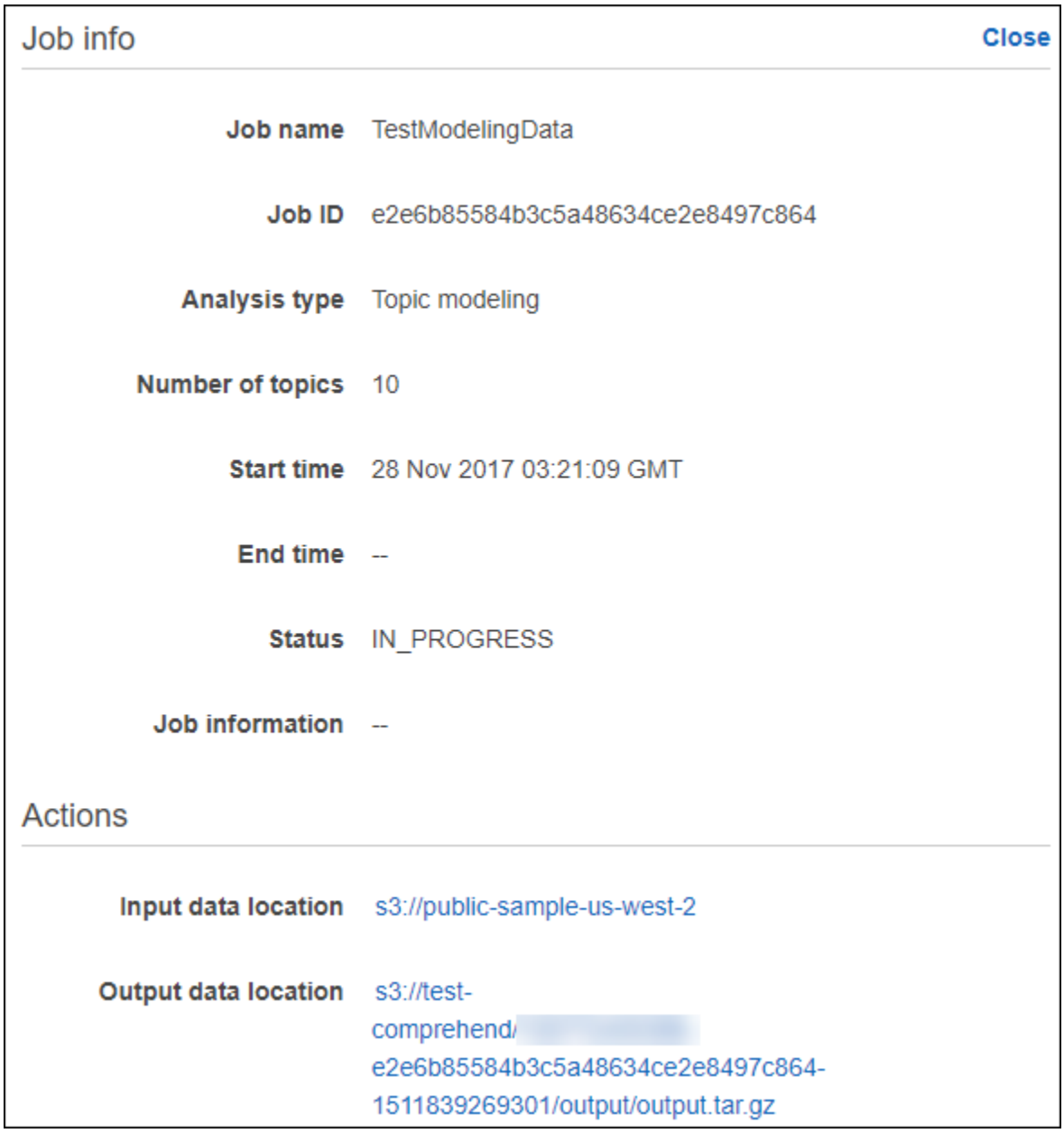
Select an IAM role

The topic modeling job will use the IAM role to access your Amazon S3 inp

- ( ) Select an existing IAM role
- (•) Create a new IAM role

Name suffix <IAM role name>

The new job appears in the job list with the status field showing the status of the job. The field can be `IN_PROGRESS` for a job that is processing, `COMPLETED` for a job that has finished successfully, and `FAILED` for a job that has an error. You can click on a job to get more information about the job, including any error messages.



Next Step

Step 4: Getting Started Using the Amazon Comprehend API (p. 23)

Step 4: Getting Started Using the Amazon Comprehend API

The following examples demonstrate how to use Amazon Comprehend operations using the AWS CLI, Java, and Python. Use them to learn about Amazon Comprehend operations and as building blocks for your own applications.

To run the AWS CLI and Python examples, you need to install the AWS CLI. For more information, see Step 2: Set Up the AWS Command Line Interface (AWS CLI) (p. 14).

To run the Java examples, you need to install the AWS SDK for Java. For instructions for installing the SDK for Java, see Set up the AWS SDK for Java.

Topics

Detecting the Dominant Language

To determine the dominant language used in text, use the Amazon Comprehend DetectDominantLanguage (p. 72) operation. To detect the dominant language in up to 25 documents in a batch, use the BatchDetectDominantLanguage (p. 58) operation. For more information, see Using the Batch APIs (p. 38).

Topics

Detecting the Dominant Language Using the AWS Command Line Interface

The following example demonstrates using the `DetectDominantLanguage` operation with the AWS CLI.

The example is formatted for Unix, Linux, and macOS. For Windows, replace the backslash (\) Unix continuation character at the end of each line with a caret (^).

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

Amazon Comprehend responds with the following:

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

Detecting the Dominant Language Using the AWS SDK for Java

The following example uses the `DetectDominantLanguage` operation with Java.

```
import com.amazonaws.auth.AWSCredentialsProvider;
import com.amazonaws.auth.DefaultAWSCredentialsProviderChain;
import com.amazonaws.services.comprehend.AmazonComprehend;
import com.amazonaws.services.comprehend.AmazonComprehendClientBuilder;
import com.amazonaws.services.comprehend.model.DetectDominantLanguageRequest;
import com.amazonaws.services.comprehend.model.DetectDominantLanguageResult;

public class App
{
    public static void main( String[] args )
    {

        String text = "It is raining today in Seattle";

        // Create credentials using a provider chain. For more information, see
        // https://docs.aws.amazon.com/sdk-for-java/v1/developer-guide/credentials.html
        AWSCredentialsProvider awsCreds = DefaultAWSCredentialsProviderChain.getInstance();

        AmazonComprehend comprehendClient =
            AmazonComprehendClientBuilder.standard()
                                         .withCredentials(awsCreds)
                                         .withRegion("region")
                                         .build();

        // Call detectDominantLanguage API
        System.out.println("Calling DetectDominantLanguage");
        DetectDominantLanguageRequest detectDominantLanguageRequest = new
 DetectDominantLanguageRequest().withText(text);
        DetectDominantLanguageResult detectDominantLanguageResult =
 comprehendClient.detectDominantLanguage(detectDominantLanguageRequest);
        detectDominantLanguageResult.getLanguages().forEach(System.out::println);
        System.out.println("Calling DetectDominantLanguage\n");
        System.out.println("Done");
    }
}
```

Detecting the Dominant Language Using the AWS SDK for Python (Boto)

The following example demonstrates using the `DetectDominantLanguage` operation with Python.

```
import boto3
import json

comprehend = boto3.client(service_name='comprehend', region_name='region')
text = "It is raining today in Seattle"
```

```
print('Calling DetectDominantLanguage')
print(json.dumps(comprehend.detect_dominant_language(Text = text), sort_keys=True,
 indent=4))
print("End of DetectDominantLanguage\n")
```

Detecting the Dominant Language Using the AWS SDK for .NET

The .NET example in this section uses the AWS SDK for .NET. You can use the AWS Toolkit for Visual Studio to develop AWS applications using .NET. It includes helpful templates and the AWS Explorer for deploying applications and managing services. For a .NET developer perspective of AWS, see the AWS Guide for .NET Developers.

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

namespace Comprehend
{
    class Program
    {
        static void Main(string[] args)
        {
            String text = "It is raining today in Seattle";

            AmazonComprehendClient comprehendClient = new
 AmazonComprehendClient(Amazon.RegionEndpoint.USWest2);

            // Call DetectDominantLanguage API
            Console.WriteLine("Calling DetectDominantLanguage\n");
            DetectDominantLanguageRequest detectDominantLanguageRequest = new
 DetectDominantLanguageRequest()
            {
                Text = text
            };
            DetectDominantLanguageResponse detectDominantLanguageResponse =
 comprehendClient.DetectDominantLanguage(detectDominantLanguageRequest);
            foreach (DominantLanguage dl in detectDominantLanguageResponse.Languages)
                Console.WriteLine("Language Code: {0}, Score: {1}", dl.LanguageCode,
 dl.Score);
            Console.WriteLine("Done");
        }
    }
}
```

Detecting Named Entities

To determine the named entities in a document, use the Amazon Comprehend DetectEntities (p. 75) operation. To detect entities in up to 25 documents in a batch, use the BatchDetectEntities (p. 61) operation. For more information, see Using the Batch APIs (p. 38).

Topics

Detecting Named Entities Using the AWS Command Line Interface

The following example demonstrates using the `DetectEntities` operation using the AWS CLI. You must specify the language of the input text.

The example is formatted for Unix, Linux, and macOS. For Windows, replace the backslash (\) Unix continuation character at the end of each line with a caret (^).

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

Amazon Comprehend responds with the following:

```
{
    "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"
}
```

Detecting Named Entities Using the AWS SDK for Java

The following example uses the `DetectEntities` operation with Java. You must specify the language of the input text.

```
import com.amazonaws.auth.AWSCredentialsProvider;
import com.amazonaws.auth.DefaultAWSCredentialsProviderChain;
import com.amazonaws.services.comprehend.AmazonComprehend;
import com.amazonaws.services.comprehend.AmazonComprehendClientBuilder;
import com.amazonaws.services.comprehend.model.DetectEntitiesRequest;
import com.amazonaws.services.comprehend.model.DetectEntitiesResult;

public class App
{
    public static void main( String[] args )
    {

        String text = "It is raining today in Seattle";

        // Create credentials using a provider chain. For more information, see
        // https://docs.aws.amazon.com/sdk-for-java/v1/developer-guide/credentials.html
        AWSCredentialsProvider awsCreds = DefaultAWSCredentialsProviderChain.getInstance();
```

```
        AmazonComprehend comprehendClient =
            AmazonComprehendClientBuilder.standard()
                                         .withCredentials(awsCreds)
                                         .withRegion("region")
                                         .build();

        // Call detectEntities API
        System.out.println("Calling DetectEntities");
        DetectEntitiesRequest detectEntitiesRequest = new
 DetectEntitiesRequest().withText(text)

 .withLanguageCode("en");
        DetectEntitiesResult detectEntitiesResult  =
 comprehendClient.detectEntities(detectEntitiesRequest);
        detectEntitiesResult.getEntities().forEach(System.out::println);
        System.out.println("End of DetectEntities\n");
    }
}
```

Detecting Named Entities Using the AWS SDK for Python (Boto)

The following example uses the `DetectEntities` operation with Python. You must specify the language of the input text.

```
import boto3
import json

comprehend = boto3.client(service_name='comprehend', region_name='region')
text = "It is raining today in Seattle"

print('Calling DetectEntities')
print(json.dumps(comprehend.detect_entities(Text=text, LanguageCode='en'), sort_keys=True,
 indent=4))
print('End of DetectEntities\n')
```

Detecting Entities Using the AWS SDK for .NET

The .NET example in this section uses the AWS SDK for .NET. You can use the AWS Toolkit for Visual Studio to develop AWS applications using .NET. It includes helpful templates and the AWS Explorer for deploying applications and managing services. For a .NET developer perspective of AWS, see the AWS Guide for .NET Developers.

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

namespace Comprehend
{
    class Program
    {
        static void Main(string[] args)
        {
            String text = "It is raining today in Seattle";

            AmazonComprehendClient comprehendClient = new
 AmazonComprehendClient(Amazon.RegionEndpoint.USWest2);

            // Call DetectEntities API
            Console.WriteLine("Calling DetectEntities\n");
            DetectEntitiesRequest detectEntitiesRequest =  new DetectEntitiesRequest()
            {
```

```
                Text = text,
                LanguageCode = "en"
            };
            DetectEntitiesResponse detectEntitiesResponse =
 comprehendClient.DetectEntities(detectEntitiesRequest);
            foreach (Entity e in detectEntitiesResponse.Entities)
                Console.WriteLine("Text: {1}, Type: {1}, Score: {2}, BeginOffset: {3},
 EndOffset: {4}",
                        e.Text, e.Type, e.Score, e.BeginOffset, e.EndOffset);
            Console.WriteLine("Done");
        }
    }
}
```

Detecting Key Phrases

To determine the key noun phrases used in text, use the Amazon Comprehend DetectKeyPhrases (p. 78) operation. To detect the key noun phrases in up to 25 documents in a batch, use the BatchDetectKeyPhrases (p. 64) operation. For more information, see Using the Batch APIs (p. 38).

