aws

Developer Guide

Amazon Transcribe



Copyright © 2026 Amazon Web Services, Inc. and/or its affiliates. All rights reserved.

Amazon Transcribe: Developer Guide

Copyright © 2026 Amazon Web Services, Inc. and/or its affiliates. All rights reserved.

Amazon's trademarks and trade dress may not be used in connection with any product or service that is not Amazon's, in any manner that is likely to cause confusion among customers, or in any manner that disparages or discredits Amazon. All other trademarks not owned by Amazon are the property of their respective owners, who may or may not be affiliated with, connected to, or sponsored by Amazon.

Services or capabilities described in AWS documentation might vary by Region. To see the differences applicable to the AWS European Sovereign Cloud Region, see the AWS European Sovereign Cloud User Guide.

Table of Contents

What is Amazon Transcribe? 1
Amazon Transcribe and HIPAA eligibility 1
Pricing 2
Region availability and quotas 2
Supported languages 4
Supported programming languages 15
Character sets 16
Abkhaz 20
Afrikaans 22
Arabic 23
Asturian 24
Azerbaijani 25
Armenian 25
Bashkir 26
Basque 29
Belarusian 29
Bengali 30
Bosnian 32
Bulgarian 33
Catalan 34
Central Kurdish 35
Chinese, Simplified 36
Chinese, Traditional 37
Chinese, Cantonese 38
Croatian 38
Czech 39
Danish 40
Dutch 40
English 41
Estonian 41
Farsi 42
Finnish 43
French 44
Galician 45

Georgian 45
German 47
Greek 47
Gujarati 48
Hausa 50
Hebrew 51
Hindi 52
Hungarian 54
Icelandic 55
Indonesian 55
Italian 55
Japanese 56
Kabyle 57
Kannada 57
Kazakh 59
Kinyarwanda 61
Korean 62
Kyrgyz 62
Latvian 64
Lithuanian 65
Luganda 66
Macedonian 66
Malay 68
Malayalam 69
Maltese 71
Marathi 71
Meadow Mari 73
Mongolian 76
Norwegian Bokmål 78
Odia/Oriya 79
Pashto 80
Polish 82
Portuguese 83
Punjabi 84
Romanian 86
Russian 87

Serbian ... 88
Sinhala ... 90
Slovak ... 92
Slovenian ... 93
Somali ... 94
Spanish ... 94
Sundanese ... 95
Swahili ... 95
Swedish ... 96
Tagalog/Filipino ... 97
Tamil ... 97
Tatar ... 98
Telugu ... 101
Thai ... 103
Turkish ... 105
Ukrainian ... 106
Uyghur ... 107
Uzbek ... 109
Vietnamese ... 110
Welsh ... 115
Wolof ... 116
Zulu ... 116
How it works ... 118
Data input and output ... 118
Media formats ... 119
Audio channels ... 120
Sample rates ... 121
Output ... 121
Transcribing numbers ... 124
Getting started ... 126
Sign up for an AWS account ... 127
Installing the AWS CLI and SDKs ... 127
Creating an Amazon S3 bucket ... 127
Creating an IAM policy ... 128
Transcribing with the AWS Management Console ... 130
Transcribing with the AWS CLI ... 138

Starting a new transcription job ... 139
Getting the status of a transcription job ... 140
Listing your transcription jobs ... 141
Deleting your transcription job ... 142
Transcribing with the AWS SDKs ... 142
Working with AWS SDKs ... 154
Transcribing with HTTP or WebSockets ... 154
Streaming transcriptions ... 157
Best practices ... 158
Handling LimitExceededException errors ... 159
Streaming and partial results ... 159
Partial-result stabilization ... 161
Setting up a streaming transcription ... 165
Event stream encoding ... 179
Data frames ... 181
Job queueing ... 183
Enabling job queueing ... 183
Tagging resources ... 187
Tag-based access control ... 188
Adding tags to your Amazon Transcribe resources ... 189
Partitioning speakers (diarization) ... 193
Partitioning speakers in a batch transcription ... 194
Partitioning speakers in a streaming transcription ... 197
Example output ... 199
Transcribing multi-channel audio ... 206
Using channel identification in a batch transcription ... 207
Using channel identification in a streaming transcription ... 211
Example output ... 212
Identifying languages ... 221
Batch language identification ... 221
Identifying languages in multi-language audio ... 222
Improving language identification accuracy ... 223
Combining language identification with other Amazon Transcribe features ... 224
Using language identification with batch transcriptions ... 225
Streaming language identification ... 231
Identifying languages in multi-language audio ... 233

Using language identification with streaming media ... 233
Alternative transcriptions ... 239
Requesting alternative transcriptions ... 241
Improving transcription accuracy ... 246
Custom vocabularies ... 247
Custom vocabulary tables versus lists ... 248
Creating a custom vocabulary using a table ... 249
Creating a custom vocabulary using a list ... 259
Using a custom vocabulary ... 263
Updating a custom vocabulary ... 270
Custom language models ... 271
Data sources ... 271
Training versus tuning data ... 272
Creating a custom language model ... 273
Using a custom language model ... 278
Changing encryption configuration for a custom language model ... 285
Filtering words ... 287
Creating a vocabulary filter ... 288
Creating custom vocabulary filters ... 289
Using a custom vocabulary filter ... 293
Using a custom vocabulary filter in a batch transcription ... 263
Using a custom vocabulary filter in a streaming transcription ... 267
Updating a custom vocabulary filter ... 300
Detecting toxic speech ... 302
Using toxic speech detection ... 303
Using toxic speech detection in a batch transcription ... 303
Example output ... 307
Redacting transcripts ... 310
Redacting PII in your batch job ... 311
Redacting or identifying PII in a real-time stream ... 319
Example output ... 326
Example redacted output (batch) ... 326
Example redacted streaming output ... 329
Example PII identification output ... 330
Creating subtitles ... 333
Generating subtitle files ... 334

Analyzing call center audio ... 337
Common use cases ... 337
Considerations and additional information ... 339
Region availability and quotas ... 340
Post-call analytics ... 341
Post-call insights ... 341
Creating categories ... 344
Starting a transcription ... 355
Post-call analytics output ... 362
Enabling generative call summarization ... 375
Real-time Call Analytics ... 379
Real-time insights ... 380
Creating categories ... 384
Post-call analytics with real-time transcriptions ... 390
Starting a transcription ... 397
Real-time Call Analytics output ... 405
Transcribing Amazon Chime calls ... 411
Code examples ... 412
Basics ... 413
Actions ... 413
Scenarios ... 484
Build an Amazon Transcribe streaming app ... 484
Convert text to speech and back to text ... 485
Create and refine a custom vocabulary ... 486
Transcribe audio and get job data ... 496
Security ... 507
Identity and Access Management ... 508
Audience ... 508
Authenticating with identities ... 508
Managing access using policies ... 510
How Amazon Transcribe works with IAM ... 511
Confused deputy prevention ... 517
Identity-based policy examples ... 517
Troubleshooting ... 529
Data protection ... 531
Inter-network traffic privacy ... 532

Data encryption ... 532
Opting out of using your data for service improvement ... 537
Monitoring Amazon Transcribe ... 537
Monitoring with CloudWatch ... 538
Monitoring Amazon Transcribe with CloudTrail ... 539
Using Amazon EventBridge with Amazon Transcribe ... 543
Compliance validation ... 551
Resilience ... 551
Infrastructure security ... 552
Vulnerability analysis and management ... 552
VPC endpoints (AWS PrivateLink) ... 552
Shared subnets ... 554
Security best practices ... 555
Amazon Transcribe Medical ... 557
Region availability and quotas ... 558
Medical specialties ... 559
Transcribing medical terms and measurements ... 560
Transcribing numbers ... 563
Transcribing a medical conversation ... 565
Transcribing an audio file ... 566
Transcribing a real-time stream ... 570
Enabling speaker partitioning ... 573
Transcribing multi-channel audio ... 583
Transcribing a medical dictation ... 591
Transcribing an audio file ... 592
Transcribing a streaming medical dictation ... 596
Creating and using medical custom vocabularies ... 599
Creating a text file for your medical custom vocabulary ... 600
Using a text file to create a medical custom vocabulary ... 604
Transcribing an audio file using a medical custom vocabulary ... 606
Transcribing a real-time stream using a medical custom vocabulary ... 608
Character set for Amazon Transcribe Medical ... 611
Identifying PHI in a transcript ... 612
Identifying PHI in an audio file ... 614
Identifying PHI in a real-time stream ... 618
Generating alternative transcriptions ... 620

VPC endpoints (AWS PrivateLink) ... 623
Considerations for Amazon Transcribe Medical VPC endpoints ... 623
Creating an interface VPC endpoint for Amazon Transcribe Medical ... 623
Creating a VPC endpoint policy for Amazon Transcribe Medical streaming ... 624
Shared subnets ... 625
AWS HealthScribe ... 626
Security ... 627
Service availability ... 627
Technical requirements ... 627
Supported Medical Specialties ... 628
Workflows ... 629
Transcript file ... 629
Clinical Documentation file ... 630
Medical Scribe context ... 631
HISTORY_AND_PHYSICAL template sections ... 632
GIRPP template sections ... 633
BIRP template sections ... 634
SIRP template sections ... 635
DAP template sections ... 635
BH_SOAP template sections ... 636
PH_SOAP template sections ... 636
Transcription jobs ... 637
Starting an AWS HealthScribe transcription job ... 637
Streaming ... 652
Guidelines and requirements ... 653
ResourceAccessRoleArn role permissions ... 653
Starting AWS HealthScribe streaming transcription ... 655
Data Encryption at rest for AWS HealthScribe ... 671
Creating a customer managed key ... 673
Specifying a customer managed key for AWS HealthScribe ... 676
AWS KMS encryption context ... 677
Monitoring your encryption keys for AWS HealthScribe ... 680
FAQs on supported protocols ... 683
What are the AWS JSON and Smithy RPC v2 CBOR protocols, and how do they differ? ... 683
How do I get started with the Smithy RPC v2 CBOR protocol? ... 683
What are the risks of enabling the Smithy RPC v2 CBOR protocol? ... 684

What if my solution does not support Smithy RPC v2 CBOR? 684
What AWS Regions support the Smithy RPC v2 CBOR protocol? 684
What latency improvements can I expect when upgrading to SDK versions that support Smithy RPC v2 CBOR? 684
Will the AWS JSON protocol be deprecated? 684
Where can I find more information about Smithy RPC v2 CBOR? 685
Document history 686
AWS Glossary 699

What is Amazon Transcribe?

Amazon Transcribe is an automatic speech recognition service that uses machine learning models to convert audio to text. You can use Amazon Transcribe as a standalone transcription service or to add speech-to-text capabilities to any application.

With Amazon Transcribe, you can improve accuracy for your specific use case with language customization, filter content to ensure customer privacy or audience-appropriate language, analyze content in multi-channel audio, partition the speech of individual speakers, and more.

You can transcribe media in real time (streaming) or you can transcribe media files located in an Amazon S3 bucket (batch). To see which languages are supported for each type of transcription, refer to the Supported languages and language-specific features table.

Topics

- Amazon Transcribe and HIPAA eligibility
- Pricing
- Region availability and quotas

See What is Amazon Transcribe? for a short video tour of this service.

To learn more, see How Amazon Transcribe works and Getting started with Amazon Transcribe.

> **Tip**
>
> Information on the **Amazon Transcribe API** is located in the API Reference.

Amazon Transcribe and HIPAA eligibility

Amazon Transcribe is covered under AWS's HIPAA eligibility and BAA which requires BAA customers to encrypt all PHI at rest and in transit when in use. Automatic PHI identification is available at no additional charge and in all regions where Amazon Transcribe operates. For more information, refer to HIPAA eligibility and BAA.

Pricing

Amazon Transcribe is a pay-as-you-go service; pricing is based on seconds of transcribed audio, billed on a monthly basis.

Usage is now charged in precise 1-second increments with no minimum duration applied. Note that additional charges apply for features such as PII content redaction and custom language models.

For cost information for each AWS Region, refer to Amazon Transcribe Pricing.

Region availability and quotas

Amazon Transcribe is supported in the following AWS Regions:

Region	Transcription type
af-south-1 (Cape Town)	batch, streaming
ap-east-1 (Hong Kong)	batch
ap-northeast-1 (Tokyo)	batch, streaming
ap-northeast-2 (Seoul)	batch, streaming
ap-south-1 (Mumbai)	batch, streaming
ap-southeast-1 (Singapore)	batch, streaming
ap-southeast-2 (Sydney)	batch, streaming
ap-southeast-5 (Malaysia)	streaming
ap-southeast-7 (Thailand)	streaming
ca-central-1 (Canada, Central)	batch, streaming
eu-central-1 (Frankfurt)	batch, streaming
eu-central-2 (Zurich)	batch, streaming
eu-north-1 (Stockholm)	batch

Region	Transcription type
eu-west-1 (Ireland)	batch, streaming
eu-west-2 (London)	batch, streaming
eu-west-3 (Paris)	batch
me-south-1 (Bahrain)	batch
mx-central-1 (Mexico)	streaming
sa-east-1 (São Paulo)	batch, streaming
us-east-1 (N. Virginia)	batch, streaming
us-east-2 (Ohio)	batch, streaming
us-gov-east-1 (GovCloud, US-East)	batch, streaming
us-gov-west-1 (GovCloud, US-West)	batch, streaming
us-west-1 (San Francisco)	batch
us-west-2 (Oregon)	batch, streaming

Important

Region support differs for Amazon Transcribe, Amazon Transcribe Medical, and Call Analytics.

To get the endpoints for each supported Region, see Service endpoints in the *AWS General Reference*.

For a list of quotas that pertain to your transcriptions, refer to the Service quotas in the *AWS General Reference*. Some quotas can be changed upon request. If the **Adjustable** column contains '**Yes**', you can request an increase. To do so, select the provided link.

Supported languages and language-specific features

The languages supported by Amazon Transcribe are listed in the following table; also listed are the features that are language-specific. Please verify that the feature you want to use is supported for the language in your media before proceeding with your transcription.

In the following table, 'batch' refers to transcribing a media file located in an Amazon S3 bucket and 'streaming' refers to transcribing streamed media in real time. For Call Analytics transcriptions, 'post-call' refers to transcribing a media file located in an Amazon S3 bucket and 'real-time' refers to transcribing streamed media in real time.