Topics

- Detecting Key Phrases Using the AWS Command Line Interface (p. 28)
- Detecting Key Phrases Using the AWS SDK for Java (p. 29)
- Detecting Key Phrases Using the AWS SDK for Python (Boto) (p. 29)
- Detecting Key Phrases Using the AWS SDK for .NET (p. 30)

Detecting Key Phrases Using the AWS Command Line Interface

The following example demonstrates using the `DetectKeyPhrases` operation with the AWS CLI. You must specify the language of the input text.

The example is formatted for Unix, Linux, and macOS. For Windows, replace the backslash (\) Unix continuation character at the end of each line with a caret (^).

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

Amazon Comprehend responds with the following:

```
{
    "LanguageCode": "en",
    "KeyPhrases": [
        {
            "Text": "today",
            "Score": 0.89,
            "BeginOffset": 14,
            "EndOffset": 19
        },
        {
            "Text": "Seattle",
            "Score": 0.91,
            "BeginOffset": 23,
            "EndOffset": 30
```

```
        }
    ]
}
```

Detecting Key Phrases Using the AWS SDK for Java

The following example uses the `DetectKeyPhrases` operation with Java. You must specify the language of the input text.

```
import com.amazonaws.auth.AWSCredentialsProvider;
import com.amazonaws.auth.DefaultAWSCredentialsProviderChain;
import com.amazonaws.services.comprehend.AmazonComprehend;
import com.amazonaws.services.comprehend.AmazonComprehendClientBuilder;
import com.amazonaws.services.comprehend.model.DetectKeyPhrasesRequest;
import com.amazonaws.services.comprehend.model.DetectKeyPhrasesResult;

public class App
{
    public static void main( String[] args )
    {

        String text = "It is raining today in Seattle";

        // Create credentials using a provider chain. For more information, see
        // https://docs.aws.amazon.com/sdk-for-java/v1/developer-guide/credentials.html
        AWSCredentialsProvider awsCreds = DefaultAWSCredentialsProviderChain.getInstance();

        AmazonComprehend comprehendClient =
            AmazonComprehendClientBuilder.standard()
                                         .withCredentials(awsCreds)
                                         .withRegion("region")
                                         .build();

        // Call detectKeyPhrases API
        System.out.println("Calling DetectKeyPhrases");
        DetectKeyPhrasesRequest detectKeyPhrasesRequest = new
 DetectKeyPhrasesRequest().withText(text)

 .withLanguageCode("en");
        DetectKeyPhrasesResult detectKeyPhrasesResult =
 comprehendClient.detectKeyPhrases(detectKeyPhrasesRequest);
        detectKeyPhrasesResult.getKeyPhrases().forEach(System.out::println);
        System.out.println("End of DetectKeyPhrases\n");
    }
}
```

Detecting Key Phrases Using the AWS SDK for Python (Boto)

The following example uses the `DetectKeyPhrases` operation with Python. You must specify the language of the input text.

```
import boto3
import json

comprehend = boto3.client(service_name='comprehend', region_name='region')

text = "It is raining today in Seattle"

print('Calling DetectKeyPhrases')
print(json.dumps(comprehend.detect_key_phrases(Text=text, LanguageCode='en'),
 sort_keys=True, indent=4))
```

```
print('End of DetectKeyPhrases\n')
```

Detecting Key Phrases Using the AWS SDK for .NET

The .NET example in this section uses the AWS SDK for .NET. You can use the AWS Toolkit for Visual Studio to develop AWS applications using .NET. It includes helpful templates and the AWS Explorer for deploying applications and managing services. For a .NET developer perspective of AWS, see the AWS Guide for .NET Developers.

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

namespace Comprehend
{
    class Program
    {
        static void Main(string[] args)
        {
            String text = "It is raining today in Seattle";

            AmazonComprehendClient comprehendClient = new
 AmazonComprehendClient(Amazon.RegionEndpoint.USWest2);

            // Call DetectKeyPhrases API
            Console.WriteLine("Calling DetectKeyPhrases");
            DetectKeyPhrasesRequest detectKeyPhrasesRequest = new DetectKeyPhrasesRequest()
            {
                Text = text,
                LanguageCode = "en"
            };
            DetectKeyPhrasesResponse detectKeyPhrasesResponse =
 comprehendClient.DetectKeyPhrases(detectKeyPhrasesRequest);
            foreach (KeyPhrase kp in detectKeyPhrasesResponse.KeyPhrases)
                Console.WriteLine("Text: {1}, Type: {1}, BeginOffset: {2}, EndOffset: {3}",
                    kp.Text, kp.Text, kp.BeginOffset, kp.EndOffset);
            Console.WriteLine("Done");
        }
    }
}
```

Detecting Sentiment

To determine the overall emotional tone of text, use the DetectSentiment (p. 81) operation. To detect the sentiment in up to 25 documents in a batch, use the BatchDetectSentiment (p. 67) operation. For more information, see Using the Batch APIs (p. 38).

Topics

Detecting Sentiment Using the AWS Command Line Interface

The following example demonstrates using the `DetectSentiment` operation with the AWS CLI. This example specifies the language of the input text.

The example is formatted for Unix, Linux, and macOS. For Windows, replace the backslash (\) Unix continuation character at the end of each line with a caret (^).

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

Amazon Comprehend responds with the following:

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

Detecting Sentiment Using the AWS SDK for Java

The following example Java program detects the sentiment of input text. You must specify the language of the input text.

```
import com.amazonaws.auth.AWSCredentialsProvider;
import com.amazonaws.auth.DefaultAWSCredentialsProviderChain;
import com.amazonaws.services.comprehend.AmazonComprehend;
import com.amazonaws.services.comprehend.AmazonComprehendClientBuilder;
import com.amazonaws.services.comprehend.model.DetectSentimentRequest;
import com.amazonaws.services.comprehend.model.DetectSentimentResult;

public class App
{
    public static void main( String[] args )
    {

        String text = "It is raining today in Seattle";

        // Create credentials using a provider chain. For more information, see
        // https://docs.aws.amazon.com/sdk-for-java/v1/developer-guide/credentials.html
        AWSCredentialsProvider awsCreds = DefaultAWSCredentialsProviderChain.getInstance();

        AmazonComprehend comprehendClient =
            AmazonComprehendClientBuilder.standard()
                                         .withCredentials(awsCreds)
                                         .withRegion("region")
                                         .build();

        // Call detectSentiment API
        System.out.println("Calling DetectSentiment");
        DetectSentimentRequest detectSentimentRequest = new
 DetectSentimentRequest().withText(text)

 .withLanguageCode("en");
        DetectSentimentResult detectSentimentResult =
 comprehendClient.detectSentiment(detectSentimentRequest);
        System.out.println(detectSentimentResult);
        System.out.println("End of DetectSentiment\n");
        System.out.println( "Done" );
    }
```

```
}
```

Detecting Sentiment Using the AWS SDK for Python (Boto)

The following Python program detects the sentiment of input text. You must specify the language of the input text.

```
import boto3
import json

comprehend = boto3.client(service_name='comprehend', region_name='region')

text = "It is raining today in Seattle"

print('Calling DetectSentiment')
print(json.dumps(comprehend.detect_sentiment(Text=text, LanguageCode='en'), sort_keys=True,
 indent=4))
print('End of DetectSentiment\n')
```

Detecting Sentiment Using the AWS SDK for .NET

The .NET example in this section uses the AWS SDK for .NET. You can use the AWS Toolkit for Visual Studio to develop AWS applications using .NET. It includes helpful templates and the AWS Explorer for deploying applications and managing services. For a .NET developer perspective of AWS, see the AWS Guide for .NET Developers.

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

namespace Comprehend
{
    class Program
    {
        static void Main(string[] args)
        {
            String text = "It is raining today in Seattle";

            AmazonComprehendClient comprehendClient = new
 AmazonComprehendClient(Amazon.RegionEndpoint.USWest2);

            // Call DetectKeyPhrases API
            Console.WriteLine("Calling DetectSentiment");
            DetectSentimentRequest detectSentimentRequest = new DetectSentimentRequest()
            {
                Text = text,
                LanguageCode = "en"
            };
            DetectSentimentResponse detectSentimentResponse =
 comprehendClient.DetectSentiment(detectSentimentRequest);
            Console.WriteLine(detectSentimentResponse.Sentiment);
            Console.WriteLine("Done");
        }
    }
}
```

Topic Modeling

To determine the topics in a document set, use the StartTopicsDetectionJob (p. 87) to start an asynchronous job. You can monitor topics in documents written in English or Spanish.

Topics

Before You Start

Before you start, make sure that you have:

- **Input and output buckets**—Identify the Amazon S3 buckets that you want to use for input and output. The buckets must be in the same region as the API that you are calling.
- **IAM service role**—You must have an IAM service role with permission to access your input and output buckets. For more information, see Role-Based Permissions Required for Topic Detection (p. 52).

Topic Modeling Using the AWS Command Line Interface

The following example demonstrates using the `StartTopicsDetectionJob` operation with the AWS CLI

The example is formatted for Unix, Linux, and macOS. For Windows, replace the backslash (\) Unix continuation character at the end of each line with a caret (^).

```
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
```

For the `cli-input-json` parameter you supply the path to a JSON file that contains the request data, as shown in the following example.

```
{
    "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"
}
```

If the request to start the topic detection job was successful, you will receive the following response:

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

Use the ListTopicsDetectionJobs (p. 84) operation to see a list of the topic detection jobs that you have submitted. The list includes information about the input and output locations that you used as well as the status of each of the detection jobs. The example is formatted for Unix, Linux, and macOS. For Windows, replace the backslash (\) Unix continuation character at the end of each line with a caret (^).

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

You will get JSON similar to the following in response:

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

You can use the DescribeTopicsDetectionJob (p. 70) operation to get the status of an existing job. The example is formatted for Unix, Linux, and macOS. For Windows, replace the backslash (\) Unix continuation character at the end of each line with a caret (^).

```
aws comprehend describe-topics-detection-job
                --region region \
                --job-id job ID
I
```

You will get the following JSON in response:

```
{
    "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
    }
}
```

Topic Modeling Using the AWS SDK for Java

The following Java program detects the topics in a document collection. It uses the StartTopicsDetectionJob (p. 87) operation to start detecting topics. Next, it uses the DescribeTopicsDetectionJob (p. 70) operation to check the status of the topic detection. Finally, it calls ListTopicsDetectionJobs (p. 84) to show a list of all jobs submitted for the account.

```
import com.amazonaws.auth.AWSCredentialsProvider;
import com.amazonaws.auth.DefaultAWSCredentialsProviderChain;
import com.amazonaws.client.builder.AwsClientBuilder;
import com.amazonaws.services.comprehend.AmazonComprehend;
import com.amazonaws.services.comprehend.AmazonComprehendClientBuilder;
import com.amazonaws.services.comprehend.model.DescribeTopicsDetectionJobRequest;
import com.amazonaws.services.comprehend.model.DescribeTopicsDetectionJobResult;
import com.amazonaws.services.comprehend.model.InputDataConfig;
import com.amazonaws.services.comprehend.model.InputFormat;
import com.amazonaws.services.comprehend.model.ListTopicsDetectionJobsRequest;
import com.amazonaws.services.comprehend.model.ListTopicsDetectionJobsResult;
import com.amazonaws.services.comprehend.model.StartTopicsDetectionJobRequest;
import com.amazonaws.services.comprehend.model.StartTopicsDetectionJobResult;

public class App
{
    public static void main( String[] args )
    {
        // Create credentials using a provider chain. For more information, see
        // https://docs.aws.amazon.com/sdk-for-java/v1/developer-guide/credentials.html
         AWSCredentialsProvider awsCreds =
 DefaultAWSCredentialsProviderChain.getInstance();

        AmazonComprehend comprehendClient =
            AmazonComprehendClientBuilder.standard()
                                         .withCredentials(awsCreds)
                                         .withRegion("region")
                                         .build();

        final String inputS3Uri = "s3://input bucket/input path";
        final InputFormat inputDocFormat = InputFormat.ONE_DOC_PER_FILE;
        final String outputS3Uri = "s3://output bucket/output path";
        final String dataAccessRoleArn = "arn:aws:iam::account ID:role/data access role";
        final int numberOfTopics = 10;

        final StartTopicsDetectionJobRequest startTopicsDetectionJobRequest = new
 StartTopicsDetectionJobRequest()
                .withInputDataConfig(new InputDataConfig()
                        .withS3Uri(inputS3Uri)
                        .withInputFormat(inputDocFormat))
                .withOutputDataConfig(new OutputDataConfig()
                        .withS3Uri(outputS3Uri))
                .withDataAccessRoleArn(dataAccessRoleArn)
                .withNumberOfTopics(numberOfTopics);

        final StartTopicsDetectionJobResult startTopicsDetectionJobResult =
 comprehendClient.startTopicsDetectionJob(startTopicsDetectionJobRequest);

        final String jobId = startTopicsDetectionJobResult.getJobId();
```

```
        System.out.println("JobId: " + jobId);

        final DescribeTopicsDetectionJobRequest describeTopicsDetectionJobRequest = new
 DescribeTopicsDetectionJobRequest()
                .withJobId(jobId);

        final DescribeTopicsDetectionJobResult describeTopicsDetectionJobResult =
 comprehendClient.describeTopicsDetectionJob(describeTopicsDetectionJobRequest);
        System.out.println("describeTopicsDetectionJobResult: " +
 describeTopicsDetectionJobResult);

        ListTopicsDetectionJobsResult listTopicsDetectionJobsResult =
 comprehendClient.listTopicsDetectionJobs(new ListTopicsDetectionJobsRequest());
        System.out.println("listTopicsDetectionJobsResult: " +
 listTopicsDetectionJobsResult);

    }
}
```

Topic Modeling Using the AWS SDK for Python (Boto)

The following Python program detects the topics in a document collection. It uses the StartTopicsDetectionJob (p. 87) operation to start detecting topics. Next, it uses the DescribeTopicsDetectionJob (p. 70) operation to check the status of the topic detection. Finally, it calls ListTopicsDetectionJobs (p. 84) to show a list of all jobs submitted for the account.

```
import boto3
import json
from bson import json_util

comprehend = boto3.client(service_name='comprehend', region_name='region')

input_s3_url = "s3://input bucket/input path"
input_doc_format = "ONE_DOC_PER_FILE"
output_s3_url = "s3://output bucket/output path"
data_access_role_arn = "arn:aws:iam::account ID:role/data access role"
number_of_topics = 10

input_data_config = {"S3Uri": input_s3_url, "InputFormat": input_doc_format}
output_data_config = {"S3Uri": output_s3_url}

start_topics_detection_job_result =
 comprehend.start_topics_detection_job(NumberOfTopics=number_of_topics,

 InputDataConfig=input_data_config,

 OutputDataConfig=output_data_config,

 DataAccessRoleArn=data_access_role_arn)

print('start_topics_detection_job_result: ' +
 json.dumps(start_topics_detection_job_result))

job_id = start_topics_detection_job_result["JobId"]

print('job_id: ' + job_id)

describe_topics_detection_job_result =
 comprehend.describe_topics_detection_job(JobId=job_id)

print('describe_topics_detection_job_result: ' +
 json.dumps(describe_topics_detection_job_result, default=json_util.default))
```

```
list_topics_detection_jobs_result = comprehend.list_topics_detection_jobs()

print('list_topics_detection_jobs_result: ' + json.dumps(list_topics_detection_jobs_result,
 default=json_util.default))
```

Topic Modeling Using the AWS SDK for .NET

The following C# program detects the topics in a document collection. It uses the StartTopicsDetectionJob (p. 87) operation to start detecting topics. Next, it uses the DescribeTopicsDetectionJob (p. 70) operation to check the status of the topic detection. Finally, it calls ListTopicsDetectionJobs (p. 84) to show a list of all jobs submitted for the account.

The .NET example in this section uses the AWS SDK for .NET. You can use the AWS Toolkit for Visual Studio to develop AWS applications using .NET. It includes helpful templates and the AWS Explorer for deploying applications and managing services. For a .NET developer perspective of AWS, see the AWS Guide for .NET Developers.

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

namespace Comprehend
{
    class Program
    {
        // Helper method for printing properties
        static private void PrintJobProperties(TopicsDetectionJobProperties props)
        {
            Console.WriteLine("JobId: {0}, JobName: {1}, JobStatus: {2}, NumberOfTopics:
 {3}\nInputS3Uri: {4}, InputFormat: {5}, OutputS3Uri: {6}",
                props.JobId, props.JobName, props.JobStatus, props.NumberOfTopics,
                props.InputDataConfig.S3Uri, props.InputDataConfig.InputFormat,
 props.OutputDataConfig.S3Uri);
        }

        static void Main(string[] args)
        {
            String text = "It is raining today in Seattle";

            AmazonComprehendClient comprehendClient = new
 AmazonComprehendClient(Amazon.RegionEndpoint.USWest2);

            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;

            StartTopicsDetectionJobRequest startTopicsDetectionJobRequest = new
 StartTopicsDetectionJobRequest()
            {
                InputDataConfig = new InputDataConfig()
                {
                    S3Uri = inputS3Uri,
                    InputFormat = inputDocFormat
                },
                OutputDataConfig = new OutputDataConfig()
                {
                    S3Uri = outputS3Uri
                },
                DataAccessRoleArn = dataAccessRoleArn,
                NumberOfTopics = numberOfTopics
            };
```

```
            StartTopicsDetectionJobResponse startTopicsDetectionJobResponse =
 comprehendClient.StartTopicsDetectionJob(startTopicsDetectionJobRequest);

            String jobId = startTopicsDetectionJobResponse.JobId;
            Console.WriteLine("JobId: " + jobId);

            DescribeTopicsDetectionJobRequest describeTopicsDetectionJobRequest = new
 DescribeTopicsDetectionJobRequest()
            {
                JobId = jobId
            };

            DescribeTopicsDetectionJobResponse describeTopicsDetectionJobResponse =
 comprehendClient.DescribeTopicsDetectionJob(describeTopicsDetectionJobRequest);

 PrintJobProperties(describeTopicsDetectionJobResponse.TopicsDetectionJobProperties);

            ListTopicsDetectionJobsResponse listTopicsDetectionJobsResponse =
 comprehendClient.ListTopicsDetectionJobs(new ListTopicsDetectionJobsRequest());
            foreach (TopicsDetectionJobProperties props in
 listTopicsDetectionJobsResponse.TopicsDetectionJobPropertiesList)
                PrintJobProperties(props);
        }
    }
}
```

Using the Batch APIs

To send batches of up to 25 documents, you can use the Amazon Comprehend batch operations. Calling a batch operation is identical to calling the single document APIs for each document in the request. Using the batch APIs can result in better performance for your applications. For more information, see Batch Processing Documents (p. 7).

Topics

- Batch Processing With the SDK for Java (p. 38)
- Batch Processing With the AWS SDK for .NET (p. 42)
- Batch Processing With the AWS CLI (p. 43)

Batch Processing With the SDK for Java

The following sample program shows how to use the

Inspects the text of a batch of documents for named entities and returns information about them. For more information about named entities, see Detecting Entities (p. 3)

Request Syntax

```
{
   "LanguageCode": "string",
   "TextList": [ "string" ]
}
```

Request Parameters

For information about the parameters that are common to all actions, see Common Parameters (p. 107).

The request accepts the following data in JSON format.

LanguageCode (p. 61)

The language of the input documents. You can specify English ("en") or Spanish ("es"). All documents must be in the same language.

Type: String

Length Constraints: Minimum length of 1.

Required: Yes

TextList (p. 61)

A list containing the text of the input documents. The list can contain a maximum of 25 documents. Each document must contain fewer than 5,000 bytes of UTF-8 encoded characters.

Type: Array of strings

Length Constraints: Minimum length of 1.

Required: Yes

Response Syntax

```
{
   "ErrorList": [
      {
         "ErrorCode": "string",
         "ErrorMessage": "string",
         "Index": number
      }
   ],
   "ResultList": [
      {
         "Entities": [
            {
               "BeginOffset": number,
               "EndOffset": number,
               "Score": number,
               "Text": "string",
               "Type": "string"
            }
         ],
```

the order of the documents in the input list. If there are no errors in the batch, the

`ErrorList` is empty.

Type: Array of BatchItemError (p. 95) objects

ResultList (p. 61)

A list of BatchDetectEntitiesItemResult (p. 92) objects containing the results of the operation. The results are sorted in ascending order by the `Index` field and match the order of the documents in the input list. If all of the documents contain an error, the `ResultList` is empty.

Type: Array of BatchDetectEntitiesItemResult (p. 92) objects

Errors

For information about the errors that are common to all actions, see Common Errors (p. 105).

BatchSizeLimitExceededException

The number of documents in the request exceeds the limit of 25. Try your request again with fewer documents.

HTTP Status Code: 400

InternalServerException

An internal server error occurred. Retry your request.

HTTP Status Code: 500

InvalidRequestException

The request is invalid.

HTTP Status Code: 400

TextSizeLimitExceededException

The size of the input text exceeds the limit. Use a smaller document.

HTTP Status Code: 400

UnsupportedLanguageException

Amazon Comprehend can't process the language of the input text. For all APIs except `DetectDominantLanguage`, Amazon Comprehend accepts only English or Spanish text. For the `DetectDominantLanguage` API, Amazon Comprehend detects 100 languages. For a list of languages, see Detecting the Primary Language (p. 4)

HTTP Status Code: 400

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS Command Line Interface
- AWS SDK for .NET
- AWS SDK for C++
- AWS SDK for Go

- AWS SDK for Java

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

(p. 61) operation with the SDK for Java. The response from the server contains a BatchDetectEntitiesItemResult (p. 92) object for each document that was successfully processed. If there is an error processing a document there will be a record in the error list in the response. The example gets each of the documents with an error and resends them.

```
import com.amazonaws.auth.AWSStaticCredentialsProvider;
import com.amazonaws.auth.DefaultAWSCredentialsProviderChain;
import com.amazonaws.services.comprehend.AmazonComprehend;
import com.amazonaws.services.comprehend.AmazonComprehendClientBuilder;
import com.amazonaws.services.comprehend.model.BatchDetectEntitiesItemResult;
import com.amazonaws.services.comprehend.model.BatchDetectEntitiesRequest;
import com.amazonaws.services.comprehend.model.BatchDetectEntitiesResult;
import com.amazonaws.services.comprehend.model.BatchItemError;

public class App
{
    public static void main( String[] args )
    {

        // Create credentials using a provider chain. For more information, see
        // https://docs.aws.amazon.com/sdk-for-java/v1/developer-guide/credentials.html
        AWSCredentialsProvider awsCreds = DefaultAWSCredentialsProviderChain.getInstance();

        AmazonComprehend comprehendClient =
            AmazonComprehendClientBuilder.standard()
                                         .withCredentials(awsCreds)
                                         .withRegion("region")
                                         .build();

       String[] textList = {"I love Seattle", "Today is Sunday", "Tomorrow is Monday", "I
 love Seattle"};

       // Call detectEntities API
       System.out.println("Calling BatchDetectEntities");
       BatchDetectEntitiesRequest batchDetectEntitiesRequest = new
 BatchDetectEntitiesRequest().withTextList(textList)

     .withLanguageCode("en");
       BatchDetectEntitiesResult batchDetectEntitiesResult  =
 client.batchDetectEntities(batchDetectEntitiesRequest);

       for(BatchDetectEntitiesItemResult item : batchDetectEntitiesResult.getResultList())
 {
           System.out.println(item);
       }

       // check if we need to retry failed requests
       if (batchDetectEntitiesResult.getErrorList().size() != 0)
       {
           System.out.println("Retrying Failed Requests");
           ArrayList<String> textToRetry = new ArrayList<String>();
           for(BatchItemError errorItem : batchDetectEntitiesResult.getErrorList())
           {
               textToRetry.add(textList[errorItem.getIndex()]);
           }
```

```
            batchDetectEntitiesRequest = new
 BatchDetectEntitiesRequest().withTextList(textToRetry).withLanguageCode("en");
            batchDetectEntitiesResult  =
 client.batchDetectEntities(batchDetectEntitiesRequest);

            for(BatchDetectEntitiesItemResult item :
 batchDetectEntitiesResult.getResultList()) {
                System.out.println(item);
            }

        }
        System.out.println("End of DetectEntities");
    }
}
```

Batch Processing With the AWS SDK for .NET

The following sample program shows how to use the BatchDetectEntities (p. 61) operation with the AWS SDK for .NET. The response from the server contains a BatchDetectEntitiesItemResult (p. 92) object for each document that was successfully processed. If there is an error processing a document there will be a record in the error list in the response. The example gets each of the documents with an error and resends them.

The .NET example in this section uses the AWS SDK for .NET. You can use the AWS Toolkit for Visual Studio to develop AWS applications using .NET. It includes helpful templates and the AWS Explorer for deploying applications and managing services. For a .NET developer perspective of AWS, see the AWS Guide for .NET Developers.