Language	Language code	Data input	Transcribing numbers	Acronyms	Custom language models	Redaction	Call Analytics *
Abkhaz	ab-GE	batch	no	batch	no	no	no
Afrikaans	af-ZA	batch, streaming	no	batch, streaming	no	no	no
Albanian	sq-AL	batch, streaming *	no	batch, streaming *	no	no	no
Amharic	am-ET	batch, streaming *	no	batch, streaming *	no	no	no
Arabic, Gulf	ar-AE	batch, streaming	batch, streaming	no	no	no	post-call
Arabic, Modern Standard	ar-SA	batch, streaming	streaming	no	no	no	no
Armenian	hy-AM	batch, streaming *	no	batch, streaming *	no	no	no

Language	Language code	Data input	Transcribing numbers	Acronyms	Custom language models	Redaction	Call Analytics *
Asturian	ast-ES	batch	no	batch	no	no	no
Azerbaijani	az-AZ	batch	no	batch	no	no	no
Bashkir	ba-RU	batch	no	batch	no	no	no
Basque	eu-ES	batch, streaming	no	batch, streaming	no	no	no
Belarusian	be-BY	batch, streaming *	no	batch, streaming *	no	no	no
Bengali	bn-IN	batch, streaming *	batch, streaming *	batch, streaming *	no	no	no
Bosnian	bs-BA	batch, streaming *	no	batch, streaming *	no	no	no
Burmese	my-MM	batch, streaming *	no	batch, streaming *	no	no	no
Bulgarian	bg-BG	batch, streaming *	no	batch, streaming *	no	no	no
Catalan	ca-ES	batch, streaming	streaming	batch, streaming	no	no	no

Language	Language code	Data input	Transcribing numbers	Acronyms	Custom language models	Redaction	Call Analytics *
Central Kurdish, Iran	ckb-IR	batch, streaming *	no	batch, streaming *	no	no	no
Central Kurdish, Iraq	ckb-IQ	batch, streaming *	no	batch, streaming *	no	no	no
Chinese, Cantonese	zh-HK (yue-HK)	batch, streaming	batch, streaming	no	no	no	no
Chinese, Simplified	zh-CN	batch, streaming	batch, streaming	no	no	no	post-call
Chinese, Traditional	zh-TW	batch, streaming	batch, streaming	no	no	no	no
Croatian	hr-HR	batch, streaming	no	batch, streaming	no	no	no
Czech	cs-CZ	batch, streaming	no	batch, streaming	no	no	no
Danish	da-DK	batch, streaming	batch, streaming	batch, streaming	no	no	no
Dutch	nl-NL	batch, streaming	batch, streaming	batch, streaming	no	no	no
English, Australian	en-AU	batch, streaming	batch, streaming	batch, streaming	batch, streaming	batch, streaming	post-call, real-time

Language	Language code	Data input	Transcribing numbers	Acronyms	Custom language models	Redaction	Call Analytics *
English, British	en-GB	batch, streaming	batch, streaming	batch, streaming	batch, streaming	batch, streaming	post-call, real-time
English, Indian	en-IN	batch, streaming	batch, streaming	batch, streaming	no	batch, streaming	post-call
English, Irish	en-IE	batch, streaming	batch, streaming	batch, streaming	no	batch, streaming	post-call
English, New Zealand	en-NZ	batch, streaming	batch, streaming	batch, streaming	no	batch, streaming	no
English, Scottish	en-AB	batch, streaming	batch, streaming	batch, streaming	no	batch, streaming	post-call
English, South African	en-ZA	batch, streaming	batch, streaming	batch, streaming	no	batch, streaming	no
English, US	en-US	batch, streaming	batch, streaming	batch, streaming	batch, streaming	batch, streaming	post-call, real-time
English, Welsh	en-WL	batch, streaming	batch, streaming	batch, streaming	no	batch, streaming	post-call
Estonian	et-EE	batch	no	batch	no	no	no
Estonian	et-ET	batch, streaming *	no	batch, streaming *	no	no	no
Farsi	fa-IR	batch, streaming	no	no	no	no	no

Language	Language code	Data input	Transcribing numbers	Acronyms	Custom language models	Redaction	Call Analytics *
Farsi, Afghan	fa-AF	batch, streaming *	no	batch, streaming *	no	no	no
Finnish	fi-FI	batch, streaming	no	batch, streaming	no	no	no
French	fr-FR	batch, streaming	batch, streaming	batch, streaming	no	batch, streaming	post-call, real-time
French, Canadian	fr-CA	batch, streaming	batch, streaming	batch, streaming	no	batch, streaming	post-call, real-time
Galician	gl-ES	batch, streaming	no	batch, streaming	no	no	no
Georgian	ka-GE	batch, streaming *	no	batch, streaming *	no	no	no
German	de-DE	batch, streaming	batch, streaming	batch, streaming	batch, streaming	batch, streaming	post-call, real-time
German, Swiss	de-CH	batch, streaming	batch, streaming	batch, streaming	no	batch, streaming	post-call
Greek	el-GR	batch, streaming	no	batch, streaming	no	no	no
Gujarati	gu-IN	batch, streaming *	no	batch, streaming *	no	no	no

Language	Language code	Data input	Transcribing numbers	Acronyms	Custom language models	Redaction	Call Analytics *
Haitian Creole	ht-HT	batch, streaming *	no	batch, streaming *	no	no	no
Hausa	ha-NG	batch	no	batch	no	no	no
Hebrew	he-IL	batch, streaming	batch, streaming	no	no	no	no
Hindi, Indian	hi-IN	batch, streaming	batch, streaming	batch, streaming	batch	no	post-call
Hungarian	hu-HU	batch, streaming *	no	batch, streaming *	no	no	no
Icelandic	is-IS	batch, streaming *	no	batch, streaming *	no	no	no
Indonesian	id-ID	batch, streaming	batch, streaming	batch, streaming	no	no	no
Italian	it-IT	batch, streaming	batch, streaming	batch, streaming	no	batch, streaming	post-call, real-time
Japanese	ja-JP	batch, streaming	batch, streaming	no	batch, streaming	no	post-call
Javanese	jv-ID	batch, streaming *	no	batch, streaming *	no	no	no

Language	Language code	Data input	Transcribing numbers	Acronyms	Custom language models	Redaction	Call Analytics *
Kabyle	kab-DZ	batch, streaming *	no	batch, streaming *	no	no	no
Kannada	kn-IN	batch, streaming *	no	batch, streaming *	no	no	no
Kazakh	kk-KZ	batch, streaming *	no	batch, streaming *	no	no	no
Khmer	km-KH	batch, streaming *	no	batch, streaming *	no	no	no
Kinyarwanda	rw-RW	batch	no	batch	no	no	no
Korean	ko-KR	batch, streaming	batch, streaming	batch, streaming	no	no	post-call
Kyrgyz	ky-KG	batch	no	batch	no	no	no
Latvian	lv-LV	batch, streaming	no	batch, streaming	no	no	no
Lithuanian	lt-LT	batch, streaming *	no	batch, streaming *	no	no	no
Luganda	lg-IN	batch, streaming *	no	batch, streaming *	no	no	no

Language	Language code	Data input	Transcribing numbers	Acronyms	Custom language models	Redaction	Call Analytics *
Macedonian	mk-MK	batch, streaming *	no	batch, streaming *	no	no	no
Malay	ms-MY	batch, streaming	no	batch, streaming	no	no	no
Malayalam	ml-IN	batch, streaming *	no	batch, streaming *	no	no	no
Maltese	mt-MT	batch	no	batch	no	no	no
Marathi	mr-IN	batch, streaming *	no	batch, streaming *	no	no	no
Meadow Mari	mhr-RU	batch	no	batch	no	no	no
Mongolian	mn-MN	batch	no	batch	no	no	no
Nepali	ne-NP	batch, streaming *	no	batch, streaming *	no	no	no
Norwegian Bokmål	no-NO	batch, streaming	batch	batch, streaming	no	no	no
Odia/Oriya	or-IN	batch, streaming *	no	batch, streaming *	no	no	no

Language	Language code	Data input	Transcribing numbers	Acronyms	Custom language models	Redaction	Call Analytics *
Pashto	ps-AF	batch, streaming *	no	batch, streaming *	no	no	no
Polish	pl-PL	batch, streaming	batch, streaming	batch, streaming	no	no	no
Portuguese	pt-PT	batch, streaming	batch, streaming	batch, streaming	no	batch, streaming	post-call
Portuguese, Brazilian	pt-BR	batch, streaming	batch, streaming	batch, streaming	no	batch, streaming	post-call, real-time
Punjabi	pa-IN	batch, streaming *	no	batch, streaming *	no	no	no
Romanian	ro-RO	batch, streaming	batch, streaming	batch, streaming	no	no	no
Russian	ru-RU	batch, streaming	batch, streaming	batch, streaming	no	no	no
Serbian	sr-RS	batch, streaming	no	batch, streaming	no	no	no
Sinhala	si-LK	batch, streaming *	no	batch, streaming *	no	no	no
Slovak	sk-SK	batch, streaming	no	batch, streaming	no	no	no

Language	Language code	Data input	Transcribing numbers	Acronyms	Custom language models	Redaction	Call Analytics *
Slovenian	sl-SI	batch, streaming *	no	batch, streaming *	no	no	no
Somali	so-SO	batch, streaming	batch, streaming	batch, streaming	no	no	no
Spanish	es-ES	batch, streaming	batch, streaming	batch, streaming	no	streaming	post-call
Spanish, Mexican	es-MX	batch, streaming *	streaming *	batch, streaming *	no	no	no
Spanish, US	es-US	batch, streaming	batch, streaming	batch, streaming	batch, streaming	batch, streaming	post-call, real-time
Sundanese	su-ID	batch, streaming *	no	batch, streaming *	no	no	no
Swahili, Kenya	sw-KE	batch, streaming *	batch, streaming *	batch, streaming *	no	no	no
Swahili, Burundi	sw-BI	batch, streaming *	batch	batch, streaming *	no	no	no
Swahili, Rwanda	sw-RW	batch, streaming *	batch	batch, streaming *	no	no	no

Language	Language code	Data input	Transcribing numbers	Acronyms	Custom language models	Redaction	Call Analytics *
Swahili, Tanzania	sw-TZ	batch, streaming *	batch	batch, streaming *	no	no	no
Swahili, Uganda	sw-UG	batch, streaming *	batch	batch, streaming *	no	no	no
Swedish	sv-SE	batch, streaming	batch, streaming	batch, streaming	no	no	no
Tagalog/ Filipino	tl-PH	batch, streaming	batch, streaming	batch, streaming	no	no	no
Tamil	ta-IN	batch, streaming *	batch, streaming *	streaming *	no	no	no
Tatar	tt-RU	batch	no	batch	no	no	no
Telugu	te-IN	batch, streaming *	no	streaming *	no	no	no
Thai	th-TH	batch, streaming	batch	batch, streaming	no	no	no
Turkish	tr-TR	batch, streaming *	batch, streaming *	batch, streaming *	no	no	no
Ukrainian	uk-UA	batch, streaming	batch, streaming	batch, streaming	no	no	no
Uyghur	ug-CN	batch	no	batch	no	no	no

Language	Language code	Data input	Transcribing numbers	Acronyms	Custom language models	Redaction	Call Analytics *
Uzbek	uz-UZ	batch, streaming *	no	batch, streaming *	no	no	no
Vietnamese	vi-VN	batch, streaming	batch	batch, streaming	no	no	no
Welsh	cy-WL	batch, streaming *	no	batch, streaming *	no	no	no
Wolof	wo-SN	batch	no	batch	no	no	no
Zulu	zu-ZA	batch, streaming	no	batch, streaming	no	no	no

*Streaming for these languages is not available in the following AWS Regions: af-south-1 (Cape Town), ap-northeast-1 (Tokyo), ap-southeast-5 (Malaysia), ap-southeast-7 (Thailand), and cn-northwest-1 (Ningxia).

*The following Call Analytics insights are only supported in select English dialects:

- Call summarization: en-* (all English dialects)
- Issue detection: en-AU, en-GB, en-US

Supported programming languages

Amazon Transcribe supports the following AWS SDKs:

Batch transcriptions	Streaming transcriptions
.NET	.NET (Transcribe Medical and HealthScribe is not supported)

Batch transcriptions	Streaming transcriptions
AWS Command Line Interface (CLI)	The CLI is not supported for streaming.
C++	C++
Go	Go
Java V2	Java V2
JavaScript	JavaScript V3
PHP V3	The SDK is not supported for streaming.
Python Boto3	Python Streaming SDK for Amazon Transcribe
Ruby V3	Ruby V3
Rust	Rust

For information on using SDKs with Amazon Transcribe, refer to Transcribing with the AWS SDKs.

For more information on all available AWS SDKs and builder tools, refer to Tools to Build on AWS.

Tip

You can find SDK code samples in these GitHub repositories:

- AWS Code Examples
- Amazon Transcribe Examples

Character sets for custom vocabularies and vocabulary filters

For each language Amazon Transcribe supports, there is a specific set of characters Amazon Transcribe can recognize. When you create a custom vocabulary or vocabulary filter, use only the characters listed in your language's character set. If you use unsupported characters, your custom vocabulary or vocabulary filter fails.

> **Important**
>
> Be sure to check that your custom vocabulary file uses only the supported Unicode code points and code point sequences listed within the following character sets.

Many Unicode characters can appear identical in popular fonts, even if they use different code points. **Only the code points listed in this guide are supported**. For example, the French word **déjà** can be rendered using *precomposed* characters (where one Unicode value represents an accented character) or *decomposed* characters (where two Unicode values represent an accented character, one value for the base character and another for the accent).

- **Precomposed version**: 0064 **`00E9`** 006A **`00E0`** (renders as **déjà**)
- **Decomposed version**: 0064 **`0065 0301`** 006A **`0061 0300`** (renders as **déjà**)

Topics

- Abkhaz character set
- Afrikaans character set
- Arabic character set
- Asturian character set
- Azerbaijani character set
- Armenian character set
- Bashkir character set
- Basque character set
- Belarusian character set
- Bengali character set
- Bosnian character set
- Bulgarian character set
- Catalan character set
- Central Kurdish character set
- Chinese, Mandarin (Mainland China), Simplified character set
- Chinese, Mandarin (Taiwan), Traditional character set
- Chinese, Cantonese (Hong Kong), Traditional character set

- Croatian character set
- Czech character set
- Danish character set
- Dutch character set
- English character set
- Estonian character set
- Farsi character set
- Finnish character set
- French character set
- Galician character set
- Georgian character set
- German character set
- Greek character set
- Gujarati character set
- Hausa character set
- Hebrew character set
- Hindi character set
- Hungarian character set
- Icelandic character set
- Indonesian character set
- Italian character set
- Japanese character set
- Kabyle character set
- Kannada character set
- Kazakh character set
- Kinyarwanda character set
- Korean character set
- Kyrgyz character set
- Latvian character set
- Lithuanian character set

- Luganda character set
- Macedonian character set
- Malay character set
- Malayalam character set
- Maltese character set
- Marathi character set
- Meadow Mari character set
- Mongolian character set
- Norwegian Bokmål character set
- Odia/Oriya character set
- Pashto character set
- Polish character set
- Portuguese character set
- Punjabi character set
- Romanian character set
- Russian character set
- Serbian character set
- Sinhala character set
- Slovak character set
- Slovenian character set
- Somali character set
- Spanish character set
- Sundanese character set
- Swahili character set
- Swedish character set
- Tagalog/Filipino character set
- Tamil character set
- Tatar character set
- Telugu character set
- Thai character set

- Turkish character set
- Ukrainian character set
- Uyghur character set
- Uzbek character set
- Vietnamese character set
- Welsh character set
- Wolof character set
- Zulu character set

Abkhaz character set

For Abkhaz custom vocabularies, you can use the following characters in the `Phrase` field:

- a - z
- - (hyphen)
- . (period)

You can also use the following Unicode characters in the `Phrase` field:

Character	Code	Character	Code
а	0430	љ	0459
б	0431	њ	045A
в	0432	ћ	045B
г	0433	ќ	045C
д	0434	ѝ	045D
е	0435	ў	045E
ж	0436	џ	045F
з	0437	ґ	0491

Character	Code	Character	Code
и	0438	ҕ	0493
й	0439	җ	0497
к	043A	ҙ	0499
л	043B	қ	049B
м	043C	ҟ	049F
н	043D	ҡ	04A1
о	043E	ң	04A3
п	043F	ҥ	04A5
р	0440	ҩ	04A9
с	0441	ҫ	04AB
т	0442	ҭ	04AD
у	0443	ү	04AF
ф	0444	ұ	04B1
х	0445	ҳ	04B3
ц	0446	ҵ	04B5
ч	0447	ҷ	04B7
ш	0448	һ	04BB
щ	0449	ҽ	04BD
ъ	044A	ҿ	04BF
ы	044B	ӈ	04CA

Character	Code	Character	Code
ь	044C	ӑ	04D1
э	044D	ӓ	04D3
ю	044E	ӗ	04D7
я	044F	ә	04D9
è	0450	ӡ	04E1
ë	0451	ӣ	04E3
ђ	0452	ӧ	04E7
ѓ	0453	ө	04E9
є	0454	ӯ	04EF
s	0455	ӱ	04F1
i	0456	ӳ	04F3
ї	0457	ӷ	04F7
j	0458	ӹ	04F9
#	0525		

Afrikaans character set

For Afrikaans custom vocabularies, you can use the following characters in the `Phrase` field:

- a - z
- - (hyphen)
- . (period)

You can also use the following Unicode characters in the `Phrase` field:

Character	Code	Character	Code
á	00E1	ï	00EF
è	00E8	ó	00F3
é	00E9	ô	00F4
ê	00EA	ö	00F6
ë	00EB	ú	00FA
í	00ED	û	00FB
î	00EE	ü	00FC

Arabic character set

For Arabic custom vocabularies, you can use the following Unicode characters in the `Phrase` field. You can also use the hyphen (-) character to separate words.