```
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");
        }
    }
}
```

Batch Processing With the AWS CLI

These examples show how to use the batch API operations using the AWS Command Line Interface. All of the operations except `BatchDetectDominantLanguage` use the following JSON file called `process.json` as input. For that operation the `LanguageCode` entity is not included.

The third document in the JSON file (`"$$$$$$$$$"`) will cause an error during batch processing. It is included so that the operations will include a BatchItemError (p. 95) in the response.

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

The examples are formatted for Unix, Linux, and macOS. For Windows, replace the backslash (\) Unix continuation character at the end of each line with a caret (^).

Topics

Detect the Dominant Language Using a Batch (AWS CLI)

The BatchDetectDominantLanguage (p. 58) operation determines the dominant language of each document in a batch. For a list of the languages that Amazon Comprehend can detect, see Detecting the Primary Language (p. 4). The following AWS CLI command calls the `BatchDetectDominantLanguage` operation.

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

The following is the response from the `BatchDetectDominantLanguage` operation:

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

Detect Entities Using a Batch (AWS CLI)

Use the BatchDetectEntities (p. 61) operation to find the entities present in a batch of documents. For more information about entities, see Detecting Entities (p. 3). The following AWS CLI command calls the `BatchDetectEntities` operation.

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

Detect Key Phrases Using a Batch (AWS CLI)

The BatchDetectKeyPhrases (p. 64) operation returns the key noun phrases in a batch of documents. The following AWS CLI command calls the `BatchDetectKeyNounPhrases` operation.

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

Detect Sentiment Using a Batch (AWS CLI)

Detect the overall sentiment of a batch of documents using the BatchDetectSentiment (p. 67) operation. The following AWS CLI command calls the `BatchDetectSentiment` operation.

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

Authentication and Access Control for Amazon Comprehend

Access to Amazon Comprehend requires credentials that AWS can use to authenticate your requests. Those credentials must have permissions to access Amazon Comprehend actions. The following sections provide details on how you can use AWS Identity and Access Management (IAM) and Amazon Comprehend to help secure your resources by controlling who can access them.

Authentication

You can access AWS as any of the following types of identities:

- **AWS account root user** – When you first create an AWS account, you begin with a single sign-in identity that has complete access to all AWS services and resources in the account. This identity is called the AWS account *root user* and is accessed by signing in with the email address and password that you used to create the account. We strongly recommend that you do not use the root user for your everyday tasks, even the administrative ones. Instead, adhere to the best practice of using the root user only to create your first IAM user. Then securely lock away the root user credentials and use them to perform only a few account and service management tasks.
- **IAM user** – An IAM user is an identity within your AWS account that has specific custom permissions (for example, permissions to create in Amazon Comprehend). You can use an IAM user name and password to sign in to secure AWS webpages like the AWS Management Console, AWS Discussion Forums, or the AWS Support Center.

  In addition to a user name and password, you can also generate access keys for each user. You can use these keys when you access AWS services programmatically, either through one of the several SDKs or by using the AWS Command Line Interface (CLI). The SDK and CLI tools use the access keys to cryptographically sign your request. If you don't use AWS tools, you must sign the request yourself. Amazon Comprehend supports *Signature Version 4*, a protocol for authenticating inbound API requests. For more information about authenticating requests, see Signature Version 4 Signing Process in the *AWS General Reference*.

- **IAM role** – An IAM role is an IAM identity that you can create in your account that has specific permissions. It is similar to an *IAM user*, but it is not associated with a specific person. An IAM role enables you to obtain temporary access keys that can be used to access AWS services and resources. IAM roles with temporary credentials are useful in the following situations:

  - **Federated user access** – Instead of creating an IAM user, you can use existing user identities from AWS Directory Service, your enterprise user directory, or a web identity provider. These are known as *federated users*. AWS assigns a role to a federated user when access is requested through an identity provider. For more information about federated users, see Federated Users and Roles in the *IAM User Guide*.

- **AWS service access** – You can use an IAM role in your account to grant an AWS service permissions to access your account's resources. For example, you can create a role that allows Amazon Redshift to access an Amazon S3 bucket on your behalf and then load data from that bucket into an Amazon Redshift cluster. For more information, see Creating a Role to Delegate Permissions to an AWS Service in the *IAM User Guide*.

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

Access Control

You must have valid credentials to authenticate your requests. The credentials must have permissions to call an Amazon Comprehend action.

The following sections describe how to manage permissions for Amazon Comprehend. We recommend that you read the overview first.

Overview of Managing Access Permissions to Amazon Comprehend Resources

Permissions to access an action are governed by permissions policies. An account administrator can attach permissions policies to IAM identities (that is, users, groups, and roles) to manage access to actions.

> **Note**
> An *account administrator* (or administrator user) is a user with administrator privileges. For more information, see IAM Best Practices in the *IAM User Guide*.

When granting permissions, you decide who is getting the permissions and the actions they get permissions for.

Topics

Managing Access to Actions

A *permissions policy* describes who has access to what. The following section explains the available options for creating permissions policies.

Note
This section discusses using IAM in the context of Amazon Comprehend. It doesn't provide detailed information about the IAM service. For complete IAM documentation, see What Is IAM? in the *IAM User Guide*. For information about IAM policy syntax and descriptions, see AWS IAM Policy Reference in the *IAM User Guide*.

Policies attached to an IAM identity are referred to as *identity-based* policies (IAM policies) and policies attached to a resource are referred to as *resource-based* policies. Amazon Comprehend supports only identity-based policies.

Identity-Based Policies (IAM Policies)

You can attach policies to IAM identities. For example, you can do the following:

- **Attach a permissions policy to a user or a group in your account** – To grant a user or a group of users permissions to call and Amazon Comprehend action, you can attach a permissions policy to a user or group that the user belongs to.
- **Attach a permissions policy to a role (grant cross-account permissions)** – To grant cross-account permissions, you can attach an identity-based permissions policy to an IAM role. For example, the administrator in Account A can create a role to grant cross-account permissions to another AWS account (for example, Account B) or an AWS service as follows:
  1. Account A administrator creates an IAM role and attaches a permissions policy to the role that grants permissions on resources in Account A.
  2. Account A administrator attaches a trust policy to the role identifying Account B as the principal who can assume the role.
  3. Account B administrator can then delegate permissions to assume the role to any users in Account B. Doing this allows users in Account B to create or access resources in Account A. If you want to grant an AWS service permissions to assume the role, the principal in the trust policy can also be an AWS service principal.

  For more information about using IAM to delegate permissions, see Access Management in the *IAM User Guide*.

For more information about using identity-based policies with Amazon Comprehend, see Using Identity-Based Policies (IAM Policies) for Amazon Comprehend (p. 49). For more information about users, groups, roles, and permissions, see Identities (Users, Groups, and Roles) in the *IAM User Guide*.

Resource-Based Policies

Other services, such as Lambda, support resource-based permissions policies. For example, you can attach a policy to an S3 bucket to manage access permissions to that bucket. Amazon Comprehend doesn't support resource-based policies.

Specifying Policy Elements: Actions, Effects, and Principals

Amazon Comprehend defines a set of API operations (see Actions (p. 57)). To grant permissions for these API operations, Amazon Comprehend defines a set of actions that you can specify in a policy.

The following are the most basic policy elements:

- **Resource** – In a policy, you use an Amazon Resource Name (ARN) to identify the resource to which the policy applies. For Amazon Comprehend, the resource is always `"*"`.

- **Action** – You use action keywords to identify operations that you want to allow or deny. For example, depending on the specified `Effect`, `comprehend:DetectEntities` either allows or denies the user permissions to perform the Amazon Comprehend `DetectEntities` operation.
- **Effect** – You specify the effect of the action that occurs when the user requests the specific action —this can be either allow or deny. If you don't explicitly grant access to (allow) a resource, access is implicitly denied. You can also explicitly deny access to a resource. You might do this to make sure that a user cannot access the resource, even if a different policy grants access.
- **Principal** – In identity-based policies (IAM policies), the user that the policy is attached to is the implicit principal.

To learn more about IAM policy syntax and descriptions, see AWS IAM Policy Reference in the *IAM User Guide*.

For a table showing all of the Amazon Comprehend API actions, see Amazon Comprehend API Permissions: Actions, Resources, and Conditions Reference (p. 54).

Specifying Conditions in a Policy

When you grant permissions, you use the IAM policy language to specify the conditions under which a policy should take effect. For example, you might want a policy to be applied only after a specific date. For more information about specifying conditions in a policy language, see Condition in the *IAM User Guide*.

AWS provides a set of predefined condition keys for all AWS services that support IAM for access control. For example, you can use the `aws:userid` condition key to require a specific AWS ID when requesting an action. For more information and a complete list of AWS-wide keys, see Available Keys for Conditions in the *IAM User Guide*.

> **Note**
> Condition keys are case sensitive.

Amazon Comprehend does not provide any additional condition keys.

Using Identity-Based Policies (IAM Policies) for Amazon Comprehend

This topic provides examples of identity-based policies that demonstrate how an account administrator can attach permissions policies to IAM identities (that is, users, groups, and roles) and thereby grant permissions to perform Amazon Comprehend actions.

> **Important**
> Before you proceed, we recommend that you review Overview of Managing Access Permissions to Amazon Comprehend Resources (p. 47).

The following is the permissions policy required to use the Amazon Comprehend document analysis actions:

```
{
   "Version": "2012-10-17",
   "Statement": [{
      "Sid": "AllowDetectActions",
      "Effect": "Allow",
      "Action": [
                "comprehend:DetectEntities",
```

```
                "comprehend:DetectKeyPhrases",
                "comprehend:DetectDominantLanguage",
                "comprehend:DetectSentiment"
            ],
        "Resource": "*"
        }
    ]
}
```

The policy has one statement that grants permission to use the `DetectEntities`, `DetectKeyPhrases`, `DetectDominantLanguage` and `DetectSentiment` actions. A user with this policy would not be able to perform batch actions in your account.

The policy doesn't specify the `Principal` element because you don't specify the principal who gets the permission in an identity-based policy. When you attach a policy to a user, the user is the implicit principal. When you attach a permissions policy to an IAM role, the principal identified in the role's trust policy gets the permissions.

For a table showing all of the Amazon Comprehend API actions and the resources that they apply to, see Amazon Comprehend API Permissions: Actions, Resources, and Conditions Reference (p. 54).

Permissions Required to Use the Amazon Comprehend Console

The permissions reference table lists the Amazon Comprehend API operations and shows the required permissions for each operation. For more information, about Amazon Comprehend API permissions, see Amazon Comprehend API Permissions: Actions, Resources, and Conditions Reference (p. 54).

To use the Amazon Comprehend console, you need to grant permissions for the actions shown in the following policy:

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Action": [
                "comprehend:*",
                "iam:ListRoles",
                "iam:GetRole",
                "s3:ListAllMyBuckets",
                "s3:ListBucket",
                "s3:GetBucketLocation"
            ],
            "Effect": "Allow",
            "Resource": "*"
        }
    ]
}
```

The Amazon Comprehend console needs these additional permissions for the following reasons:

- `iam` permissions to list the available IAM roles for your account.
- `s3` permissions to access the Amazon S3 buckets and objects that contain the data for topic modeling.

When you create a topic modeling job using the console, you have the option to have the console create an IAM role for your job. To create an IAM role, users must be granted the following additional permissions to create IAM roles and policies, and to attach policies to roles:

```
{
  "Version": "2012-10-17",
  "Statement": [
      {
         "Action": [
            "iam:CreateRole",
            "iam:CreatePolicy",
            "iam:AttachRolePolicy",
            "iam:PassRole"
         ],
         "Effect": "Allow",
         "Resource": "*"
      }
   ]
}
```

The Amazon Comprehend console needs these additional permissions for the following reasons:

- `iam` permissions to create roles and policies and to attach roles and policies. The `iam:PassRole` action enables the console to pass the role to Amazon Comprehend.

AWS Managed (Predefined) Policies for Amazon Comprehend

AWS addresses many common use cases by providing standalone IAM policies that are created and administered by AWS. These AWS managed policies grant necessary permissions for common use cases so that you can avoid having to investigate what permissions are needed. For more information, see AWS Managed Policies in the *IAM User Guide*.

The following AWS managed policies, which you can attach to users in your account, are specific to Amazon Comprehend:

- **ComprehendFullAccess** – Grants full access to Amazon Comprehend resources including running topic modeling jobs. Includes permission to list and get IAM roles.
- **ComprehendReadOnly** – Grants permission to run all Amazon Comprehend actions except `StartTopicsDetectionJob`.

You need to apply the following additional policy to any user that will use Amazon Comprehend:

```
{
  "Version": "2012-10-17",
  "Statement": [
      {
         "Action": [
            "iam:PassRole"
         ],
         "Effect": "Allow",
         "Resource": "*"
      }
   ]
}
```

You can review the managed permissions policies by signing in to the IAM console and searching for specific policies there.

These policies work when you are using AWS SDKs or the AWS CLI.

You can also create your own custom IAM policies to allow permissions for Amazon Comprehend actions and resources. You can attach these custom policies to the IAM users or groups that require those permissions.

Role-Based Permissions Required for Topic Detection

To use the Amazon Comprehend topic detection operations, you must grant Amazon Comprehend access to the Amazon S3 bucket that contains your document collection. You do this by creating a data access role in your account to trust the Amazon Comprehend service principal. For more information about creating a role, see Creating a Role to Delegate Permissions to an AWS Service in the *AWS Identity and Access Management User Guide*.

The following is the role's trust policy:

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "Service": "comprehend.amazonaws.com"
      },
      "Action": "sts:AssumeRole"
    }
  ]
}
```

After you have created the role, you must create an access policy for that role. The should grant the Amazon S3 `GetObject` and `ListBucket` permissions to the Amazon S3 bucket that contains your input data, and the Amazon S3 `PutObject` permission to your Amazon S3 output data bucket.

The following example access policy contains those permissions.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Action": [
                "s3:GetObject"
            ],
            "Resource": [
                "arn:aws:s3:::input bucket/*"
            ],
            "Effect": "Allow"
        },
        {
            "Action": [
                "s3:ListBucket"
            ],
            "Resource": [
                "arn:aws:s3:::input bucket"
            ],
            "Effect": "Allow"
        },
        {
            "Action": [
                "s3:PutObject"
            ],
            "Resource": [
                "arn:aws:s3:::output bucket/*"
            ],
```

```
            "Effect": "Allow"
        }
    ]
}
```

Customer Managed Policy Examples

In this section, you can find example user policies that grant permissions for various Amazon Comprehend actions. These policies work when you are using AWS SDKs or the AWS CLI. When you are using the console, you need to grant permissions to all the Amazon Comprehend APIs. This is discussed in Permissions Required to Use the Amazon Comprehend Console (p. 50).

Note
All examples use the us-east-2 region and contain fictitious account IDs.

Examples

Example 1: Allow All Amazon Comprehend Actions

After you sign up for AWS, you create an administrator user to manage your account, including creating users and managing their permissions.

You might choose to create a user who has permissions for all Amazon Comprehend actions (think of this user as a service-specific administrator) for working with Amazon Comprehend. You can attach the following permissions policy to this user.

```
{
   "Version": "2012-10-17",
   "Statement": [{
      "Sid": "AllowAllComprehendActions",
      "Effect": "Allow",
      "Action": [
         "comprehend:*"],
      "Resource": "*"
      }
   ]
}
```

Example 2: Allow Topic Modeling Actions

The following permissions policy grants user permissions to perform the Amazon Comprehend topic modeling operations.

```
{
   "Version": "2012-10-17",
   "Statement": [{
      "Sid": "AllowTopicModelingActions",
      "Effect": "Allow",
      "Action": [
                "comprehend:DescribeTopicsDetectionJob",
                "comprehend:ListTopicsDetectionJobs",
                "comprehend:StartTopicsDetectionJob",
             ],
            "Resource": "*"
            ]
        }
    ]
}
```

Amazon Comprehend API Permissions: Actions, Resources, and Conditions Reference

Use the following table as a reference when setting up Access Control (p. 47) and writing a permissions policy that you can attach to an IAM identity (an identity-based policy). The list includes each Amazon Comprehend API operation, the corresponding action for which you can grant permissions to perform the action, and the AWS resource for which you can grant the permissions. You specify the actions in the policy's `Action` field, and you specify the resource value in the policy's `Resource` field.

To express conditions, you can use AWS-wide condition keys in your Amazon Comprehend policies. For a complete list of AWS-wide keys, see Available Keys in the *IAM User Guide*.

Note
To specify an action, use the `comprehend:` prefix followed by the API operation name, for example, `comprehend:DetectEntities`.

Guidelines and Limits

Keep in mind the following information when using Amazon Comprehend.

Supported Regions

For a list of AWS Regions where Amazon Comprehend is availabe, see AWS Regions and Endpoints in the *Amazon Web Services General Reference*.

Throttling

For information about throttling for Amazon Comprehend and to request a limit increase, see Amazon Comprehend Limits in the *Amazon Web Services General Reference*.

You may be able to avoid throttling by using the batch operations instead of the single transaction operations. For more information, see Batch Operations (p. 55).

Overall Limits

All operations except topic modeling operations have the following limits:

Description	Limit
Character encoding	UTF-8
Document size (UTF-8 characters)	5,000 bytes

Amazon Comprehend may store your content to continuously improve the quality of its analysis models. See the Amazon Comprehend FAQ to learn more. To request that we delete content that may have been stored by Amazon Comprehend, open a case with AWS Support.

Batch Operations

The BatchDetectDominantLanguage (p. 58), BatchDetectEntities (p. 61), BatchDetectKeyPhrases (p. 64), and BatchDetectSentiment (p. 67) operations have the following limits:

Description	Limit
Documents per request	25

If you plan to send more than 20 requests per second, you should consider using the batch operations. Batch operations enable you to send more documents in each request which may result in higher

throughput. For example, when you use the `DetectDominantLanguage` operation, you can send up to 20 documents per second. However, if you use the `BatchRequestDominantLanguage` operation, you can send up to 250 documents per second, but processing speed may be lower. For more information about throttling limits see Amazon Comprehend Limits in the *Amazon Web Services General Reference*. For more information about using the batch APIs, see Batch Processing Documents (p. 7).

Language Detection

The BatchDetectDominantLanguage (p. 58) and DetectDominantLanguage (p. 72) operations have the following limitations:

- They don't support phonetic language detection. For example, they will not detect "arigato" as Japanese nor "nihao" as Chinese.
- They may have trouble distinguishing close language pairs, such as Indonesian and Malay; or Bosnian, Croation, and Serbian.
- For best results the input text should be at least 20 characters long.

Topic Modeling

Topic detection jobs created with the StartTopicsDetectionJob (p. 87) operation have the following limits:

Description	Limit
Character encoding	UTF-8
Maximum number of topics to return	100
Total size of all files in request	5 Gb
Maximum file size for one file, one document per file	100 Mb
Maximum number of files, one document per file	1,000,000
Maximum number of lines, one document per line	1,000,000

For best results, you should include at least 1,000 input documents.

API Reference

This section provides documentation for the Amazon Comprehend API operations.

Actions

The following actions are supported:

BatchDetectDominantLanguage

Determines the dominant language of the input text for a batch of documents. For a list of languages that Amazon Comprehend can detect, see Amazon Comprehend Supported Languages.

Request Syntax

```
{
   "TextList": [ "string" ]
}
```

Request Parameters

For information about the parameters that are common to all actions, see Common Parameters (p. 107).

The request accepts the following data in JSON format.

TextList (p. 58)

A list containing the text of the input documents. The list can contain a maximum of 25 documents. Each document should contain at least 20 characters and must contain fewer than 5,000 bytes of UTF-8 encoded characters.

Type: Array of strings

Length Constraints: Minimum length of 1.

Required: Yes

Response Syntax

```
{
   "ErrorList": [
      {
         "ErrorCode": "string",
         "ErrorMessage": "string",
         "Index": number
      }
   ],
   "ResultList": [
      {
         "Index": number,
         "Languages": [
            {
               "LanguageCode": "string",
               "Score": number
            }
         ]
      }
   ]
}
```

Response Elements

If the action is successful, the service sends back an HTTP 200 response.

The following data is returned in JSON format by the service.

ErrorList (p. 58)

A list containing one BatchItemError (p. 95) object for each document that contained an error. The results are sorted in ascending order by the `Index` field and match the order of the documents in the input list. If there are no errors in the batch, the `ErrorList` is empty.

Type: Array of BatchItemError (p. 95) objects

ResultList (p. 58)

A list of BatchDetectDominantLanguageItemResult (p. 91) objects containing the results of the operation. The results are sorted in ascending order by the `Index` field and match the order of the documents in the input list. If all of the documents contain an error, the `ResultList` is empty.

Type: Array of BatchDetectDominantLanguageItemResult (p. 91) objects

Errors

For information about the errors that are common to all actions, see Common Errors (p. 105).

BatchSizeLimitExceededException

The number of documents in the request exceeds the limit of 25. Try your request again with fewer documents.

HTTP Status Code: 400

InternalServerException

An internal server error occurred. Retry your request.

HTTP Status Code: 500

InvalidRequestException

The request is invalid.

HTTP Status Code: 400

TextSizeLimitExceededException

The size of the input text exceeds the limit. Use a smaller document.

HTTP Status Code: 400

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS Command Line Interface
- AWS SDK for .NET
- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Java
- AWS SDK for JavaScript
- AWS SDK for PHP V3
- AWS SDK for Python

- AWS SDK for Ruby V2

BatchDetectEntities

Inspects the text of a batch of documents for named entities and returns information about them. For more information about named entities, see Detecting Entities (p. 3)

Request Syntax

```
{
   "LanguageCode": "string",
   "TextList": [ "string" ]
}
```

Request Parameters

For information about the parameters that are common to all actions, see Common Parameters (p. 107).

The request accepts the following data in JSON format.

LanguageCode (p. 61)

The language of the input documents. You can specify English ("en") or Spanish ("es"). All documents must be in the same language.

Type: String

Length Constraints: Minimum length of 1.

Required: Yes

TextList (p. 61)

A list containing the text of the input documents. The list can contain a maximum of 25 documents. Each document must contain fewer than 5,000 bytes of UTF-8 encoded characters.

Type: Array of strings

Length Constraints: Minimum length of 1.

Required: Yes

Response Syntax

```
{
   "ErrorList": [
      {
         "ErrorCode": "string",
         "ErrorMessage": "string",
         "Index": number
      }
   ],
   "ResultList": [
      {
         "Entities": [
            {
               "BeginOffset": number,
               "EndOffset": number,
               "Score": number,
               "Text": "string",
```

```
                "Type": "string"
            }
        ],
        "Index": number
    }
]
}
```

Response Elements

If the action is successful, the service sends back an HTTP 200 response.

The following data is returned in JSON format by the service.

ErrorList (p. 61)

A list containing one BatchItemError (p. 95) object for each document that contained an error. The results are sorted in ascending order by the `Index` field and match the order of the documents in the input list. If there are no errors in the batch, the `ErrorList` is empty.

Type: Array of BatchItemError (p. 95) objects

ResultList (p. 61)

A list of BatchDetectEntitiesItemResult (p. 92) objects containing the results of the operation. The results are sorted in ascending order by the `Index` field and match the order of the documents in the input list. If all of the documents contain an error, the `ResultList` is empty.

Type: Array of BatchDetectEntitiesItemResult (p. 92) objects

Errors

For information about the errors that are common to all actions, see Common Errors (p. 105).

BatchSizeLimitExceededException

The number of documents in the request exceeds the limit of 25. Try your request again with fewer documents.

HTTP Status Code: 400

InternalServerException

An internal server error occurred. Retry your request.

HTTP Status Code: 500

InvalidRequestException

The request is invalid.

HTTP Status Code: 400

TextSizeLimitExceededException

The size of the input text exceeds the limit. Use a smaller document.

HTTP Status Code: 400

UnsupportedLanguageException

Amazon Comprehend can't process the language of the input text. For all APIs except `DetectDominantLanguage`, Amazon Comprehend accepts only English or Spanish text. For

the `DetectDominantLanguage` API, Amazon Comprehend detects 100 languages. For a list of languages, see Detecting the Primary Language (p. 4)

HTTP Status Code: 400

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS Command Line Interface
- AWS SDK for .NET
- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Java
- AWS SDK for JavaScript
- AWS SDK for PHP V3
- AWS SDK for Python
- AWS SDK for Ruby V2

BatchDetectKeyPhrases

Detects the key noun phrases found in a batch of documents.

Request Syntax

```
{
   "LanguageCode": "string",
   "TextList": [ "string" ]
}
```

Request Parameters

For information about the parameters that are common to all actions, see Common Parameters (p. 107).

The request accepts the following data in JSON format.

LanguageCode (p. 64)

The language of the input documents. You can specify English ("en") or Spanish ("es"). All documents must be in the same language.

Type: String

Length Constraints: Minimum length of 1.

Required: Yes

TextList (p. 64)

A list containing the text of the input documents. The list can contain a maximum of 25 documents. Each document must contain fewer that 5,000 bytes of UTF-8 encoded characters.