Character	Code	Character	Code
ء	0621	س	0633
آ	0622	ش	0634
أ	0623	ص	0635
ؤ	0624	ض	0636
إ	0625	ط	0637
ئ	0626	ظ	0638
ا	0627	ع	0639
ب	0628	غ	063A
ة	0629	ف	0641

Character	Code	Character	Code
ت	062A	ق	0642
ث	062B	ك	0643
ج	062C	ل	0644
ح	062D	م	0645
خ	062E	ن	0646
د	062F	ه	0647
ذ	0630	و	0648
ر	0631	ى	0649
ز	0632	ي	064A

Asturian character set

For Asturian custom vocabularies, you can use the following characters in the `Phrase` field:

- a - z
- - (hyphen)
- . (period)

You can also use the following Unicode characters in the `Phrase` field:

Character	Code	Character	Code
á	00E1	ñ	00F1
é	00E9	ó	00F3
í	00ED	ú	00FA
ü	00FC		

Azerbaijani character set

For Azerbaijani custom vocabularies, you can use the following characters in the `Phrase` field:

- a - z
- - (hyphen)
- . (period)

You can also use the following Unicode characters in the `Phrase` field:

Character	Code	Character	Code
ä	00E4	ğ	011F
ç	00E7	ı	0131
ö	00F6	ş	015F
ü	00FC	ə	0259
̇	0307		

Armenian character set

For Armenian custom vocabularies, you can use the following characters in the `Phrase` field:

- a - z
- - (hyphen)
- . (period)

You can also use the following Unicode characters in the `Phrase` field:

Character	Code	Character	Code
ɯ	0561	ɻ	0574
ʰ	0562	ɾ	0575

Character	Code	Character	Code
ɋ	0563	ɕ	0576
ŋ	0564	ɕ	0577
ɦ	0565	ɲ	0578
ɋ	0566	ɔ	0579
ɬ	0567	ɯ	057A
ɭ	0568	ɵ	057B
ɒ	0569	ɲ	057C
ԁ	056A	ɯ	057D
հ	056B	ʋ	057E
l	056C	ɯ	057F
խ	056D	ɪ	0580
ծ	056E	ɡ	0581
կ	056F	ɪ	0582
h	0570	ɸ	0583
ձ	0571	f	0584
ղ	0572	o	0585
ɦ	0573	ɸ	0586

Bashkir character set

For Bashkir custom vocabularies, you can use the following characters in the `Phrase` field:

- a - z

- - (hyphen)
- . (period)

You can also use the following Unicode characters in the `Phrase` field:

Character	Code	Character	Code
а	0430	љ	0459
б	0431	њ	045A
в	0432	ћ	045B
г	0433	ќ	045C
д	0434	ѝ	045D
е	0435	ў	045E
ж	0436	џ	045F
з	0437	ґ	0491
и	0438	ғ	0493
й	0439	ҷ	0497
к	043A	ҙ	0499
л	043B	қ	049B
м	043C	ҟ	049F
н	043D	ҡ	04A1
о	043E	ң	04A3
п	043F	ҥ	04A5
р	0440	ҩ	04A9

Character	Code	Character	Code
с	0441	ҫ	04AB
т	0442	ҭ	04AD
у	0443	ү	04AF
ф	0444	ұ	04B1
х	0445	ҳ	04B3
ц	0446	ҵ	04B5
ч	0447	ҷ	04B7
ш	0448	һ	04BB
щ	0449	ҽ	04BD
ъ	044A	ҿ	04BF
ы	044B	ӎ	04CA
ь	044C	ӑ	04D1
э	044D	ӓ	04D3
ю	044E	ӗ	04D7
я	044F	ә	04D9
è	0450	ӟ	04E1
ё	0451	ӣ	04E3
ђ	0452	ӧ	04E7
ѓ	0453	ө	04E9
є	0454	ӯ	04EF

Character	Code	Character	Code
s	0455	ӱ	04F1
і	0456	ӳ	04F3
ї	0457	ҷ	04F7
ј	0458	ӹ	04F9

Basque character set

For Basque custom vocabularies, you can use the following characters in the `Phrase` field:

- a - z
- - (hyphen)
- . (period)

You can also use the following Unicode characters in the `Phrase` field:

Character	Code	Character	Code
á	00E1	ñ	00F1
é	00E9	ó	00F3
í	00ED	ú	00FA
ü	00FC		

Belarusian character set

For Belarusian custom vocabularies, you can use the following characters in the `Phrase` field:

- a - z
- - (hyphen)
- . (period)

You can also use the following Unicode characters in the `Phrase` field:

Character	Code	Character	Code
а	0430	с	0441
б	0431	т	0442
в	0432	у	0443
г	0433	ф	0444
д	0434	х	0445
е	0435	ц	0446
ж	0436	ч	0447
з	0437	ш	0448
й	0439	ы	044B
к	043A	ь	044C
л	043B	э	044D
м	043C	ю	044E
н	043D	я	044F
о	043E	ё	0451
п	043F	і	0456
р	0440	ў	045E

Bengali character set

For Bengali custom vocabularies, you can use the following characters in the `Phrase` field:

- a - z

- - (hyphen)
- . (period)

You can also use the following Unicode characters in the `Phrase` field:

Character	Code	Character	Code
ঁ	0981	দ	09A6
ং	0982	ধ	09A7
ঃ	0983	ন	09A8
অ	0985	প	09AA
আ	0986	ফ	09AB
ই	0987	ব	09AC
ঈ	0988	ভ	09AD
উ	0989	ম	09AE
ঊ	098A	য	09AF
ঋ	098B	র	09B0
এ	098F	ল	09B2
ঐ	0990	শ	09B6
ও	0993	ষ	09B7
ঔ	0994	স	09B8
ক	0995	হ	09B9
খ	0996	়	09BC
গ	0997	#	09BD

Character	Code	Character	Code
ঘ	0998	া	09BE
ঙ	0999	ি	09BF
চ	099A	ী	09C0
ছ	099B	ু	09C1
জ	099C	ূ	09C2
ঝ	099D	ৃ	09C3
ঞ	099E	ৄ	09C4
ট	099F	ে	09C7
ঠ	09A0	ৈ	09C8
ড	09A1	ো	09CB
ঢ	09A2	ৌ	09CC
ণ	09A3	্	09CD
ত	09A4	#	09CE
থ	09A5	ৗ	09D7

Bosnian character set

For Bosnian custom vocabularies, you can use the following characters in the `Phrase` field:

- a - z
- - (hyphen)
- . (period)

You can also use the following Unicode characters in the `Phrase` field:

Character	Code	Character	Code
ć	0107	đ	0111
č	010D	š	0161
ž	017E		

Bulgarian character set

For Bulgarian custom vocabularies, you can use the following characters in the `Phrase` field:

- a - z
- - (hyphen)
- . (period)

You can also use the following Unicode characters in the `Phrase` field:

Character	Code	Character	Code
а	0430	п	043F
б	0431	р	0440
в	0432	с	0441
г	0433	т	0442
д	0434	у	0443
е	0435	ф	0444
ж	0436	х	0445
з	0437	ц	0446
и	0438	ч	0447

Character	Code	Character	Code
й	0439	ш	0448
к	043A	щ	0449
л	043B	ъ	044A
м	043C	ь	044C
н	043D	ю	044E
о	043E	я	044F

Catalan character set

For Catalan custom vocabularies, you can use the following characters in the `Phrase` field:

- a - z
- - (hyphen)
- . (period)

You can also use the following Unicode characters in the `Phrase` field:

Character	Code	Character	Code
à	00E0	ï	00EF
ç	00E7	ò	00F2
è	00E8	ó	00F3
é	00E9	ú	00FA
í	00ED	ü	00FC
l·	0140		

Central Kurdish character set

For Central Kurdish custom vocabularies, you can use the following characters in the `Phrase` field:

- a - z
- - (hyphen)
- . (period)

You can also use the following Unicode characters in the `Phrase` field:

Character	Code	Character	Code
ئ	0626	م	0645
ا	0627	ن	0646
ب	0628	و	0648
ت	062A	پ	067E
ج	062C	چ	0686
ح	062D	ڕ	0695
خ	062E	ژ	0698
د	062F	ڤ	06A4
ر	0631	ک	06A9
ز	0632	گ	06AF
س	0633	ڵ	06B5
ش	0634	ھ	06BE
ع	0639	ۆ	06C6
غ	063A	ۇ	06C7

Character	Code	Character	Code
ف	0641	ى	06CC
ق	0642	ێ	06CE
ل	0644	ه	06D5

Chinese, Mandarin (Mainland China), Simplified character set

For Chinese (Simplified) custom vocabularies, the `Phrase` field can use any of the characters listed in the following file:

- zh-cn-character-set

The `SoundsLike` field can contain the pinyin syllables listed in the following file:

- pinyin-character-set

When you use pinyin syllables in the `SoundsLike` field, separate the syllables with a hyphen (-).

Amazon Transcribe represents the four tones in Chinese (Simplified) using numbers. The following table shows how tone marks are mapped for the word 'ma'.

Tone	Tone mark	Tone number
Tone 1	mā	ma1
Tone 2	má	ma2
Tone 3	mǎ	ma3
Tone 4	mà	ma4

> **Note**
>
> For the 5th (neutral) tone, you can use Tone 1, with the exception of 'er', which must be mapped to Tone 2. For example, 打转儿 would be represented as 'da3-zhuan4-er2'.

Chinese (Simplified) custom vocabularies don't use the `IPA` field, but you must still include the `IPA` header in the custom vocabulary table.

The following example is an input file in text format. The example uses spaces to align the columns. Your input files should use TAB characters to separate the columns. Include spaces only in the `DisplayAs` column.

```
Phrase       SoundsLike               IPA     DisplayAs
##          kang1-jian4
##          qian3-ze2
####        guo2-fang2-da4-chen2
#####      shi4-jie4-bo2-lan3-hui4          ###
```

Chinese, Mandarin (Taiwan), Traditional character set

For Chinese (Traditional) custom vocabularies, the `Phrase` field can use any of the characters listed in the following file:

- zh-tw-character-set

The `SoundsLike` field can contain the zhuyin syllables listed in the following file:

- zhuyin-character-set

When you use zhuyin syllables in the `SoundsLike` field, separate the syllables with a hyphen (-).

Amazon Transcribe represents the four tones in Chinese (Traditional) using numbers. The following table shows how tone marks are mapped for the word ㄇㄚ.

Tone	Tone mark	
Tone 1	ㄇㄚ	

Tone	Tone mark
Tone 2	ㄇㄚˊ
Tone 3	ㄇㄚˇ
Tone 4	ㄇㄚˋ

Chinese (Traditional) custom vocabularies don't use the `IPA` field, but you must still include the `IPA` header in the custom vocabulary table.

The following example is an input file in text format. The example uses spaces to align the columns. Your input files should use TAB characters to separate the columns. Include spaces only in the `DisplayAs` column.

```
Phrase        SoundsLike                    IPA        DisplayAs
##           ###`-##
##           ###ˇ-##ˊ
####        ###ˊ-##ˊ-##`-##ˇ
#####     #`-###`-##ˊ-##ˇ-###`            ###
```

Chinese, Cantonese (Hong Kong), Traditional character set

For Chinese (Cantonese) Hong Kong custom vocabularies, the `Phrase` field can use any of the characters listed in the following file:

- zh-hk-character-set

Croatian character set

For Croatian custom vocabularies, you can use the following characters in the `Phrase` field:

- a - z
- - (hyphen)
- . (period)

You can also use the following Unicode characters in the `Phrase` field:

Character	Code	Character	Code
ć	0107	đ	0111
č	010D	š	0161
ž	017E		

Czech character set

For Czech custom vocabularies, you can use the following characters in the `Phrase` field:

- a - z
- - (hyphen)
- . (period)

You can also use the following Unicode characters in the `Phrase` field:

Character	Code	Character	Code
á	00E1	ď	010F
é	00E9	ě	011B
í	00ED	ň	0148
ó	00F3	ř	0159
ú	00FA	š	0161
ý	00FD	ť	0165
č	010D	ů	016F
ž	017E		

Danish character set

For Danish custom vocabularies, you can use the following characters in the `Phrase` field:

- a - z
- A - Z
- - (hyphen)
- . (period)

You can also use the following Unicode characters in the `Phrase` field:

Character	Code	Character	Code
Å	00C5	æ	00E6
Æ	00C6	é	00E9
Ø	00D8	ø	00F8
å	00E5		

Dutch character set

For Dutch custom vocabularies, you can use the following characters in the `Phrase` field:

- a - z
- A - Z
- ' (apostrophe)
- - (hyphen)
- . (period)

You can also use the following Unicode characters in the `Phrase` field:

Character	Code	Character	Code
à	00E0	î	00EE
á	00E1	ï	00EF
â	00E2	ñ	00F1
ä	00E4	ò	00F2
ç	00E7	ó	00F3
è	00E8	ô	00F4
é	00E9	ö	00F6
ê	00EA	ù	00F9
ë	00EB	ú	00FA
ì	00EC	û	00FB
í	00ED	ü	00FC

English character set

For English custom vocabularies, you can use the following characters in the `Phrase` field:

- a - z
- A - Z
- ' (apostrophe)
- - (hyphen)
- . (period)

Estonian character set

For Estonian custom vocabularies, you can use the following characters in the `Phrase` field:

- a - z
- - (hyphen)
- . (period)

You can also use the following Unicode characters in the `Phrase` field:

Character	Code	Character	Code
ä	00E4	ü	00FC
õ	00F5	š	0161
ö	00F6	ž	017E

Farsi character set

For Farsi custom vocabularies, you can use the following characters in the `Phrase` field.