Type: Array of strings

Length Constraints: Minimum length of 1.

Required: Yes

Response Syntax

```
{
   "ErrorList": [
      {
         "ErrorCode": "string",
         "ErrorMessage": "string",
         "Index": number
      }
   ],
   "ResultList": [
      {
         "Index": number,
         "KeyPhrases": [
            {
               "BeginOffset": number,
               "EndOffset": number,
               "Score": number,
```

```
            "Text": "string"
        }
    ]
  }
 ]
}
```

Response Elements

If the action is successful, the service sends back an HTTP 200 response.

The following data is returned in JSON format by the service.

ErrorList (p. 64)

A list containing one BatchItemError (p. 95) object for each document that contained an error. The results are sorted in ascending order by the `Index` field and match the order of the documents in the input list. If there are no errors in the batch, the `ErrorList` is empty.

Type: Array of BatchItemError (p. 95) objects

ResultList (p. 64)

A list of BatchDetectKeyPhrasesItemResult (p. 93) objects containing the results of the operation. The results are sorted in ascending order by the `Index` field and match the order of the documents in the input list. If all of the documents contain an error, the `ResultList` is empty.

Type: Array of BatchDetectKeyPhrasesItemResult (p. 93) objects

Errors

For information about the errors that are common to all actions, see Common Errors (p. 105).

BatchSizeLimitExceededException

The number of documents in the request exceeds the limit of 25. Try your request again with fewer documents.

HTTP Status Code: 400

InternalServerException

An internal server error occurred. Retry your request.

HTTP Status Code: 500

InvalidRequestException

The request is invalid.

HTTP Status Code: 400

TextSizeLimitExceededException

The size of the input text exceeds the limit. Use a smaller document.

HTTP Status Code: 400

UnsupportedLanguageException

Amazon Comprehend can't process the language of the input text. For all APIs except `DetectDominantLanguage`, Amazon Comprehend accepts only English or Spanish text. For

the `DetectDominantLanguage` API, Amazon Comprehend detects 100 languages. For a list of languages, see Detecting the Primary Language (p. 4)

HTTP Status Code: 400

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS Command Line Interface
- AWS SDK for .NET
- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Java
- AWS SDK for JavaScript
- AWS SDK for PHP V3
- AWS SDK for Python
- AWS SDK for Ruby V2

BatchDetectSentiment

Inspects a batch of documents and returns an inference of the prevailing sentiment, `POSITIVE`, `NEUTRAL`, `MIXED`, or `NEGATIVE`, in each one.

Request Syntax

```
{
   "LanguageCode": "string",
   "TextList": [ "string" ]
}
```

Request Parameters

For information about the parameters that are common to all actions, see Common Parameters (p. 107).

The request accepts the following data in JSON format.

LanguageCode (p. 67)

The language of the input documents. You can specify English ("en") or Spanish ("es"). All documents must be in the same language.

Type: String

Length Constraints: Minimum length of 1.

Required: Yes

TextList (p. 67)

A list containing the text of the input documents. The list can contain a maximum of 25 documents. Each document must contain fewer that 5,000 bytes of UTF-8 encoded characters.

Type: Array of strings

Length Constraints: Minimum length of 1.

Required: Yes

Response Syntax

```
{
   "ErrorList": [
      {
         "ErrorCode": "string",
         "ErrorMessage": "string",
         "Index": number
      }
   ],
   "ResultList": [
      {
         "Index": number,
         "Sentiment": "string",
         "SentimentScore": {
            "Mixed": number,
            "Negative": number,
```

```
            "Neutral": number,
            "Positive": number
         }
      }
   ]
}
```

Response Elements

If the action is successful, the service sends back an HTTP 200 response.

The following data is returned in JSON format by the service.

ErrorList (p. 67)

A list containing one BatchItemError (p. 95) object for each document that contained an error. The results are sorted in ascending order by the `Index` field and match the order of the documents in the input list. If there are no errors in the batch, the `ErrorList` is empty.

Type: Array of BatchItemError (p. 95) objects

ResultList (p. 67)

A list of BatchDetectSentimentItemResult (p. 94) objects containing the results of the operation. The results are sorted in ascending order by the `Index` field and match the order of the documents in the input list. If all of the documents contain an error, the `ResultList` is empty.

Type: Array of BatchDetectSentimentItemResult (p. 94) objects

Errors

For information about the errors that are common to all actions, see Common Errors (p. 105).

BatchSizeLimitExceededException

The number of documents in the request exceeds the limit of 25. Try your request again with fewer documents.

HTTP Status Code: 400

InternalServerException

An internal server error occurred. Retry your request.

HTTP Status Code: 500

InvalidRequestException

The request is invalid.

HTTP Status Code: 400

TextSizeLimitExceededException

The size of the input text exceeds the limit. Use a smaller document.

HTTP Status Code: 400

UnsupportedLanguageException

Amazon Comprehend can't process the language of the input text. For all APIs except `DetectDominantLanguage`, Amazon Comprehend accepts only English or Spanish text. For

the `DetectDominantLanguage` API, Amazon Comprehend detects 100 languages. For a list of languages, see Detecting the Primary Language (p. 4)

HTTP Status Code: 400

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS Command Line Interface
- AWS SDK for .NET
- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Java
- AWS SDK for JavaScript
- AWS SDK for PHP V3
- AWS SDK for Python
- AWS SDK for Ruby V2

DescribeTopicsDetectionJob

Gets the properties associated with a topic detection job. Use this operation to get the status of a detection job.

Request Syntax

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

Request Parameters

For information about the parameters that are common to all actions, see Common Parameters (p. 107).

The request accepts the following data in JSON format.

JobId (p. 70)

The identifier assigned by the user to the detection job.

Type: String

Length Constraints: Minimum length of 1. Maximum length of 32.

Required: Yes

Response Syntax

```
{
   "TopicsDetectionJobProperties": {
      "EndTime": number,
      "InputDataConfig": {
         "InputFormat": "string",
         "S3Uri": "string"
      },
      "JobId": "string",
      "JobName": "string",
      "JobStatus": "string",
      "Message": "string",
      "NumberOfTopics": number,
      "OutputDataConfig": {
         "S3Uri": "string"
      },
      "SubmitTime": number
   }
}
```

Response Elements

If the action is successful, the service sends back an HTTP 200 response.

The following data is returned in JSON format by the service.

TopicsDetectionJobProperties (p. 70)

The list of properties for the requested job.

Type: TopicsDetectionJobProperties (p. 104) object

Errors

For information about the errors that are common to all actions, see Common Errors (p. 105).

InternalServerException

An internal server error occurred. Retry your request.

HTTP Status Code: 500

InvalidRequestException

The request is invalid.

HTTP Status Code: 400

JobNotFoundException

The specified job was not found. Check the job ID and try again.

HTTP Status Code: 400

TooManyRequestsException

The number of requests exceeds the limit. Resubmit your request later.

HTTP Status Code: 400

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS Command Line Interface
- AWS SDK for .NET
- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Java
- AWS SDK for JavaScript
- AWS SDK for PHP V3
- AWS SDK for Python
- AWS SDK for Ruby V2

DetectDominantLanguage

Determines the dominant language of the input text. For a list of languages that Amazon Comprehend can detect, see Amazon Comprehend Supported Languages.

Request Syntax

```
{
   "Text": "string"
}
```

Request Parameters

For information about the parameters that are common to all actions, see Common Parameters (p. 107).

The request accepts the following data in JSON format.

Text (p. 72)

A UTF-8 text string. Each string should contain at least 20 characters and must contain fewer that 5,000 bytes of UTF-8 encoded characters.

Type: String

Length Constraints: Minimum length of 1.

Required: Yes

Response Syntax

```
{
   "Languages": [
      {
         "LanguageCode": "string",
         "Score": number
      }
   ]
}
```

Response Elements

If the action is successful, the service sends back an HTTP 200 response.

The following data is returned in JSON format by the service.

Languages (p. 72)

The languages that Amazon Comprehend detected in the input text. For each language, the response returns the RFC 5646 language code and the level of confidence that Amazon Comprehend has in the accuracy of its inference. For more information about RFC 5646, see Tags for Identifying Languages on the *IETF Tools* web site.

Type: Array of DominantLanguage (p. 96) objects

Errors

For information about the errors that are common to all actions, see Common Errors (p. 105).

InternalServerException

An internal server error occurred. Retry your request.

HTTP Status Code: 500

InvalidRequestException

The request is invalid.

HTTP Status Code: 400

TextSizeLimitExceededException

The size of the input text exceeds the limit. Use a smaller document.

HTTP Status Code: 400

Example

Detect dominant language

If the input text is "Bob lives in Seattle. He is a software engineer at Amazon.", the operation returns the following:

```
{
    "Languages": [
        {
            "LanguageCode": "en",
            "Score": 0.9774383902549744
        },
        {
            "LanguageCode": "de",
            "Score": 0.010717987082898617
        }
    ]
}
```

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS Command Line Interface
- AWS SDK for .NET
- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Java
- AWS SDK for JavaScript
- AWS SDK for PHP V3
- AWS SDK for Python
- AWS SDK for Ruby V2

DetectEntities

Inspects text for named entities, and returns information about them. For more information, about named entities, see Detecting Entities (p. 3).

Request Syntax

```
{
   "LanguageCode": "string",
   "Text": "string"
}
```

Request Parameters

For information about the parameters that are common to all actions, see Common Parameters (p. 107).

The request accepts the following data in JSON format.

LanguageCode (p. 75)

The language of the input documents. You can specify English ("en") or Spanish ("es"). All documents must be in the same language.

Type: String

Valid Values: `en | es`

Required: Yes

Text (p. 75)

A UTF-8 text string. Each string must contain fewer that 5,000 bytes of UTF-8 encoded characters.

Type: String

Length Constraints: Minimum length of 1.

Required: Yes

Response Syntax

```
{
   "Entities": [
      {
         "BeginOffset": number,
         "EndOffset": number,
         "Score": number,
         "Text": "string",
         "Type": "string"
      }
   ]
}
```

Response Elements

If the action is successful, the service sends back an HTTP 200 response.

The following data is returned in JSON format by the service.

Entities (p. 75)

A collection of entities identified in the input text. For each entity, the response provides the entity text, entity type, where the entity text begins and ends, and the level of confidence that Amazon Comprehend has in the detection. For a list of entity types, see Detecting Entities (p. 3).

Type: Array of Entity (p. 97) objects

Errors

For information about the errors that are common to all actions, see Common Errors (p. 105).

InternalServerException

An internal server error occurred. Retry your request.

HTTP Status Code: 500

InvalidRequestException

The request is invalid.

HTTP Status Code: 400

TextSizeLimitExceededException

The size of the input text exceeds the limit. Use a smaller document.

HTTP Status Code: 400

UnsupportedLanguageException

Amazon Comprehend can't process the language of the input text. For all APIs except `DetectDominantLanguage`, Amazon Comprehend accepts only English or Spanish text. For the `DetectDominantLanguage` API, Amazon Comprehend detects 100 languages. For a list of languages, see Detecting the Primary Language (p. 4)

HTTP Status Code: 400

Example

Detect entities

If the input text is "Bob ordered two sandwiches and three ice cream cones today from a store in Seattle.", the operation returns the following:

```
{
"Entities": [
    {
        "Text": "Bob",
        "Score": 1.0,
        "Type": "PERSON",
        "BeginOffset": 0,
        "EndOffset": 3
    },
    {
        "Text": "two",
        "Score": 1.0,
```

```
            "Type": "QUANTITY",
            "BeginOffset": 12,
            "EndOffset": 15
        },
        {
            "Text": "three",
            "Score": 1.0,
            "Type": "QUANTITY",
            "BeginOffset": 32,
            "EndOffset": 37
        },
        {
            "Text": "Today",
            "Score": 1.0,
            "Type": "DATE",
            "BeginOffset": 54,
            "EndOffset": 59
        },
        {
            "Text": "Seattle",
            "Score": 1.0,
            "Type": "LOCATION",
            "BeginOffset": 76,
            "EndOffset": 83
        }
    ],
}
```


See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS Command Line Interface
- AWS SDK for .NET
- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Java
- AWS SDK for JavaScript
- AWS SDK for PHP V3
- AWS SDK for Python
- AWS SDK for Ruby V2

DetectKeyPhrases

Detects the key noun phrases found in the text.

Request Syntax

```
{
   "LanguageCode": "string",
   "Text": "string"
}
```

Request Parameters

For information about the parameters that are common to all actions, see Common Parameters (p. 107).

The request accepts the following data in JSON format.

LanguageCode (p. 78)

The language of the input documents. You can specify English ("en") or Spanish ("es"). All documents must be in the same language.

Type: String

Valid Values: `en | es`

Required: Yes

Text (p. 78)

A UTF-8 text string. Each string must contain fewer that 5,000 bytes of UTF-8 encoded characters.

Type: String

Length Constraints: Minimum length of 1.

Required: Yes

Response Syntax

```
{
   "KeyPhrases": [
      {
         "BeginOffset": number,
         "EndOffset": number,
         "Score": number,
         "Text": "string"
      }
   ]
}
```

Response Elements

If the action is successful, the service sends back an HTTP 200 response.

The following data is returned in JSON format by the service.

KeyPhrases (p. 78)

A collection of key phrases that Amazon Comprehend identified in the input text. For each key phrase, the response provides the text of the key phrase, where the key phrase begins and ends, and the level of confidence that Amazon Comprehend has in the accuracy of the detection.

Type: Array of KeyPhrase (p. 100) objects

Errors

For information about the errors that are common to all actions, see Common Errors (p. 105).

InternalServerException

An internal server error occurred. Retry your request.

HTTP Status Code: 500

InvalidRequestException

The request is invalid.

HTTP Status Code: 400

TextSizeLimitExceededException

The size of the input text exceeds the limit. Use a smaller document.

HTTP Status Code: 400

UnsupportedLanguageException

Amazon Comprehend can't process the language of the input text. For all APIs except `DetectDominantLanguage`, Amazon Comprehend accepts only English or Spanish text. For the `DetectDominantLanguage` API, Amazon Comprehend detects 100 languages. For a list of languages, see Detecting the Primary Language (p. 4)

HTTP Status Code: 400

Example

Detect phrases

If the input text is "Bob lives in Seattle. He is a software engineer at Amazon.", the API returns the following:

```
      {
"KeyPhrases": [
    {
        "Text": "Bob",
        "Score": 1.0,
        "BeginOffset": 0,
        "EndOffset": 3
    },
    {
        "Text": "Seattle",
        "Score": 1.0,
        "BeginOffset": 13,
        "EndOffset": 20
    },
```

```
        {
            "Text": "an engineer",
            "Score": 1.0,
            "BeginOffset": 28,
            "EndOffset": 39
        },
        {
            "Text": "Amazon",
            "Score": 1.0,
            "BeginOffset": 43,
            "EndOffset": 49
        }
    ]
}}
```

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS Command Line Interface
- AWS SDK for .NET
- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Java
- AWS SDK for JavaScript
- AWS SDK for PHP V3
- AWS SDK for Python
- AWS SDK for Ruby V2

DetectSentiment

Inspects text and returns an inference of the prevailing sentiment (`POSITIVE`, `NEUTRAL`, `MIXED`, or `NEGATIVE`).

Request Syntax

```
{
   "LanguageCode": "string",
   "Text": "string"
}
```

Request Parameters

For information about the parameters that are common to all actions, see Common Parameters (p. 107).

The request accepts the following data in JSON format.

LanguageCode (p. 81)

The language of the input documents. You can specify English ("en") or Spanish ("es"). All documents must be in the same language.

Type: String

Valid Values: `en | es`

Required: Yes

Text (p. 81)

A UTF-8 text string. Each string must contain fewer that 5,000 bytes of UTF-8 encoded characters.

Type: String

Length Constraints: Minimum length of 1.

Required: Yes

Response Syntax

```
{
   "Sentiment": "string",
   "SentimentScore": {
      "Mixed": number,
      "Negative": number,
      "Neutral": number,
      "Positive": number
   }
}
```

Response Elements

If the action is successful, the service sends back an HTTP 200 response.

The following data is returned in JSON format by the service.

Sentiment (p. 81)

The inferred sentiment that Amazon Comprehend has the highest level of confidence in.

Type: String

Valid Values: `POSITIVE | NEGATIVE | NEUTRAL | MIXED`

SentimentScore (p. 81)

An object that lists the sentiments, and their corresponding confidence levels.

Type: SentimentScore (p. 102) object

Errors

For information about the errors that are common to all actions, see Common Errors (p. 105).

InternalServerException

An internal server error occurred. Retry your request.

HTTP Status Code: 500

InvalidRequestException

The request is invalid.

HTTP Status Code: 400

TextSizeLimitExceededException

The size of the input text exceeds the limit. Use a smaller document.

HTTP Status Code: 400

UnsupportedLanguageException

Amazon Comprehend can't process the language of the input text. For all APIs except `DetectDominantLanguage`, Amazon Comprehend accepts only English or Spanish text. For the `DetectDominantLanguage` API, Amazon Comprehend detects 100 languages. For a list of languages, see Detecting the Primary Language (p. 4)

HTTP Status Code: 400

Example

Detect sentiment

If the input text is "Today is my birthday, I am so happy.", the operation returns the following response:

```
{
    "SentimentScore": {
        "Mixed": 0.0033542951568961143,
        "Positive": 0.9869875907897949,
        "Neutral": 0.008563132025301456,
        "Negative": 0.0010949420975521207
    },
    "Sentiment": "POSITIVE",
 }
}
```

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS Command Line Interface
- AWS SDK for .NET
- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Java
- AWS SDK for JavaScript
- AWS SDK for PHP V3
- AWS SDK for Python
- AWS SDK for Ruby V2

ListTopicsDetectionJobs

Gets a list of the topic detection jobs that you have submitted.

Request Syntax

```
{
   "Filter": {
      "JobName": "string",
      "JobStatus": "string",
      "SubmitTimeAfter": number,
      "SubmitTimeBefore": number
   },
   "MaxResults": number,
   "NextToken": "string"
}
```

Request Parameters

For information about the parameters that are common to all actions, see Common Parameters (p. 107).

The request accepts the following data in JSON format.

Filter (p. 84)

Filters the jobs that are returned. Jobs can be filtered on their name, status, or the date and time that they were submitted. You can set only one filter at a time.

Type: TopicsDetectionJobFilter (p. 103) object

Required: No

MaxResults (p. 84)

The maximum number of results to return in each page.

Type: Integer

Valid Range: Minimum value of 1. Maximum value of 500.

Required: No

NextToken (p. 84)

Identifies the next page of results to return.

Type: String

Length Constraints: Minimum length of 1.

Required: No

Response Syntax

```
{
   "NextToken": "string",
   "TopicsDetectionJobPropertiesList": [
      {
```

```
         "EndTime": number,
         "InputDataConfig": {
            "InputFormat": "string",
            "S3Uri": "string"
         },
         "JobId": "string",
         "JobName": "string",
         "JobStatus": "string",
         "Message": "string",
         "NumberOfTopics": number,
         "OutputDataConfig": {
            "S3Uri": "string"
         },
         "SubmitTime": number
      }
   ]
}
```

Response Elements

If the action is successful, the service sends back an HTTP 200 response.

The following data is returned in JSON format by the service.

NextToken (p. 84)

Identifies the next page of results to return.

Type: String

Length Constraints: Minimum length of 1.

TopicsDetectionJobPropertiesList (p. 84)

A list containing the properties of each job that is returned.

Type: Array of TopicsDetectionJobProperties (p. 104) objects

Errors

For information about the errors that are common to all actions, see Common Errors (p. 105).

InternalServerException

An internal server error occurred. Retry your request.

HTTP Status Code: 500

InvalidFilterException

The filter specified for the `ListTopicDetectionJobs` operation is invalid. Specify a different filter.

HTTP Status Code: 400

InvalidRequestException

The request is invalid.

HTTP Status Code: 400

TooManyRequestsException

The number of requests exceeds the limit. Resubmit your request later.

HTTP Status Code: 400

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS Command Line Interface
- AWS SDK for .NET
- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Java
- AWS SDK for JavaScript
- AWS SDK for PHP V3
- AWS SDK for Python
- AWS SDK for Ruby V2

StartTopicsDetectionJob

Starts an asynchronous topic detection job. Use the `DescribeTopicDetectionJob` operation to track the status of a job.

Request Syntax

```
{
   "ClientRequestToken": "string",
   "DataAccessRoleArn": "string",
   "InputDataConfig": {
      "InputFormat": "string",
      "S3Uri": "string"
   },
   "JobName": "string",
   "NumberOfTopics": number,
   "OutputDataConfig": {
      "S3Uri": "string"
   }
}
```

Request Parameters

For information about the parameters that are common to all actions, see Common Parameters (p. 107).

The request accepts the following data in JSON format.

ClientRequestToken (p. 87)

A unique identifier for the request. If you do not set the client request token, Amazon Comprehend generates one.

Type: String

Length Constraints: Minimum length of 1. Maximum length of 64.

Pattern: `^[a-zA-Z0-9-]+$`

Required: No

DataAccessRoleArn (p. 87)

The Amazon Resource Name (ARN) of the AWS Identity and Access Management (IAM) role that grants Amazon Comprehend read access to your input data.

Type: String

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

Required: Yes

InputDataConfig (p. 87)

Specifies the format and location of the input data for the job.

Type: InputDataConfig (p. 99) object

Required: Yes

JobName (p. 87)

The identifier of the job.

Type: String

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

Pattern: `^([\p{L}\p{Z}\p{N}_.:/=+\-%@]*)$`

Required: No

NumberOfTopics (p. 87)

The number of topics to detect.

Type: Integer

Valid Range: Minimum value of 1. Maximum value of 100.

Required: No

OutputDataConfig (p. 87)

Specifies where to send the output files.

Type: OutputDataConfig (p. 101) object

Required: Yes

Response Syntax

```
{
   "JobId": "string",
   "JobStatus": "string"
}
```

Response Elements

If the action is successful, the service sends back an HTTP 200 response.

The following data is returned in JSON format by the service.

JobId (p. 88)

The identifier generated for the job. To get the status of the job, use this identifier with the `DescribeTopicDetectionJob` operation.

Type: String

Length Constraints: Minimum length of 1. Maximum length of 32.

JobStatus (p. 88)

The status of the job:

- SUBMITTED - The job has been received and is queued for processing.
- IN_PROGRESS - Amazon Comprehend is processing the job.
- COMPLETED - The job was successfully completed and the output is available.
- FAILED - The job did not complete. To get details, use the `DescribeTopicDetectionJob` operation.

Type: String

Valid Values: `SUBMITTED | IN_PROGRESS | COMPLETED | FAILED`

Errors

For information about the errors that are common to all actions, see Common Errors (p. 105).

InternalServerException

An internal server error occurred. Retry your request.

HTTP Status Code: 500

InvalidRequestException

The request is invalid.

HTTP Status Code: 400

TooManyRequestsException

The number of requests exceeds the limit. Resubmit your request later.

HTTP Status Code: 400

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS Command Line Interface
- AWS SDK for .NET
- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Java
- AWS SDK for JavaScript
- AWS SDK for PHP V3
- AWS SDK for Python
- AWS SDK for Ruby V2

Data Types

The following data types are supported:

BatchDetectDominantLanguageItemResult

The result of calling the BatchDetectDominantLanguage (p. 58) operation. The operation returns one object for each document that is successfully processed by the operation.

Contents

Index

The zero-based index of the document in the input list.

Type: Integer

Required: No

Languages

One or more DominantLanguage (p. 96) objects describing the dominant languages in the document.

Type: Array of DominantLanguage (p. 96) objects

Required: No

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Java
- AWS SDK for Ruby V2

BatchDetectEntitiesItemResult

The result of calling the BatchDetectKeyPhrases (p. 64) operation. The operation returns one object for each document that is successfully processed by the operation.

Contents

Entities

One or more Entity (p. 97) objects, one for each entity detected in the document.

Type: Array of Entity (p. 97) objects

Required: No

Index

The zero-based index of the document in the input list.

Type: Integer

Required: No

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Java
- AWS SDK for Ruby V2

BatchDetectKeyPhrasesItemResult

The result of calling the BatchDetectKeyPhrases (p. 64) operation. The operation returns one object for each document that is successfully processed by the operation.

Contents

Index

The zero-based index of the document in the input list.

Type: Integer

Required: No

KeyPhrases

One or more KeyPhrase (p. 100) objects, one for each key phrase detected in the document.

Type: Array of KeyPhrase (p. 100) objects

Required: No

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Java
- AWS SDK for Ruby V2

BatchDetectSentimentItemResult

The result of calling the BatchDetectSentiment (p. 67) operation. The operation returns one object for each document that is successfully processed by the operation.

Contents

Index

The zero-based index of the document in the input list.

Type: Integer

Required: No

Sentiment

The sentiment detected in the document.

Type: String

Valid Values: `POSITIVE | NEGATIVE | NEUTRAL | MIXED`

Required: No

SentimentScore

The level of confidence that Amazon Comprehend has in the accuracy of its sentiment detection.