Character	Code	Character	Code
ء	0621	ظ	0638
آ	0622	ع	0639
أ	0623	غ	063A
ؤ	0624	ف	0641
ئ	0626	ق	0642
ا	0627	ل	0644
ب	0628	م	0645
ت	062A	ن	0646
ث	062B	ه	0647

Character	Code	Character	Code
ج	062C	و	0648
ح	062D	ـَ	064E
خ	062E	ـُ	064F
د	062F	ـِ	0650
ذ	0630	ـّ	0651
ر	0631	پ	067E
ز	0632	چ	0686
س	0633	ژ	0698
ش	0634	ک	06A9
ص	0635	گ	06AF
ض	0636	ی	06CC
ط	0637		

Finnish character set

For Finnish custom vocabularies, you can use the following characters in the `Phrase` field:

- a - z
- - (hyphen)
- . (period)

You can also use the following Unicode characters in the `Phrase` field:

Character	Code	Character	Code
ä	00E4	ö	00F6

Character	Code	Character	Code
å	00E5	š	0161
ž	017E		

French character set

For French custom vocabularies, you can use the following characters in the `Phrase` field:

- a - z
- A - Z
- ' (apostrophe)
- - (hyphen)
- . (period)

You can also use the following Unicode characters in the `Phrase` field:

Character	Code	Character	Code
À	00C0	à	00E0
Â	00C2	â	00E2
Ç	00C7	ç	00E7
È	00C8	è	00E8
É	00C9	é	00E9
Ê	00CA	ê	00EA
Ë	00CB	ë	00EB
Î	00CE	î	00EE
Ï	00CF	ï	00EF

Character	Code	Character	Code
Ô	00D4	ô	00F4
Ö	00D6	ö	00F6
Ù	00D9	ù	00F9
Û	00DB	û	00FB
Ü	00DC	ü	00FC

Galician character set

For Galician custom vocabularies, you can use the following characters in the `Phrase` field:

- a - z
- - (hyphen)
- . (period)

You can also use the following Unicode characters in the `Phrase` field:

Character	Code	Character	Code
á	00E1	ñ	00F1
é	00E9	ó	00F3
í	00ED	ú	00FA
ü	00FC		

Georgian character set

For Georgian custom vocabularies, you can use the following characters in the `Phrase` field:

- a - z

- - (hyphen)
- . (period)

You can also use the following Unicode characters in the `Phrase` field:

Character	Code	Character	Code
ა	10D0	რ	10E0
ბ	10D1	ს	10E1
გ	10D2	ტ	10E2
დ	10D3	უ	10E3
ე	10D4	ფ	10E4
ვ	10D5	ქ	10E5
ზ	10D6	ღ	10E6
თ	10D7	ყ	10E7
ი	10D8	შ	10E8
კ	10D9	ჩ	10E9
ლ	10DA	ც	10EA
მ	10DB	ძ	10EB
ნ	10DC	წ	10EC
ო	10DD	ჭ	10ED
პ	10DE	ხ	10EE
ჟ	10DF	ჯ	10EF
ჰ	10F0		

German character set

For German custom vocabularies, you can use the following characters in the `Phrase` field:

- a - z
- A - Z
- ' (apostrophe)
- - (hyphen)
- . (period)

You can also use the following Unicode characters in the `Phrase` field:

Character	Code	Character	Code
ä	00E4	Ä	00C4
ö	00F6	Ö	00D6
ü	00FC	Ü	00DC
ß	00DF		

Greek character set

For Greek custom vocabularies, you can use the following characters in the `Phrase` field:

- a - z
- - (hyphen)
- . (period)

You can also use the following Unicode characters in the `Phrase` field:

Character	Code	Character	Code
ά	03AC	ν	03BD

Character	Code	Character	Code
έ	03AD	ξ	03BE
ή	03AE	ο	03BF
ί	03AF	π	03C0
ΰ	03B0	ρ	03C1
α	03B1	ς	03C2
β	03B2	σ	03C3
γ	03B3	τ	03C4
δ	03B4	υ	03C5
ε	03B5	φ	03C6
ζ	03B6	χ	03C7
η	03B7	ψ	03C8
θ	03B8	ω	03C9
ι	03B9	ϊ	03CA
κ	03BA	ϋ	03CB
λ	03BB	ό	03CC
μ	03BC	ώ	03CE
ΐ	0390		

Gujarati character set

For Gujarati custom vocabularies, you can use the following characters in the `Phrase` field:

- a - z

- - (hyphen)
- . (period)

You can also use the following Unicode characters in the `Phrase` field:

Character	Code	Character	Code
ઁ	0A81	દ	0AA6
ં	0A82	ધ	0AA7
ઃ	0A83	ન	0AA8
અ	0A85	પ	0AAA
આ	0A86	ફ	0AAB
ઇ	0A87	બ	0AAC
ઈ	0A88	ભ	0AAD
ઉ	0A89	મ	0AAE
ઊ	0A8A	ય	0AAF
ઋ	0A8B	ર	0AB0
ઍ	0A8D	લ	0AB2
એ	0A8F	ળ	0AB3
ઐ	0A90	વ	0AB5
ઑ	0A91	શ	0AB6
ઓ	0A93	ષ	0AB7
ઔ	0A94	સ	0AB8
ક	0A95	હ	0AB9

Character	Code	Character	Code
ખ	0A96	઼	0ABC
ગ	0A97	ા	0ABE
ઘ	0A98	િ	0ABF
ઙ	0A99	ી	0AC0
ચ	0A9A	ુ	0AC1
છ	0A9B	ૂ	0AC2
જ	0A9C	ૃ	0AC3
ઝ	0A9D	ૅ	0AC5
ઞ	0A9E	ે	0AC7
ટ	0A9F	ૈ	0AC8
ઠ	0AA0	ૉ	0AC9
ડ	0AA1	ો	0ACB
ઢ	0AA2	ૌ	0ACC
ણ	0AA3	્	0ACD
ત	0AA4	ૐ	0AD0
થ	0AA5	ૠ	0AE0

Hausa character set

For Hausa custom vocabularies, you can use the following characters in the `Phrase` field:

- a - z
- - (hyphen)

- . (period)

You can also use the following Unicode characters in the `Phrase` field:

Character	Code	Character	Code
ƙ	0199	ɓ	0253
ƴ	01B4	ƈ	0257
~	0303		

Hebrew character set

For Hebrew custom vocabularies, you can use the following Unicode characters in the `Phrase` field:

Character	Code	Character	Code
-	002D	ם	05DD
א	05D0	מ	05DE
ב	05D1	ן	05DF
ג	05D2	נ	05E0
ד	05D3	ס	05E1
ה	05D4	ע	05E2
ו	05D5	ף	05E3
ז	05D6	פ	05E4
ח	05D7	ץ	05E5
ט	05D8	צ	05E6
י	05D9	ק	05E7

Character	Code	Character	Code
ך	05DA	ר	05E8
כ	05DB	ש	05E9
ל	05DC	ת	05EA

Hindi character set

For Hindi custom vocabularies, you can use the following Unicode characters in the `Phrase` field:

Character	Code	Character	Code
-	002D	थ	0925
.	002E	द	0926
ँ	0901	ध	0927
ं	0902	न	0928
ः	0903	प	092A
अ	0905	फ	092B
आ	0906	ब	092C
इ	0907	भ	092D
ई	0908	म	092E
उ	0909	य	092F
ऊ	090A	र	0930
ऋ	090B	ल	0932
ए	090F	व	0935

Character	Code	Character	Code
ऐ	0910	श	0936
ऑ	0911	ष	0937
ओ	0913	स	0938
औ	0914	ह	0939
क	0915	ा	093E
ख	0916	ि	093F
ग	0917	ी	0940
घ	0918	ु	0941
ङ	0919	ू	0942
च	091A	ृ	0943
छ	091B	ॅ	0945
ज	091C	ॉ	0947
झ	091D	ॊ	0948
ञ	091E	ॏ	0949
ट	091F	ो	094B
ठ	0920	ौ	094C
ड	0921	्	094D
ढ	0922	ज़	095B
ण	0923	ड़	095C
त	0924	ढ़	095D

Amazon Transcribe maps the following characters:

Character	Mapped to
ऩ (0929)	न (0928)
ऱ (0931)	र (0930)
क़ (0958)	क (0915)
ख़ (0959)	ख (0916)
ग़ (095A)	ग (0917)
फ़ (095E)	फ (092B)
य़ (095F)	य (092F)

Hungarian character set

For Hungarian custom vocabularies, you can use the following characters in the `Phrase` field:

- a - z
- - (hyphen)
- . (period)

You can also use the following Unicode characters in the `Phrase` field:

Character	Code	Character	Code
á	00E1	ö	00F6
é	00E9	ú	00FA
í	00ED	ü	00FC
ó	00F3	ő	0151
ű	0171		

Icelandic character set

For Icelandic custom vocabularies, you can use the following characters in the `Phrase` field:

- a - z
- - (hyphen)
- . (period)

You can also use the following Unicode characters in the `Phrase` field:

Character	Code	Character	Code
á	00E1	ú	00FA
é	00E9	ý	00FD
ð	00F0	þ	00FE
í	00ED	æ	00E6
ó	00F3	ö	00F6

Indonesian character set

For Indonesian custom vocabularies, you can use the following characters in the `Phrase` field:

- a - z
- A - Z
- ' (apostrophe)
- - (hyphen)
- . (period)

Italian character set

For Italian custom vocabularies, you can use the following characters in the `Phrase` field:

- a - z

- A - Z
- ' (apostrophe)
- - (hyphen)
- . (period)

You can also use the following Unicode characters in the `Phrase` field:

Character	Code	Character	Code
À	00C0	à	00E0
Ä	00C4	ä	00E4
Ç	00C7	ç	00E7
È	00C8	è	00E8
É	00C9	é	00E9
Ê	00CA	ê	00EA
Ë	00CB	ë	00EB
Ì	00CC	ì	00EC
Ò	00D2	ò	00F2
Ù	00D9	ù	00F9
Ü	00DC	ü	00FC

Japanese character set

For Japanese custom vocabularies, the `DisplayAs` field supports all hiragana, katakana, and kanji characters, and fullwidth romaji capital letters.

The `Phrase` field supports the characters listed in the following file:

- ja-jp-character-set

Kabyle character set

For Kabyle custom vocabularies, you can use the following characters in the `Phrase` field:

- a - z
- - (hyphen)
- . (period)

You can also use the following Unicode characters in the `Phrase` field:

Character	Code	Character	Code
ï	00EF	ḍ	1E0D
č	010D	ḥ	1E25
ř	0159	ṛ	1E5B
ǧ	01E7	ṣ	1E63
ɛ	025B	ṭ	1E6D
ɣ	0263	ẓ	1E93

Kannada character set

For Kannada custom vocabularies, you can use the following characters in the `Phrase` field:

- a - z
- - (hyphen)
- . (period)

You can also use the following Unicode characters in the `Phrase` field:

Character	Code	Character	Code
ಂ	0C82	ಧ	0CA7

Character	Code	Character	Code
ಃ	0C83	ನ	0CA8
ಅ	0C85	ಪ	0CAA
ಆ	0C86	ಫ	0CAB
ಇ	0C87	ಬ	0CAC
ಈ	0C88	ಭ	0CAD
ಉ	0C89	ಮ	0CAE
ಊ	0C8A	ಯ	0CAF
ಋ	0C8B	ರ	0CB0
ಎ	0C8E	ಲ	0CB2
ಏ	0C8F	ಳ	0CB3
ಐ	0C90	ವ	0CB5
ಒ	0C92	ಶ	0CB6
ಓ	0C93	ಷ	0CB7
ಔ	0C94	ಸ	0CB8
ಕ	0C95	ಹ	0CB9
ಖ	0C96	#	0CBC
ಗ	0C97	#	0CBD
ಘ	0C98	ಾ	0CBE
ಙ	0C99	ಿ	0CBF
ಚ	0C9A	ೀ	0CC0

Character	Code	Character	Code
ಛ	0C9B	ು	0CC1
ಜ	0C9C	ೂ	0CC2
ಝ	0C9D	ೃ	0CC3
ಞ	0C9E	ೆ	0CC6
ಟ	0C9F	ೇ	0CC7
ಠ	0CA0	ೈ	0CC8
ಡ	0CA1	ೊ	0CCA
ಢ	0CA2	ೋ	0CCB
ಣ	0CA3	ೌ	0CCC
ತ	0CA4	್	0CCD
ಥ	0CA5	ೕ	0CD5
ದ	0CA6	ೖ	0CD6
ೱ	0CE0		

Kazakh character set

For Kazakh custom vocabularies, you can use the following characters in the `Phrase` field:

- a - z
- - (hyphen)
- . (period)

You can also use the following Unicode characters in the `Phrase` field:

Character	Code	Character	Code
т	0442	ы	044B
б	0431	я	044F
о	043E	с	0441
п	043F	h	04BB
ш	0448	д	0434
и	0438	р	0440
ч	0447	г	0433
н	043D	ё	0451
қ	049B	й	0439
і	0456	ө	04E9
щ	0449	в	0432
е	0435	э	044D
ә	04D9	ң	04A3
ю	044E	л	043B
з	0437	ф	0444
х	0445	к	043A
ц	0446	у	0443
ү	04AF	ж	0436
м	043C	ғ	0493
ь	044C	а	0430

Character	Code	Character	Code
ъ	044A	ӳ	04B1

Kinyarwanda character set

For Kinyarwanda custom vocabularies, you can use the following characters in the `Phrase` field:

- a - z
- - (hyphen)
- . (period)

You can also use the following Unicode characters in the `Phrase` field:

Character	Code	Character	Code
á	00E1	ó	00F3
â	00E2	ô	00F4
ã	00E3	ú	00FA
ç	00E7	ü	00FC
è	00E8	ā	0101
é	00E9	ē	0113
ê	00EA	ī	012B
ë	00EB	ō	014D
í	00ED	ū	016B
ï	00EF	´	0301

Korean character set

For Korean custom vocabularies, you can use the following in the `Phrase` field:

- a - z
- A - Z
- - (hyphen)
- . (period)

The `Phrase` field can use any of the Hangul syllables listed in the following link:

- Hangul Syllables on Wikipedia.