Type: SentimentScore (p. 102) object

Required: No

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Java
- AWS SDK for Ruby V2

BatchItemError

Describes an error that occurred while processing a document in a batch. The operation returns on `BatchItemError` object for each document that contained an error.

Contents

ErrorCode

The numeric error code of the error.

Type: String

Length Constraints: Minimum length of 1.

Required: No

ErrorMessage

A text description of the error.

Type: String

Length Constraints: Minimum length of 1.

Required: No

Index

The zero-based index of the document in the input list.

Type: Integer

Required: No

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Java
- AWS SDK for Ruby V2

DominantLanguage

Returns the code for the dominant language in the input text and the level of confidence that Amazon Comprehend has in the accuracy of the detection.

Contents

LanguageCode

The RFC 5646 language code for the dominant language. For more information about RFC 5646, see Tags for Identifying Languages on the *IETF Tools* web site.

Type: String

Length Constraints: Minimum length of 1.

Required: No

Score

The level of confidence that Amazon Comprehend has in the accuracy of the detection.

Type: Float

Required: No

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Java
- AWS SDK for Ruby V2

Entity

Provides information about an entity.

Contents

BeginOffset

A character offset in the input text that shows where the entity begins (the first character is at position 0). The offset returns the position of each UTF-8 code point in the string. A *code point* is the abstract character from a particular graphical representation. For example, a multi-byte UTF-8 character maps to a single code point.

Type: Integer

Required: No

EndOffset

A character offset in the input text that shows where the entity ends. The offset returns the position of each UTF-8 code point in the string. A *code point* is the abstract character from a particular graphical representation. For example, a multi-byte UTF-8 character maps to a single code point.

Type: Integer

Required: No

Score

The level of confidence that Amazon Comprehend has in the accuracy of the detection.

Type: Float

Required: No

Text

The text of the entity.

Type: String

Length Constraints: Minimum length of 1.

Required: No

Type

The entity's type.

Type: String

Valid Values: `PERSON | LOCATION | ORGANIZATION | COMMERCIAL_ITEM | EVENT | DATE | QUANTITY | TITLE | OTHER`

Required: No

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS SDK for C++

- AWS SDK for Go
- AWS SDK for Java
- AWS SDK for Ruby V2

InputDataConfig

The input properties for a topic detection job.

Contents

InputFormat

Specifies how the text in an input file should be processed:

- `ONE_DOC_PER_FILE` - Each file is considered a separate document. Use this option when you are processing large documents, such as newspaper articles or scientific papers.
- `ONE_DOC_PER_LINE` - Each line in a file is considered a separate document. Use this option when you are processing many short documents, such as text messages.

Type: String

Valid Values: `ONE_DOC_PER_FILE | ONE_DOC_PER_LINE`

Required: No

S3Uri

The Amazon S3 URI for the input data. The URI must be in same region as the API endpoint that you are calling. The URI can point to a single input file or it can provide the prefix for a collection of data files.

For example, if you use the URI `S3://bucketName/prefix`, if the prefix is a single file, Amazon Comprehend uses that file as input. If more than one file begins with the prefix, Amazon Comprehend uses all of them as input.

Type: String

Length Constraints: Maximum length of 1024.

Pattern: `s3://([^/]+)(/.*)?`

Required: Yes

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Java
- AWS SDK for Ruby V2

KeyPhrase

Describes a key noun phrase.

Contents

BeginOffset

A character offset in the input text that shows where the key phrase begins (the first character is at position 0). The offset returns the position of each UTF-8 code point in the string. A *code point* is the abstract character from a particular graphical representation. For example, a multi-byte UTF-8 character maps to a single code point.

Type: Integer

Required: No

EndOffset

A character offset in the input text where the key phrase ends. The offset returns the position of each UTF-8 code point in the string. A `code point` is the abstract character from a particular graphical representation. For example, a multi-byte UTF-8 character maps to a single code point.

Type: Integer

Required: No

Score

The level of confidence that Amazon Comprehend has in the accuracy of the detection.

Type: Float

Required: No

Text

The text of a key noun phrase.

Type: String

Length Constraints: Minimum length of 1.

Required: No

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Java
- AWS SDK for Ruby V2

OutputDataConfig

Provides configuration parameters for the output of topic detection jobs.

Contents

S3Uri

When you use the `OutputDataConfig` object with the `StartTopicsDetectionJob` operation, you specify the Amazon S3 location where you want to write the output data. The URI must be in the same region as the API endpoint that you are calling. The location is used as the prefix for the actual location of the output file.

When the topic detection job is finished, the service creates an output file in a directory specific to the job. The `S3Uri` field contains the location of the output file, called `output.tar.gz`. It is a compressed archive that contains two files, `topic-terms.csv` that lists the terms associated with each topic, and `doc-topics.csv` that lists the documents associated with each topic. For more information, see Topic Modeling (p. 9).

Type: String

Length Constraints: Maximum length of 1024.

Pattern: `s3://([^/]+)(/.*)?`

Required: Yes

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Java
- AWS SDK for Ruby V2

SentimentScore

Describes the level of confidence that Amazon Comprehend has in the accuracy of its detection of sentiments.

Contents

Mixed

The level of confidence that Amazon Comprehend has in the accuracy of its detection of the `MIXED` sentiment.

Type: Float

Required: No

Negative

The level of confidence that Amazon Comprehend has in the accuracy of its detection of the `NEGATIVE` sentiment.

Type: Float

Required: No

Neutral

The level of confidence that Amazon Comprehend has in the accuracy of its detection of the `NEUTRAL` sentiment.

Type: Float

Required: No

Positive

The level of confidence that Amazon Comprehend has in the accuracy of its detection of the `POSITIVE` sentiment.

Type: Float

Required: No

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Java
- AWS SDK for Ruby V2

TopicsDetectionJobFilter

Provides information for filtering topic detection jobs. For more information, see ListTopicsDetectionJobs (p. 84).

Contents

JobName

Type: String

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

Pattern: `^([\p{L}\p{Z}\p{N}_.:/=+\-%@]*)$`

Required: No

JobStatus

Filters the list of topic detection jobs based on job status. Returns only jobs with the specified status.

Type: String

Valid Values: `SUBMITTED | IN_PROGRESS | COMPLETED | FAILED`

Required: No

SubmitTimeAfter

Filters the list of jobs based on the time that the job was submitted for processing. Only returns jobs submitted after the specified time. Jobs are returned in ascending order, oldest to newest.

Type: Timestamp

Required: No

SubmitTimeBefore

Filters the list of jobs based on the time that the job was submitted for processing. Only returns jobs submitted before the specified time. Jobs are returned in descending order, newest to oldest.

Type: Timestamp

Required: No

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Java
- AWS SDK for Ruby V2

TopicsDetectionJobProperties

Provides information about a topic detection job.

Contents

EndTime

The time that the topic detection job was completed.

Type: Timestamp

Required: No

InputDataConfig

The input data configuration supplied when you created the topic detection job.

Type: InputDataConfig (p. 99) object

Required: No

JobId

The identifier assigned to the topic detection job.

Type: String

Length Constraints: Minimum length of 1. Maximum length of 32.

Required: No

JobName

The name of the topic detection job.

Type: String

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

Pattern: `^([\p{L}\p{Z}\p{N}_.:/=+\-%@]*)$`

Required: No

JobStatus

The current status of the topic detection job. If the status is `Failed`, the reason for the failure is shown in the `Message` field.

Type: String

Valid Values: `SUBMITTED | IN_PROGRESS | COMPLETED | FAILED`

Required: No

Message

A description for the status of a job.

Type: String

Required: No

NumberOfTopics

The number of topics to detect supplied when you created the topic detection job. The default is 10.

Type: Integer

Required: No

OutputDataConfig

The output data configuration supplied when you created the topic detection job.

Type: OutputDataConfig (p. 101) object

Required: No

SubmitTime

The time that the topic detection job was submitted for processing.

Type: Timestamp

Required: No

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Java
- AWS SDK for Ruby V2

Common Errors

This section lists the errors common to the API actions of all AWS services. For errors specific to an API action for this service, see the topic for that API action.

AccessDeniedException

You do not have sufficient access to perform this action.

HTTP Status Code: 400

IncompleteSignature

The request signature does not conform to AWS standards.

HTTP Status Code: 400

InternalFailure

The request processing has failed because of an unknown error, exception or failure.

HTTP Status Code: 500

InvalidAction

The action or operation requested is invalid. Verify that the action is typed correctly.

HTTP Status Code: 400

InvalidClientTokenId

The X.509 certificate or AWS access key ID provided does not exist in our records.

HTTP Status Code: 403

InvalidParameterCombination

Parameters that must not be used together were used together.

HTTP Status Code: 400

InvalidParameterValue

An invalid or out-of-range value was supplied for the input parameter.

HTTP Status Code: 400

InvalidQueryParameter

The AWS query string is malformed or does not adhere to AWS standards.

HTTP Status Code: 400

MalformedQueryString

The query string contains a syntax error.

HTTP Status Code: 404

MissingAction

The request is missing an action or a required parameter.

HTTP Status Code: 400

MissingAuthenticationToken

The request must contain either a valid (registered) AWS access key ID or X.509 certificate.

HTTP Status Code: 403

MissingParameter

A required parameter for the specified action is not supplied.

HTTP Status Code: 400

OptInRequired

The AWS access key ID needs a subscription for the service.

HTTP Status Code: 403

RequestExpired

The request reached the service more than 15 minutes after the date stamp on the request or more than 15 minutes after the request expiration date (such as for pre-signed URLs), or the date stamp on the request is more than 15 minutes in the future.

HTTP Status Code: 400

ServiceUnavailable

The request has failed due to a temporary failure of the server.

HTTP Status Code: 503

ThrottlingException

The request was denied due to request throttling.

HTTP Status Code: 400

ValidationError

The input fails to satisfy the constraints specified by an AWS service.

HTTP Status Code: 400

Common Parameters

The following list contains the parameters that all actions use for signing Signature Version 4 requests with a query string. Any action-specific parameters are listed in the topic for that action. For more information about Signature Version 4, see Signature Version 4 Signing Process in the *Amazon Web Services General Reference*.

Action

The action to be performed.

Type: string

Required: Yes

Version

The API version that the request is written for, expressed in the format YYYY-MM-DD.

Type: string

Required: Yes

X-Amz-Algorithm

The hash algorithm that you used to create the request signature.

Condition: Specify this parameter when you include authentication information in a query string instead of in the HTTP authorization header.

Type: string

Valid Values: `AWS4-HMAC-SHA256`

Required: Conditional

X-Amz-Credential

The credential scope value, which is a string that includes your access key, the date, the region you are targeting, the service you are requesting, and a termination string ("aws4_request"). The value is expressed in the following format: *access_key*/*YYYYMMDD*/*region*/*service*/aws4_request.

For more information, see Task 2: Create a String to Sign for Signature Version 4 in the *Amazon Web Services General Reference*.

Condition: Specify this parameter when you include authentication information in a query string instead of in the HTTP authorization header.

Type: string

Required: Conditional

X-Amz-Date

The date that is used to create the signature. The format must be ISO 8601 basic format (YYYYMMDD'T'HHMMSS'Z'). For example, the following date time is a valid X-Amz-Date value: `20120325T120000Z`.

Condition: X-Amz-Date is optional for all requests; it can be used to override the date used for signing requests. If the Date header is specified in the ISO 8601 basic format, X-Amz-Date is not required. When X-Amz-Date is used, it always overrides the value of the Date header. For more information, see Handling Dates in Signature Version 4 in the *Amazon Web Services General Reference*.

Type: string

Required: Conditional

X-Amz-Security-Token

The temporary security token that was obtained through a call to AWS Security Token Service (AWS STS). For a list of services that support temporary security credentials from AWS Security Token Service, go to AWS Services That Work with IAM in the *IAM User Guide*.

Condition: If you're using temporary security credentials from the AWS Security Token Service, you must include the security token.

Type: string

Required: Conditional

X-Amz-Signature

Specifies the hex-encoded signature that was calculated from the string to sign and the derived signing key.

Condition: Specify this parameter when you include authentication information in a query string instead of in the HTTP authorization header.

Type: string

Required: Conditional

X-Amz-SignedHeaders

Specifies all the HTTP headers that were included as part of the canonical request. For more information about specifying signed headers, see Task 1: Create a Canonical Request For Signature Version 4 in the *Amazon Web Services General Reference*.

Condition: Specify this parameter when you include authentication information in a query string instead of in the HTTP authorization header.

Type: string

Required: Conditional

Document History for Amazon Comprehend

The following table describes the documentation for this release of Amazon Comprehend.

- **Latest documentation update:**November 29, 2017

Change	Description	Date
New guide	This is the first release of the *Amazon Comprehend Developer Guide*.	November 29, 2017