Kyrgyz character set

For Kyrgyz custom vocabularies, you can use the following characters in the `Phrase` field:

- a - z
- - (hyphen)
- . (period)

You can also use the following Unicode characters in the `Phrase` field:

Character	Code	Character	Code
а	0430	љ	0459
б	0431	њ	045A
в	0432	ћ	045B
г	0433	ќ	045C
д	0434	ѝ	045D
е	0435	ў	045E

Character	Code	Character	Code
ж	0436	ҵ	045F
з	0437	ґ	0491
и	0438	ғ	0493
й	0439	ҕ	0497
к	043A	ҙ	0499
л	043B	қ	049B
м	043C	ҟ	049F
н	043D	ҡ	04A1
о	043E	ң	04A3
п	043F	ҥ	04A5
р	0440	ҩ	04A9
с	0441	ҫ	04AB
т	0442	ҭ	04AD
у	0443	ү	04AF
ф	0444	ұ	04B1
х	0445	ҳ	04B3
ц	0446	ҵ	04B5
ч	0447	ҷ	04B7
ш	0448	һ	04BB
щ	0449	ҽ	04BD

Character	Code	Character	Code
ъ	044A	ҏ	04BF
ы	044B	ӈ	04CA
ь	044C	ӑ	04D1
э	044D	ӓ	04D3
ю	044E	ӗ	04D7
я	044F	ә	04D9
è	0450	ӟ	04E1
ë	0451	ӣ	04E3
ђ	0452	ӧ	04E7
ѓ	0453	ө	04E9
є	0454	ӯ	04EF
s	0455	ӱ	04F1
i	0456	ӳ	04F3
ї	0457	ӷ	04F7
j	0458	ӹ	04F9

Latvian character set

For Latvian custom vocabularies, you can use the following characters in the `Phrase` field:

- a - z
- - (hyphen)
- . (period)

You can also use the following Unicode characters in the `Phrase` field:

Character	Code	Character	Code
ā	0101	ķ	0137
č	010D	ļ	013C
ē	0113	ņ	0146
ģ	0123	š	0161
ī	012B	ū	016B
ž	017E		

Lithuanian character set

For Lithuanian custom vocabularies, you can use the following characters in the `Phrase` field:

- a - z
- - (hyphen)
- . (period)

You can also use the following Unicode characters in the `Phrase` field:

Character	Code	Character	Code
ą	0105	į	012F
č	010D	š	0161
ę	0119	ų	0173
ė	0117	ū	016B
ž	017E		

Luganda character set

For Luganda custom vocabularies, you can use the following characters in the `Phrase` field:

- a - z
- - (hyphen)
- . (period)

You can also use the following Unicode characters in the `Phrase` field:

Character	Code	Character	Code
ÿ	00FF	ŋ	014B

Macedonian character set

For Macedonian custom vocabularies, you can use the following characters in the `Phrase` field:

- a - z
- - (hyphen)
- . (period)

You can also use the following Unicode characters in the `Phrase` field:

Character	Code	Character	Code
а	0430	љ	0459
б	0431	њ	045A
в	0432	ћ	045B
г	0433	ќ	045C
д	0434	ѝ	045D
е	0435	ў	045E

Character	Code	Character	Code
ж	0436	ҷ	045F
з	0437	ґ	0491
и	0438	ғ	0493
й	0439	җ	0497
к	043A	ҙ	0499
л	043B	қ	049B
м	043C	ҟ	049F
н	043D	ҡ	04A1
о	043E	ң	04A3
п	043F	ҥ	04A5
р	0440	ҩ	04A9
с	0441	ҫ	04AB
т	0442	ҭ	04AD
у	0443	ү	04AF
ф	0444	ұ	04B1
х	0445	ҳ	04B3
ц	0446	ҵ	04B5
ч	0447	ҷ	04B7
ш	0448	һ	04BB
щ	0449	ҽ	04BD

Character	Code	Character	Code
ъ	044A	ҏ	04BF
ы	044B	ӈ	04CA
ь	044C	ӑ	04D1
э	044D	ӓ	04D3
ю	044E	ӗ	04D7
я	044F	ә	04D9
è	0450	ӟ	04E1
ë	0451	ӣ	04E3
ђ	0452	ӧ	04E7
ѓ	0453	ө	04E9
є	0454	ӯ	04EF
s	0455	ӱ	04F1
i	0456	ӳ	04F3
ї	0457	ӷ	04F7
j	0458	ӹ	04F9

Malay character set

For Malay custom vocabularies, you can use the following characters in the `Phrase` field:

- a - z
- A - Z
- ' (apostrophe)
- - (hyphen)

- . (period)

Malayalam character set

For Malayalam custom vocabularies, you can use the following characters in the `Phrase` field:

- a - z
- - (hyphen)
- . (period)

You can also use the following Unicode characters in the `Phrase` field:

Character	Code	Character	Code
ം	0D02	ന	0D28
ഃ	0D03	പ	0D2A
അ	0D05	ഫ	0D2B
ആ	0D06	ബ	0D2C
ഇ	0D07	ഭ	0D2D
ഈ	0D08	മ	0D2E
ഉ	0D09	യ	0D2F
ഊ	0D0A	ര	0D30
ഋ	0D0B	റ	0D31
എ	0D0E	ല	0D32
ഏ	0D0F	ള	0D33
ഐ	0D10	ഴ	0D34
ഒ	0D12	വ	0D35

Character	Code	Character	Code
ഓ	0D13	ശ	0D36
ഔ	0D14	ഷ	0D37
ക	0D15	സ	0D38
ഖ	0D16	ഹ	0D39
ഗ	0D17	ാ	0D3E
ഘ	0D18	ി	0D3F
ങ	0D19	ീ	0D40
ച	0D1A	ു	0D41
ഛ	0D1B	ൂ	0D42
ജ	0D1C	ൃ	0D43
ഝ	0D1D	െ	0D46
ഞ	0D1E	േ	0D47
ട	0D1F	ൈ	0D48
ഠ	0D20	ൊ	0D4A
ഡ	0D21	ോ	0D4B
ഢ	0D22	ൌ	0D4C
ണ	0D23	്	0D4D
ത	0D24	#	0D7A
ഥ	0D25	#	0D7B
ദ	0D26	#	0D7C

Character	Code	Character	Code
ω	0D27	#	0D7D

Maltese character set

For Maltese custom vocabularies, you can use the following characters in the `Phrase` field:

- a - z
- - (hyphen)
- . (period)

You can also use the following Unicode characters in the `Phrase` field:

Character	Code	Character	Code
à	00E0	ù	00F9
è	00E8	ċ	010B
ì	00EC	ġ	0121
ò	00F2	ħ	0127
ż	017C		

Marathi character set

For Marathi custom vocabularies, you can use the following characters in the `Phrase` field:

- a - z
- - (hyphen)
- . (period)

You can also use the following Unicode characters in the `Phrase` field:

Character	Code	Character	Code
ँ	0901	थ	0925
ं	0902	द	0926
ः	0903	ध	0927
अ	0905	न	0928
आ	0906	प	092A
इ	0907	फ	092B
ई	0908	ब	092C
उ	0909	भ	092D
ऊ	090A	म	092E
ऋ	090B	य	092F
ऍ	090D	र	0930
ए	090F	ल	0932
ऐ	0910	ळ	0933
ऑ	0911	व	0935
ओ	0913	श	0936
औ	0914	ष	0937
क	0915	स	0938
ख	0916	ह	0939
ग	0917	़	093C
घ	0918	ा	093E

Character	Code	Character	Code
ङ	0919	ि	093F
च	091A	ी	0940
छ	091B	ु	0941
ज	091C	ू	0942
झ	091D	ृ	0943
ञ	091E	ॅ	0945
ट	091F	ॉ	0947
ठ	0920	ॊ	0948
ड	0921	ॏ	0949
ढ	0922	ो	094B
ण	0923	ौ	094C
त	0924	्	094D
ॐ	0950		

Meadow Mari character set

For Meadow Mari custom vocabularies, you can use the following characters in the `Phrase` field:

- a - z
- - (hyphen)
- . (period)

You can also use the following Unicode characters in the `Phrase` field:

Character	Code	Character	Code
а	0430	љ	0459
б	0431	њ	045A
в	0432	ћ	045B
г	0433	ќ	045C
д	0434	ѝ	045D
е	0435	ў	045E
ж	0436	џ	045F
з	0437	ґ	0491
и	0438	ғ	0493
й	0439	ҷ	0497
к	043A	ҙ	0499
л	043B	қ	049B
м	043C	ҟ	049F
н	043D	ҡ	04A1
о	043E	ң	04A3
п	043F	ҥ	04A5
р	0440	ҩ	04A9
с	0441	ҫ	04AB
т	0442	ҭ	04AD
у	0443	ү	04AF

Character	Code	Character	Code
ф	0444	ұ	04B1
х	0445	ҳ	04B3
ц	0446	ҵ	04B5
ч	0447	ҷ	04B7
ш	0448	һ	04BB
щ	0449	ҽ	04BD
ъ	044A	ҿ	04BF
ы	044B	ӎ	04CA
ь	044C	ӑ	04D1
э	044D	ӓ	04D3
ю	044E	ӗ	04D7
я	044F	ә	04D9
è	0450	ӟ	04E1
ё	0451	ӣ	04E3
ђ	0452	ӧ	04E7
ѓ	0453	ө	04E9
є	0454	ӯ	04EF
ѕ	0455	ӱ	04F1
і	0456	ӳ	04F3
ї	0457	ӷ	04F7

Character	Code	Character	Code
ј	0458	ӹ	04F9

Mongolian character set

For Mongolian custom vocabularies, you can use the following characters in the `Phrase` field:

- a - z
- - (hyphen)
- . (period)

You can also use the following Unicode characters in the `Phrase` field:

Character	Code	Character	Code
а	0430	љ	0459
б	0431	њ	045A
в	0432	ћ	045B
г	0433	ќ	045C
д	0434	ѝ	045D
е	0435	ў	045E
ж	0436	џ	045F
з	0437	ґ	0491
и	0438	ғ	0493
й	0439	ҷ	0497
к	043A	ҙ	0499
л	043B	қ	049B

Character	Code	Character	Code
м	043C	ҟ	049F
н	043D	ҡ	04A1
о	043E	ң	04A3
п	043F	ҥ	04A5
р	0440	ҩ	04A9
с	0441	ҫ	04AB
т	0442	ҭ	04AD
у	0443	ү	04AF
ф	0444	ұ	04B1
х	0445	ҳ	04B3
ц	0446	ҵ	04B5
ч	0447	ҷ	04B7
ш	0448	һ	04BB
щ	0449	ҽ	04BD
ъ	044A	ҿ	04BF
ы	044B	ӄ	04CA
ь	044C	ӑ	04D1
э	044D	ӓ	04D3
ю	044E	ӗ	04D7
я	044F	ә	04D9

Character	Code	Character	Code
è	0450	ҙ	04E1
ë	0451	ӣ	04E3
ђ	0452	ӧ	04E7
ѓ	0453	ө	04E9
є	0454	ӯ	04EF
ѕ	0455	ӱ	04F1
і	0456	ӳ	04F3
ї	0457	ӷ	04F7
ј	0458	ӹ	04F9

Norwegian Bokmål character set

For Norwegian Bokmål custom vocabularies, you can use the following characters in the `Phrase` field:

- a - z
- - (hyphen)
- . (period)

You can also use the following Unicode characters in the `Phrase` field:

Character	Code	Character	Code
å	00E5	æ	00E6
ø	00F8		

Odia/Oriya character set

For Odia/Oriya custom vocabularies, you can use the following characters in the `Phrase` field:

- a - z
- - (hyphen)
- . (period)

You can also use the following Unicode characters in the `Phrase` field:

Character	Code	Character	Code
ଁ	0B01	ଦ	0B26
ଂ	0B02	ଧ	0B27
ଃ	0B03	ନ	0B28
ଅ	0B05	ପ	0B2A
ଆ	0B06	ଫ	0B2B
ଇ	0B07	ବ	0B2C
ଈ	0B08	ଭ	0B2D
ଉ	0B09	ମ	0B2E
ଊ	0B0A	ଯ	0B2F
ଋ	0B0B	ର	0B30
ଏ	0B0F	ଲ	0B32
ଐ	0B10	ଳ	0B33
ଓ	0B13	ଗ	0B36
ଔ	0B14	ଷ	0B37

Character	Code	Character	Code
କ	0B15	ସ	0B38
ଖ	0B16	ହ	0B39
ଗ	0B17	଼	0B3C
ଘ	0B18	ଽ	0B3E
ଙ	0B19	ା	0B3F
ଚ	0B1A	ି	0B40
ଛ	0B1B	ୀ	0B41
ଜ	0B1C	ୁ	0B42
ଝ	0B1D	ୂ	0B43
ଞ	0B1E	େ	0B47
ଟ	0B1F	ୈ	0B48
ଠ	0B20	େ#	0B4B
ଡ	0B21	ୋ	0B4C
ଢ	0B22	୍	0B4D
ଣ	0B23	ୖ	0B56
ତ	0B24	ୟ	0B5F
ଥ	0B25	ୠ	0B60
#	0B71		

Pashto character set

For Pashto custom vocabularies, you can use the following characters in the `Phrase` field:

- a - z
- - (hyphen)
- . (period)

You can also use the following Unicode characters in the `Phrase` field:

Character	Code	Character	Code
آ	0622	و	0648
أ	0623	ي	064A
ؤ	0624	ً	064B
ئ	0626	ٌ	064C
ا	0627	ٍ	064D
ب	0628	َ	064E
ت	062A	ُ	064F
ث	062B	ِ	0650
ج	062C	ّ	0651
ح	062D	ْ	0652
خ	062E	#	0654
د	062F	ٰ	0670
ذ	0630	ټ	067C
ر	0631	پ	067E
ز	0632	ځ	0681
س	0633	څ	0685

Character	Code	Character	Code
ش	0634	چ	0686
ص	0635	ڈ	0689
ض	0636	ڑ	0693
ط	0637	ڔ	0696
ظ	0638	ژ	0698
ع	0639	ښ	069A
غ	063A	ک	06A9
ف	0641	گ	06AB
ق	0642	گ	06AF
ل	0644	ڼ	06BC
م	0645	ی	06CC
ن	0646	ۍ	06CD
ه	0647	ې	06D0

Polish character set

For Polish custom vocabularies, you can use the following characters in the `Phrase` field:

- a - z
- - (hyphen)
- . (period)

You can also use the following Unicode characters in the `Phrase` field:

Character	Code	Character	Code
ó	00F3	ł	0142
ą	0105	ń	0144
ć	0107	ś	015B
ę	0119	ź	017A
ż	017C		

Portuguese character set

For Portuguese custom vocabularies, you can use the following characters in the `Phrase` field:

- a - z
- A - Z
- ' (apostrophe)
- - (hyphen)
- . (period)

You can also use the following Unicode characters in the `Phrase` field:

Character	Code	Character	Code
À	00C0	à	00E0
Á	00C1	á	00E1
Â	00C2	â	00E2
Ã	00C3	ã	00E3
Ä	00C4	ä	00E4
Ç	00C7	ç	00E7

Character	Code	Character	Code
È	00C8	è	00E8
É	00C9	é	00E9
Ê	00CA	ê	00EA
Ë	00CB	ë	00EB
Í	00CD	í	00ED
Ñ	00D1	ñ	00F1
Ó	00D3	ó	00F3
Ô	00D4	ô	00F4
Õ	00D5	õ	00F5
Ö	00D6	ö	00F6
Ú	00DA	ú	00FA
Ü	00DC	ü	00FC

Punjabi character set

For Punjabi custom vocabularies, you can use the following characters in the `Phrase` field:

- a - z
- - (hyphen)
- . (period)

You can also use the following Unicode characters in the `Phrase` field:

Character	Code	Character	Code
ਅ	0A05	ਧ	0A27

Character	Code	Character	Code
ਆ	0A06	ਨ	0A28
ਇ	0A07	ਪ	0A2A
ਈ	0A08	ਫ	0A2B
ਉ	0A09	ਬ	0A2C
ਊ	0A0A	ਭ	0A2D
ਏ	0A0F	ਮ	0A2E
ਐ	0A10	ਯ	0A2F
ਓ	0A13	ਰ	0A30
ਔ	0A14	ਲ	0A32
ਕ	0A15	ਵ	0A35
ਖ	0A16	ਸ	0A38
ਗ	0A17	ਹ	0A39
ਘ	0A18	਼	0A3C
ਙ	0A19	ਾ	0A3E
ਚ	0A1A	ਿ	0A3F
ਛ	0A1B	ੀ	0A40
ਜ	0A1C	ੁ	0A41
ਝ	0A1D	ੂ	0A42
ਞ	0A1E	ੇ	0A47
ਟ	0A1F	ੈ	0A48

Character	Code	Character	Code
ਠ	0A20	੍	0A4B
ਡ	0A21	੍	0A4C
ਢ	0A22	੍	0A4D
ਣ	0A23	ੜ	0A5C
ਤ	0A24	ੰ	0A70
ਥ	0A25	ੱ	0A71
ਦ	0A26	ੲ	0A72
ਓ	0A73		

Romanian character set

For Romanian custom vocabularies, you can use the following characters in the `Phrase` field:

- a - z
- - (hyphen)
- . (period)

You can also use the following Unicode characters in the `Phrase` field:

Character	Code	Character	Code
ă	0103	ș	0219
â	00E2	ț	021B
î	00EE	ş	015F
ţ	0163		

Russian character set

For Russian custom vocabularies, you can use the following characters in the `Phrase` field:

Character	Code	Character	Code
'	0027	п	043F
-	002D	р	0440
.	002E	с	0441
а	0430	т	0442
б	0431	у	0443
в	0432	ф	0444
г	0433	х	0445
д	0434	ц	0446
е	0435	ч	0447
ж	0436	ш	0448
з	0437	щ	0449
и	0438	ъ	044A
й	0439	ы	044B
к	043A	ь	044C
л	043B	э	044D
м	043C	ю	044E
н	043D	я	044F
о	043E	ё	0451

Serbian character set

For Serbian custom vocabularies, you can use the following characters in the `Phrase` field:

- a - z
- - (hyphen)
- . (period)

You can also use the following Unicode characters in the `Phrase` field:

Character	Code	Character	Code
ć	0107	і	0456
č	010D	ї	0457
đ	0111	ј	0458
š	0161	љ	0459
ž	017E	њ	045A
а	0430	ћ	045B
б	0431	ќ	045C
в	0432	ѝ	045D
г	0433	ў	045E
д	0434	џ	045F
е	0435	ґ	0491
ж	0436	ғ	0493
з	0437	ҷ	0497
и	0438	ҙ	0499

Character	Code	Character	Code
й	0439	қ	049B
к	043A	ҟ	049F
л	043B	ҡ	04A1
м	043C	ң	04A3
н	043D	ҥ	04A5
о	043E	ҩ	04A9
п	043F	ҫ	04AB
р	0440	ҭ	04AD
с	0441	ү	04AF
т	0442	ұ	04B1
у	0443	ҳ	04B3
ф	0444	ҵ	04B5
х	0445	ҷ	04B7
ц	0446	һ	04BB
ч	0447	ҽ	04BD
ш	0448	ҿ	04BF
щ	0449	ӂ	04CA
ъ	044A	ӑ	04D1
ы	044B	ӓ	04D3
ь	044C	ӗ	04D7

Character	Code	Character	Code
э	044D	ә	04D9
ю	044E	ӡ	04E1
я	044F	ӣ	04E3
è	0450	ӧ	04E7
ë	0451	ө	04E9
ђ	0452	ӯ	04EF
ѓ	0453	ӱ	04F1
є	0454	ӳ	04F3
s	0455	ӷ	04F7
ьы	04F9		

Sinhala character set

For Sinhala custom vocabularies, you can use the following characters in the `Phrase` field:

- a - z
- - (hyphen)
- . (period)

You can also use the following Unicode characters in the `Phrase` field:

Character	Code	Character	Code
#	0D82	#	0DAF
#	0D83	#	0DB0
#	0D85	#	0DB1

Character	Code	Character	Code
#	0D86	#	0DB3
#	0D87	#	0DB4
#	0D88	#	0DB5
#	0D89	#	0DB6
#	0D8A	#	0DB7
#	0D8B	#	0DB8
#	0D8C	#	0DB9
#	0D8D	#	0DBA
#	0D91	#	0DBB
#	0D92	#	0DBD
#	0D93	#	0DC0
#	0D94	#	0DC1
#	0D95	#	0DC2
#	0D96	#	0DC3
#	0D9A	#	0DC4
#	0D9B	#	0DC5
#	0D9C	#	0DC6
#	0D9D	#	0DCA
#	0D9E	#	0DCF
#	0D9F	#	0DD0

Character	Code	Character	Code
#	0DA0	#	0DD1
#	0DA1	#	0DD2
#	0DA2	#	0DD3
#	0DA3	#	0DD4
#	0DA4	#	0DD6
#	0DA5	#	0DD8
#	0DA7	#	0DD9
#	0DA8	##	0DDA
#	0DA9	#	0DDB
#	0DAA	##	0DDC
#	0DAB	###	0DDD
#	0DAC	##	0DDE
#	0DAD	#	0DDF
#	0DAE	#	0DF2

Slovak character set

For Slovak custom vocabularies, you can use the following characters in the `Phrase` field:

- a - z
- - (hyphen)
- . (period)

You can also use the following Unicode characters in the `Phrase` field:

Character	Code	Character	Code
á	00E1	ň	0148
ä	00E4	ó	00F3
č	010D	ô	00F4
ď	010F	ŕ	0155
é	00E9	š	0161
í	00ED	ť	0165
Í	013A	ú	00FA
ľ	013E	ý	00FD
ž	017E		

Slovenian character set

For Slovenian custom vocabularies, you can use the following characters in the `Phrase` field:

- a - z
- - (hyphen)
- . (period)

You can also use the following Unicode characters in the `Phrase` field:

Character	Code	Character	Code
č	010D	š	0161
ž	017E		

Somali character set

For Somali custom vocabularies, you can use the following characters in the `Phrase` field:

- a - z
- - (hyphen)
- . (period)

You can also use the following Unicode characters in the `Phrase` field:

Character	Code	Character	Code
s	0073	d	0064
t	0074	a	0061
a	0061	r	0072
n	006E	d	0064

Spanish character set

For Spanish custom vocabularies, you can use the following characters in the `Phrase` field:

- a - z
- A - Z
- ' (apostrophe)
- - (hyphen)
- . (period)

You can also use the following Unicode characters in the `Phrase` field:

Character	Code	Character	Code
Á	00C1	á	00E1

Character	Code	Character	Code
É	00C9	é	00E9
Í	00CD	í	00ED
Ó	00D3	ó	0XF3
Ú	00DA	ú	00FA
Ñ	00D1	ñ	0XF1
ü	00FC		

Sundanese character set

For Sundanese custom vocabularies, you can use the following characters in the `Phrase` field:

- a - z
- - (hyphen)
- . (period)

You can also use the following Unicode characters in the `Phrase` field:

Character	Code	Character	Code
s	0073	d	0064
t	0074	a	0061
a	0061	r	0072
n	006E	d	0064

Swahili character set

For Swahili custom vocabularies, you can use the following characters in the `Phrase` field:

- a - z
- - (hyphen)
- . (period)

You can also use the following Unicode characters in the `Phrase` field:

Character	Code	Character	Code
s	0073	d	0064
t	0074	a	0061
a	0061	r	0072
n	006E	d	0064

Swedish character set

For Swedish custom vocabularies, you can use the following characters in the `Phrase` field:

- a - z
- A - Z
- ' (apostrophe)
- - (hyphen)
- . (period)

You can also use the following Unicode characters in the `Phrase` field:

Character	Code	Character	Code
Ä	00C4	ä	00E4
Å	00C5	å	00E5
Ö	00D6	ö	00F6

Tagalog/Filipino character set

For Tagalog/Filipino custom vocabularies, you can use the following characters in the `Phrase` field:

- a - z
- - (hyphen)
- . (period)

You can also use the following Unicode characters in the `Phrase` field:

Character	Code		
ñ	00F1		

Tamil character set

For Tamil custom vocabularies, you can use the following characters in the `Phrase` field:

Character	Code	Character	Code
அ	0B85	ர	0BB0
ஆ	0B86	ல	0BB2
இ	0B87	வ	0BB5
ஈ	0B88	ழ	0BB4
உ	0B89	ள	0BB3
ஊ	0B8A	ற	0BB1
எ	0B8E	ன	0BA9
ஏ	0B8F	ஜ	0B9C
ஐ	0B90	#	0BB6

Character	Code	Character	Code
ஒ	0B92	ஷ	0BB7
ஓ	0B93	ஸ	0BB8
ஔ	0B94	ஹ	0BB9
ஃ	0B83	◌்	0BCD
க	0B95	ா	0BBE
ங	0B99	ி	0BBF
ச	0B9A	ீ	0BC0
ஞ	0B9E	ு	0BC1
ட	0B9F	ூ	0BC2
ண	0BA3	ெ	0BC6
த	0BA4	ே	0BC7
ந	0BA8	ை	0BC8
ப	0BAA	ொ	0BCA
ம	0BAE	ோ	0BCB
ய	0BAF	ௌ	0BCC

Tatar character set

For Tatar custom vocabularies, you can use the following characters in the `Phrase` field:

- a - z
- - (hyphen)
- . (period)

You can also use the following Unicode characters in the `Phrase` field:

Character	Code	Character	Code
а	0430	љ	0459
б	0431	њ	045A
в	0432	ћ	045B
г	0433	ќ	045C
д	0434	ѝ	045D
е	0435	ў	045E
ж	0436	џ	045F
з	0437	ґ	0491
и	0438	ғ	0493
й	0439	ҷ	0497
к	043A	ҙ	0499
л	043B	қ	049B
м	043C	ҟ	049F
н	043D	ҡ	04A1
о	043E	ң	04A3
п	043F	ҥ	04A5
р	0440	ҩ	04A9
с	0441	ҫ	04AB
т	0442	ҭ	04AD

Character	Code	Character	Code
у	0443	ү	04AF
ф	0444	ұ	04B1
х	0445	ҳ	04B3
ц	0446	ҵ	04B5
ч	0447	ҷ	04B7
ш	0448	һ	04BB
щ	0449	ҽ	04BD
ъ	044A	ҿ	04BF
ы	044B	ӊ	04CA
ь	044C	ӑ	04D1
э	044D	ӓ	04D3
ю	044E	ӗ	04D7
я	044F	ә	04D9
ѐ	0450	ӟ	04E1
ё	0451	ӣ	04E3
ђ	0452	ӧ	04E7
ѓ	0453	ө	04E9
є	0454	ӯ	04EF
ѕ	0455	ӱ	04F1
і	0456	ӳ	04F3

Character	Code	Character	Code
ï	0457	ҷ	04F7
j	0458	ӹ	04F9

Telugu character set

For Telugu custom vocabularies, you can use the following characters in the `Phrase` field:

Character	Code	Character	Code
-	002D	త	0C24
ఁ	0C01	థ	0C25
ం	0C02	ద	0C26
ః	0C03	ధ	0C27
అ	0C05	న	0C28
ఆ	0C06	ప	0C2A
ఇ	0C07	ఫ	0C2B
ఈ	0C08	బ	0C2C
ఉ	0C09	భ	0C2D
ఊ	0C0A	మ	0C2E
ఋ	0C0B	య	0C2F
ర	0C30	ఎ	0C0E
ఱ	0C31	ఏ	0C0F
ల	0C32	ఐ	0C10

Character	Code	Character	Code
ళ	0C33	ఒ	0C12
వ	0C35	ఓ	0C13
శ	0C36	ఔ	0C14
ష	0C37	క	0C15
స	0C38	ఖ	0C16
హ	0C39	గ	0C17
ా	0C3E	ఘ	0C18
ి	0C3F	ఙ	0C19
ీ	0C40	చ	0C1A
ు	0C41	ఛ	0C1B
ూ	0C42	జ	0C1C
ృ	0C43	ఝ	0C1D
ౄ	0C44	ఞ	0C1E
ె	0C47	ట	0C1F
ే	0C48	ఠ	0C20
ై	0C4A	డ	0C21
ొ	0C4B	ఢ	0C22
ో	0C4C	ణ	0C23
్	0C4D		

Thai character set

For Thai custom vocabularies, you can use the following characters in the `Phrase` field:

- - (hyphen)
- . (period)

You can also use the following Unicode characters in the `Phrase` field:

Character	Code	Character	Code
ก	0E01	ล	0E25
ข	0E02	ฦ	0E26
ฃ	0E03	ว	0E27
ค	0E04	ศ	0E28
ฅ	0E05	ษ	0E29
ฆ	0E06	ส	0E2A
ง	0E07	ห	0E2B
จ	0E08	ฬ	0E2C
ฉ	0E09	อ	0E2D
ช	0E0A	ฮ	0E2E
ซ	0E0B	ฯ	0E2F
ฌ	0E0C	ะ	0E30
ญ	0E0D	ั	0E31
ฎ	0E0E	า	0E32
ฏ	0E0F	ิ	0E34

Character	Code	Character	Code
ฐ	0E10	◌ี	0E35
ฑ	0E11	◌ึ	0E36
ฒ	0E12	◌ื	0E37
ณ	0E13	◌ุ	0E38
ด	0E14	◌ู	0E39
ต	0E15	◌ฺ	0E3A
ถ	0E16	เ	0E40
ท	0E17	แ	0E41
ธ	0E18	โ	0E42
น	0E19	ใ	0E43
บ	0E1A	ไ	0E44
ป	0E1B	ๅ	0E45
ผ	0E1C	ๆ	0E46
ฝ	0E1D	◌็	0E47
พ	0E1E	◌่	0E48
ฟ	0E1F	◌้	0E49
ภ	0E20	◌๊	0E4A
ม	0E21	◌๋	0E4B
ย	0E22	◌์	0E4C
ร	0E23	◌ํ	0E4D

Character	Code	Character	Code
ฦ	0E24		

Turkish character set

For Turkish custom vocabularies, you can use the following characters in the `Phrase` field:

- a - z
- A - Z
- ' (apostrophe)
- - (hyphen)
- . (period)

You can also use the following Unicode characters in the `Phrase` field:

Character	Code	Character	Code
Ç	00C7	ö	00F6
Ö	00D6	û	00FB
Ü	00DC	ü	00FC
â	00E2	Ğ	011E
ä	00E4	ğ	011F
ç	00E7	İ	0130
è	00E8	ı	0131
é	00E9	Ş	015E
ê	00EA	ş	015F
í	00ED	š	0161

Character	Code	Character	Code
î	00EE	ž	017E
ó	00F3		

Ukrainian character set

For Ukrainian custom vocabularies, you can use the following characters in the `Phrase` field:

- a - z
- - (hyphen)
- . (period)

You can also use the following Unicode characters in the `Phrase` field:

Character	Code	Character	Code
а	0430	р	0440
б	0431	с	0441
в	0432	т	0442
г	0433	у	0443
д	0434	ф	0444
е	0435	х	0445
ж	0436	ц	0446
з	0437	ч	0447
и	0438	ш	0448
й	0439	щ	0449
к	043A	ь	044C

Character	Code	Character	Code
л	043B	ю	044E
м	043C	я	044F
н	043D	є	0454
о	043E	і	0456
п	043F	ї	0457
ґ	0491		

Uyghur character set

For Uyghur custom vocabularies, you can use the following characters in the `Phrase` field:

- a - z
- - (hyphen)
- . (period)

You can also use the following Unicode characters in the `Phrase` field:

Character	Code	Character	Code
#	0611	و	0648
#	0613	ى	0649
#	0614	ي	064A
ء	0621	ٙ	064B
آ	0622	ٚ	064C
أ	0623	ٛ	064D
ؤ	0624	ٜ	064E

Character	Code	Character	Code
إ	0625	ٗ	064F
ئ	0626	ـ	0650
ا	0627	ّ	0651
ب	0628	ٔ	0652
ة	0629	#	0653
ت	062A	#	0654
ث	062B	#	0657
ج	062C	ٰ	0670
ح	062D	ٹ	0679
خ	062E	ٺ	067A
د	062F	ٻ	067B
ذ	0630	ټ	067C
ر	0631	ٽ	067D
ز	0632	پ	067E
س	0633	ٿ	067F
ش	0634	ڀ	0680
ص	0635	ځ	0681
ض	0636	ڃ	0683
ط	0637	ڄ	0684
ظ	0638	څ	0685

Character	Code	Character	Code
ع	0639	چ	0686
غ	063A	ڇ	0687
ـ	0640	ڈ	0688
ف	0641	ډ	0689
ق	0642	ڊ	068A
ك	0643	ڌ	068C
ل	0644	ڍ	068D
م	0645	ڏ	068F
ن	0646	ڑ	0691
ه	0647	ړ	0693
ڕ	0695		

Uzbek character set

For Uzbek custom vocabularies, you can use the following characters in the `Phrase` field:

- a - z
- - (hyphen)
- . (period)

You can also use the following Unicode characters in the `Phrase` field:

Character	Code	Character	Code
т	0442	я	044F
б	0431	с	0441

Character	Code	Character	Code
о	043E	ҳ	04B3
п	043F	д	0434
ш	0448	р	0440
и	0438	ў	045E
ч	0447	г	0433
н	043D	ё	0451
қ	049B	й	0439
е	0435	в	0432
ю	044E	э	044D
з	0437	л	043B
х	0445	ф	0444
ц	0446	к	043A
м	043C	у	0443
ь	044C	ж	0436
ъ	044A	ғ	0493
а	0430		

Vietnamese character set

Amazon Transcribe represents the six tones in Vietnamese using numbers. The following table shows how tone marks are mapped for the word 'ma'.

Tone name	Tone mark	Tone number
ngang	ma	ma1
sắc	má	ma2
huyền	mà	ma3
hỏi	mả	ma4
ngã	mã	ma5
nặng	mạ	ma6

For Vietnamese custom vocabularies, you can use the following characters in the `Phrase` field:

- a - z
- A - Z
- ' (apostrophe)
- - (hyphen)
- . (period)
- & (ampersand)
- ; (semicolon)
- _ (low line)

You can also use the following Unicode characters in the `Phrase` field:

Character	Code	Character	Code
à	00E0	À	00C0
á	00E1	Á	00C1
â	00E2	Â	00C2
ã	00E3	Ã	00C3

Character	Code	Character	Code
è	00E8	È	00C8
é	00E9	É	00C9
ê	00EA	Ê	00CA
ì	00EC	Ì	00CC
í	00ED	Í	00CD
ò	00F2	Ò	00D2
ó	00F3	Ó	00D3
ô	00F4	Ô	00D4
õ	00F5	Õ	00D5
ù	00F9	Ù	00D9
ú	00FA	Ú	00DA
ý	00FD	Ý	00DD
ă	0103	Ă	0102
đ	0111	Đ	0110
ĩ	0129	Ĩ	0128
ũ	0169	Ũ	0168
ơ	01A1	Ơ	01A0
ư	01B0	Ư	01AF
ạ	1EA1	Ạ	1EA0
ả	1EA3	Ả	1EA2

Character	Code	Character	Code
ấ	1EA5	Ấ	1EA4
ầ	1EA7	Ầ	1EA6
ẩ	1EA9	Ẩ	1EA8
ẫ	1EAB	Ẫ	1EAA
ậ	1EAD	Ậ	1EAC
ắ	1EAF	Ắ	1EAE
ằ	1EB1	Ằ	1EB0
ẳ	1EB3	Ẳ	1EB2
ẵ	1EB5	Ẵ	1EB4
ặ	1EB7	Ặ	1EB6
ẹ	1EB9	Ẹ	1EB8
ẻ	1EBB	Ẻ	1EBA
ẽ	1EBD	Ẽ	1EBC
ế	1EBF	Ế	1EBE
ề	1EC1	Ề	1EC0
ể	1EC3	Ể	1EC2
ễ	1EC5	Ễ	1EC4
ệ	1EC7	Ệ	1EC6
ỉ	1EC9	Ỉ	1EC8
ị	1ECB	Ị	1ECA

Character	Code	Character	Code
ọ	1ECD	Ọ	1ECC
ỏ	1ECF	Ỏ	1ECE
ố	1ED1	Ố	1ED0
ồ	1ED3	Ồ	1ED2
ổ	1ED5	Ổ	1ED4
ỗ	1ED7	Ỗ	1ED6
ộ	1ED9	Ộ	1ED8
ớ	1EDB	Ớ	1EDA
ờ	1EDD	Ờ	1EDC
ở	1EDF	Ở	1EDE
ỡ	1EE1	Ỡ	1EE0
ợ	1EE3	Ợ	1EE2
ụ	1EE5	Ụ	1EE4
ủ	1EE7	Ủ	1EE6
ứ	1EE9	Ứ	1EE8
ừ	1EEB	Ừ	1EEA
ử	1EED	Ử	1EEC
ữ	1EEF	Ữ	1EEE
ự	1EF1	Ự	1EF0
ỳ	1EF3	Ỳ	1EF2

Character	Code	Character	Code
ỵ	1EF5	Ỵ	1EF4
ỷ	1EF7	Ỷ	1EF6
ỹ	1EF9	Ỹ	1EF8

Welsh character set

For Welsh custom vocabularies, you can use the following characters in the `Phrase` field:

- a - z
- - (hyphen)
- . (period)

You can also use the following Unicode characters in the `Phrase` field:

Character	Code	Character	Code
à	00E0	ò	00F2
á	00E1	ó	00F3
â	00E2	ô	00F4
ä	00E4	ö	00F6
è	00E8	ù	00F9
é	00E9	ú	00FA
ê	00EA	û	00FB
ë	00EB	ü	00FC
ì	00EC	ý	00FD
í	00ED	ÿ	00FF

Character	Code	Character	Code
î	00EE	ŵ	0175
ï	00EF	ŷ	0177
ỳ	1EF3		

Wolof character set

For Wolof custom vocabularies, you can use the following characters in the `Phrase` field:

- a - z
- - (hyphen)
- . (period)

You can also use the following Unicode characters in the `Phrase` field:

Character	Code	Character	Code
à	00E0	ê	00EA
ã	00E3	ë	00EB
ç	00E7	ñ	00F1
è	00E8	ó	00F3
é	00E9	ô	00F4
ŋ	014B		

Zulu character set

For Zulu custom vocabularies, you can use the following characters in the `Phrase` field:

- a - z

- - (hyphen)
- . (period)

You can also use the following Unicode characters in the `Phrase` field:

Character	Code	Character	Code
s	0073	d	0064
t	0074	a	0061
a	0061	r	0072
n	006E	d	0064

How Amazon Transcribe works

Amazon Transcribe uses machine learning models to convert speech to text.

In addition to the transcribed text, transcripts contains data about the transcribed content, including confidence scores and timestamps for each word or punctuation mark. To see an output example, refer to the Data input and output section. For a complete list of features that you can apply to your transcription, refer to the feature summary.

Transcription methods can be separated into two main categories:

- **Batch transcriptions**: Transcribe media files that have been uploaded into an Amazon S3 bucket. You can use the AWS CLI, AWS Management Console, and various AWS SDKs for batch transcriptions.
- **Streaming transcriptions**: Transcribe media streams in real time. You can use the AWS Management Console, HTTP/2, WebSockets, and various AWS SDKs for streaming transcriptions.

Note that feature and language support differs for batch and streaming transcriptions. For more information, refer to ??? and Supported languages.

Topics

- Data input and output
- Transcribing numbers and punctuation

API operations to get you started

Batch: `StartTranscriptionJob`
Streaming: `StartStreamTranscription`, StartStreamTranscriptionWebSocket

Data input and output

Amazon Transcribe takes audio data, as a media file in an Amazon S3 bucket or a media stream, and converts it to text data.

If you're transcribing media files stored in an Amazon S3 bucket, you're performing **batch transcriptions**. If you're transcribing media streams, you're performing **streaming transcriptions**. These two processes have different rules and requirements.

With batch transcriptions, you can use Job queueing if you don't need to process all of your transcription jobs concurrently. This allows Amazon Transcribe to keep track of your transcription jobs and process them when slots are available.

> **Note**
>
> Amazon Transcribe may temporarily store your content to continuously improve the quality of its analysis models. See the Amazon Transcribe FAQ to learn more. To request the deletion of content that may have been stored by Amazon Transcribe, open a case with Support.

Topics

- Media formats
- Audio channels
- Sample rates
- Output

Media formats

Supported media types differ between batch transcriptions and streaming transcriptions, though lossless formats are recommended for both. See the following table for details:

	Batch	Streaming
Supported formats	• AMR • FLAC • M4A • MP3 • MP4 • Ogg	• FLAC • Ogg Opus • PCM encoding

	Batch	Streaming
	• WebM • WAV	
Recommended formats	• FLAC • WAV with PCM 16-bit encoding	• FLAC • PCM signed 16-bit little-endian audio (note that this does **not** include WAV)

For best results, use a lossless format, such as FLAC or WAV with PCM 16-bit encoding.

> **Note**
>
> Streaming transcriptions are not supported with all languages. Refer to the 'Data input' column in the supported languages table for details.

Audio channels

Amazon Transcribe supports single-channel and dual-channel media. Media with more than two channels is not currently supported.

If your audio contains multiple speakers on one channel and you want to partition and label each speaker in your transcription output, you can use Speaker partitioning (diarization).

If your audio contains speech on two separate channels, you can use Channel identification to transcribe each channel separately within your transcript.

Both of these options produce one transcript file.

> **Note**
>
> If you don't enable Speaker partitioning or Channel identification, your transcript text is provided as one continuous section.

Sample rates

With batch transcription jobs, you can choose to provide a sample rate, though this parameter is optional. If you include it in your request, make sure the value you provide matches the actual sample rate in your audio. If you provide a sample rate that doesn't match your audio, your job may fail.

With streaming transcriptions, you must include a sample rate in your request. As with batch transcription jobs, make sure the value you provide matches the actual sample rate in your audio.

Sample rates for low fidelity audio, such as telephone recordings, typically use 8,000 Hz. For high fidelity audio, Amazon Transcribe supports values between 16,000 Hz and 48,000 Hz.

Output

Transcription output is in JSON format. The first part of your transcript contains the transcript itself in paragraph form, followed by additional data for every word and punctuation mark. The data provided depends on the features you include in your request. At a minimum, your transcript contains the start time, end time, and confidence score for every word. The following section shows example output from a basic transcription request that didn't include any additional options or features.

All **batch transcripts** are stored in Amazon S3 buckets. You can choose to save your transcript in your own Amazon S3 bucket, or have Amazon Transcribe use a secure default bucket. To learn more about creating and using Amazon S3 buckets, see Working with buckets.

If you want your transcript stored in an Amazon S3 bucket that you own, specify the bucket's URI in your transcription request. Make sure you give Amazon Transcribe write permissions for this bucket before starting your batch transcription job. If you specify your own bucket, your transcript remains in that bucket until you remove it.

If you don't specify an Amazon S3 bucket, Amazon Transcribe uses a secure service-managed bucket and provides you with a temporary URI you can use to download your transcript. Note that temporary URIs are valid for 15 minutes. If you get an `AccessDenied` error when using the provided URI, make a `GetTranscriptionJob` request to get a new temporary URI for your transcript.

If you opt for a default bucket, your transcript is deleted when your job expires (90 days). If you want to keep your transcript past this expiration date, you must download it.

Streaming transcripts are returned via the same method you're using for your stream.

> **Tip**
>
> If you want to convert your JSON output into a turn-by-turn transcript in Word format, see this GitHub example (for Python3). This script works with post-call analytics transcripts and standard batch transcripts with diarization enabled.

Example output

Transcripts provide a complete transcription in paragraph form, followed by a word-for-word breakdown, which provides data for every word and punctuation mark. This includes start time, end time, a confidence score, and a type (`pronunciation` or `punctuation`).

The following example is from a simple batch transcription job that didn't include any additional features. With each additional feature that you apply to your transcription request, you get additional data in your transcript output file.

Basic batch transcripts contain two main sections:

1. `transcripts`: Contains the entire transcript in one text block.
2. `items`: Contains information on each word and punctuation mark from the `transcripts` section.
3. `audio_segments`: An audio segment is a specific portion of an audio recording that contains uninterrupted spoken language, with minimal pauses or breaks. This segment captures a natural flow of speech and is captured in `audio_segments` with a start time and end time. The `items` element within an audio segment is a sequence of identifiers that correspond to each item within the segment.

Each additional feature you include in your transcription request produces additional information in your transcript.


```
{
    "jobName": "my-first-transcription-job",
    "accountId": "111122223333",
    "results": {
        "transcripts": [
            {
```

```
                "transcript": "Welcome to Amazon Transcribe."
            }
        ],
        "items": [
            {
                "id": 0,
                "start_time": "0.64",
                "end_time": "1.09",
                "alternatives": [
                    {
                        "confidence": "1.0",
                        "content": "Welcome"
                    }
                ],
                "type": "pronunciation"
            },
            {
                "id": 1,
                "start_time": "1.09",
                "end_time": "1.21",
                "alternatives": [
                    {
                        "confidence": "1.0",
                        "content": "to"
                    }
                ],
                "type": "pronunciation"
            },
            {
                "id": 2,
                "start_time": "1.21",
                "end_time": "1.74",
                "alternatives": [
                    {
                        "confidence": "1.0",
                        "content": "Amazon"
                    }
                ],
                "type": "pronunciation"
            },
            {
                "id": 3,
                "start_time": "1.74",
                "end_time": "2.56",
```

```
                "alternatives": [
                    {
                        "confidence": "1.0",
                        "content": "Transcribe"
                    }
                ],
                "type": "pronunciation"
            },
            {
                "id": 4,
                "alternatives": [
                    {
                        "confidence": "0.0",
                        "content": "."
                    }
                ],
                "type": "punctuation"
            }
        ],
        "audio_segments": [
            {
                "id": 0,
                "transcript": "Welcome to Amazon Transcribe.",
                "start_time": "0.64",
                "end_time": "2.56",
                "items": [
                    0,
                    1,
                    2,
                    3,
                    4
                ]
            }
        ]
    },
    "status": "COMPLETED"
}
```


Transcribing numbers and punctuation

Amazon Transcribe automatically adds punctuation to all supported languages, and capitalizes words appropriately for languages that use case distinction in their writing systems.

For most languages, numbers are transcribed into their word forms. However, for languages with support for transcribing numbers, Amazon Transcribe treats numbers differently depending on the context in which they're used.

For example, if a speaker says "*Meet me at eight-thirty AM on June first at one-hundred Main Street with three-dollars-and-fifty-cents and one-point-five chocolate bars*," this is transcribed as:

- Languages with number transcription support: Meet me at 8:30 a.m. on June 1st at 100 Main Street with $3.50 and 1.5 chocolate bars
- All other languages: Meet me at eight thirty a m on June first at one hundred Main Street with three dollars and fifty cents and one point five chocolate bars

To view the languages with support for transcribing numbers, refer to Supported languages and language-specific features.

Getting started with Amazon Transcribe

Before you can create transcriptions, you have a few prerequisites:

- Sign up for an AWS account
- Install the AWS CLI and SDKs (if you're using the AWS Management Console for your transcriptions, you can skip this step)
- Configure IAM credentials
- Set up an Amazon S3 bucket
- Create an IAM policy

Once you complete these prerequisites, you're ready to transcribe. Select your preferred transcription method from the following list to get started.

- AWS CLI
- AWS Management Console
- AWS SDK
- HTTP
- WebSockets

> **Tip**
>
> If you're new to Amazon Transcribe or would like to explore our features, we recommend using the AWS Management Console. This is also the easiest option if you'd like to start a stream using your computer microphone.

Because streaming using HTTP/2 and WebSockets is more complicated than the other transcription methods, we recommend reviewing the Setting up a streaming transcription section before getting started with these methods. **Note that we strongly recommend using an SDK for streaming transcriptions.**

Sign up for an AWS account

To get started with AWS, you need an AWS account. There are two ways to sign up for AWS. If you want complete control over account configuration and where your content is stored, or if you have regulated workloads such as HIPAA or FedRAMP, use Sign up for AWS (advanced). If you are a new user and want to use a login you already own to sign in to AWS and create an AWS environment with preconfigured defaults in select AWS Regions, use Sign up for AWS.

Installing the AWS CLI and SDKs

To use the Amazon Transcribe API, you must first install the AWS CLI. The current AWS CLI is version 2. You can find installation instructions for Linux, Mac, Windows, and Docker in the *AWS Command Line Interface User Guide*.

Once you have the AWS CLI installed, you must configure it for your security credentials and AWS Region.

If you want to use Amazon Transcribe with an SDK, select your preferred language for installation instructions:

- .NET
- C++
- Go
- Java V2
- JavaScript
- PHP V3
- AWS SDK for Python (Boto3) (batch transcriptions)
- Python (streaming transcriptions)
- Ruby V3
- Rust (batch transcriptions)
- Rust (streaming transcriptions)

Creating an Amazon S3 bucket

Amazon S3 is a secure object storage service. Amazon S3 stores your files (called *objects*) in containers (called *buckets*).

To run a batch transcription, you must first upload your media files into an Amazon S3 bucket. If you don't specify an Amazon S3 bucket for your transcription output, Amazon Transcribe puts your transcript in a temporary AWS-managed Amazon S3 bucket. Transcription output in AWS-managed buckets is automatically deleted after 90 days.

Learn how to Create your first S3 bucket and Upload an object to your bucket.

Creating an IAM policy

To manage access in AWS, you must create policies and attach them to IAM identities (users, groups, or roles) or AWS resources. A policy defines the permissions of the entity it is attached to. For example, a role can only access a media file located in your Amazon S3 bucket if you've attached a policy to that role which grants it access. If you want to further restrict that role, you can instead limit its access to a specific file within an Amazon S3 bucket.

To learn more about using AWS policies see:

- Policies and permissions in IAM
- Creating IAM policies
- How Amazon Transcribe works with IAM

For example policies you can use with Amazon Transcribe, see Amazon Transcribe identity-based policy examples. If you want to generate custom policies, consider using the AWS Policy Generator.

You can add a policy using the AWS Management Console, AWS CLI, or AWS SDK. For instructions, see Adding and removing IAM identity permissions.

Policies have the format:

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Sid": "my-policy-name",
            "Effect": "Allow",
            "Action": [
```

```
                "service:action"
            ],
            "Resource": [
                "amazon-resource-name"
            ]
        }
    ]
}
```

Amazon Resource Names (ARNs) uniquely identify all AWS resources, such as an Amazon S3 bucket. You can use ARNs in your policy to grant permissions for specific actions to use specific resources. For example, if you want to grant read access to an Amazon S3 bucket and its sub-folders, you can add the following code to your trust policy's `Statement` section:

```
{
        "Effect": "Allow",
        "Action": [
            "s3:GetObject",
            "s3:ListBucket"
        ],
        "Resource": [
            "arn:aws:s3:::amzn-s3-demo-bucket",
            "arn:aws:s3:::amzn-s3-demo-bucket/*"
        ]
}
```

Here's an example policy that grants Amazon Transcribe read (`GetObject`, `ListBucket`) and write (`PutObject`) permissions to an Amazon S3 bucket, `amzn-s3-demo-bucket`, and its sub-folders:

JSON

```
{
  "Version":"2012-10-17",
  "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "s3:GetObject",
                "s3:ListBucket"
            ],
```

```
            "Resource": [
                "arn:aws-eusc:s3:::amzn-s3-demo-bucket",
                "arn:aws-eusc:s3:::amzn-s3-demo-bucket/*"
            ]
        },
        {
             "Effect": "Allow",
             "Action": [
                 "s3:PutObject"
             ],
             "Resource": [
                 "arn:aws-eusc:s3:::amzn-s3-demo-bucket",
                 "arn:aws-eusc:s3:::amzn-s3-demo-bucket/*"
             ]
        }
    ]
}
```

Transcribing with the AWS Management Console

You can use the AWS console for batch and streaming transcriptions. If you're transcribing a media file located in an Amazon S3 bucket, you're performing a batch transcription. If you're transcribing a real-time stream of audio data, you're performing a streaming transcription.

Before starting a batch transcription, you must first upload your media file to an Amazon S3 bucket. For streaming transcriptions using the AWS Management Console, you must use your computer microphone.

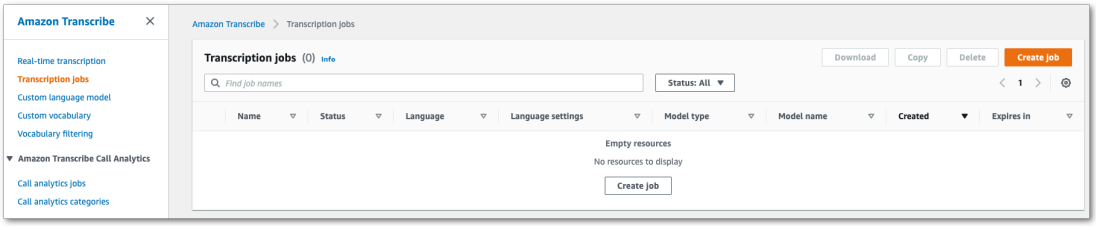
To view supported media formats and other media requirements and constraints, see Data input and output.

Expand the following sections for short walkthroughs of each transcription method.

Batch transcriptions

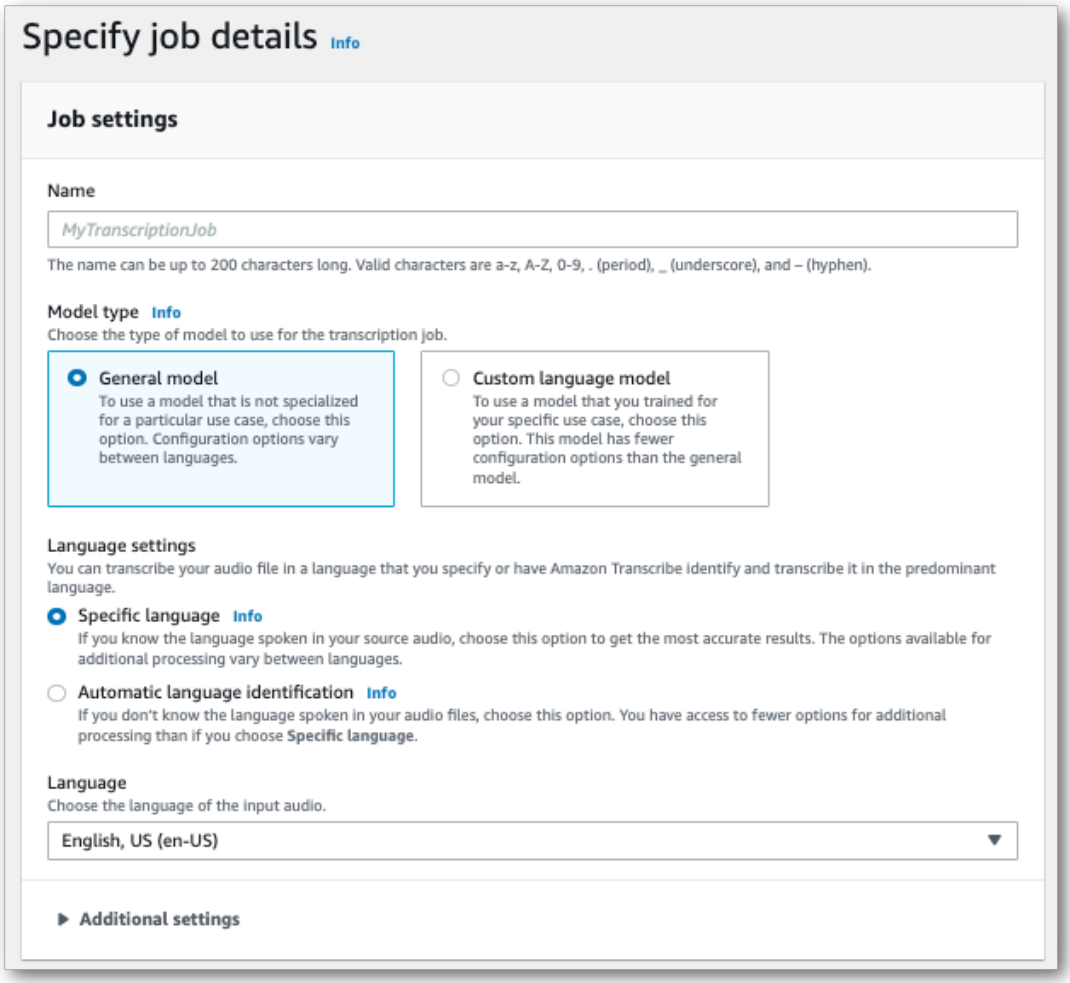
First make sure that you've uploaded the media file you want to transcribe into an Amazon S3 bucket. If you're unsure how to do this, refer to the *Amazon S3 User Guide*: Upload an object to your bucket.

1. From the AWS Management Console, select **Transcription jobs** in the left navigation pane. This takes you to a list of your transcription jobs.



Select **Create job**.

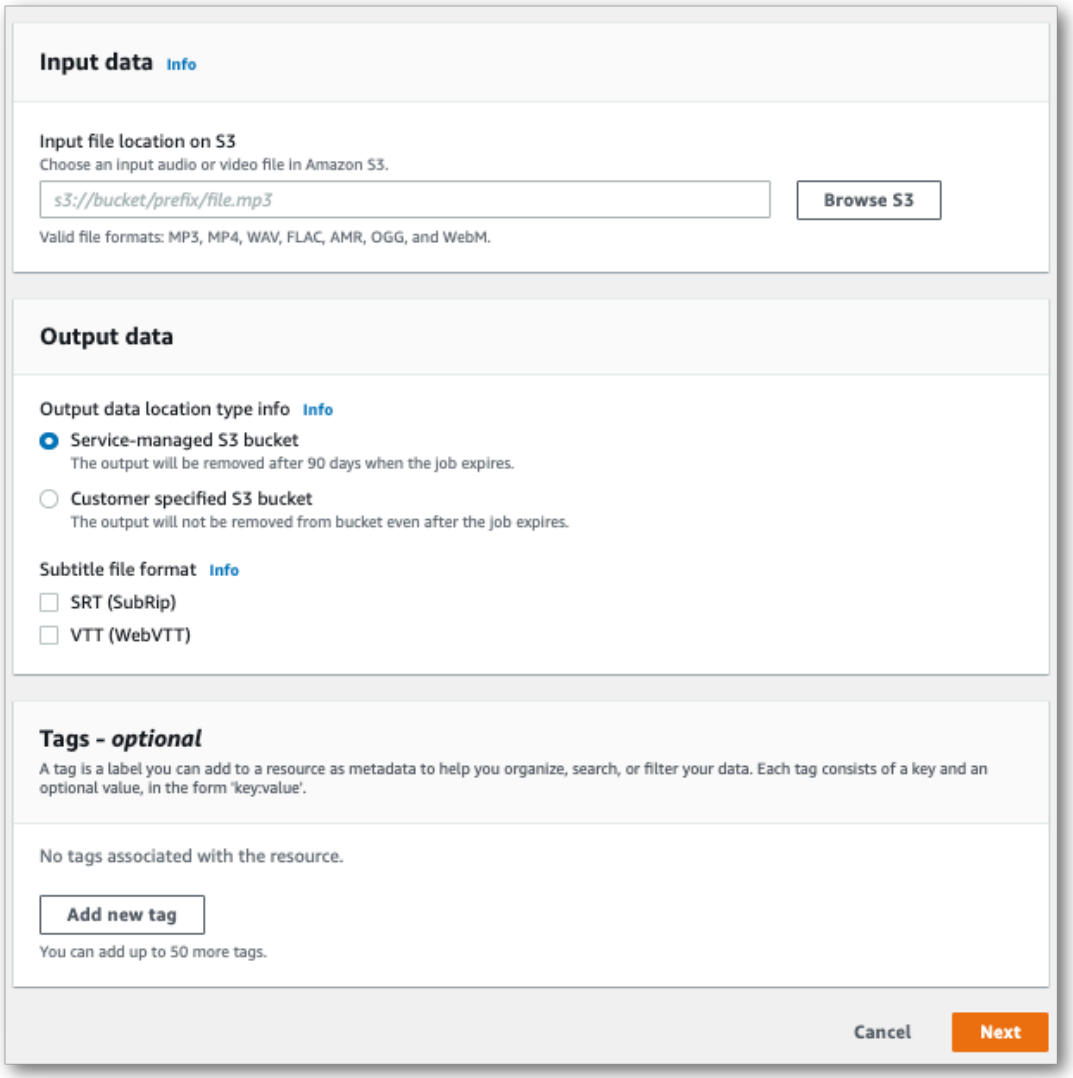
2. Complete the fields on the **Specify job details** page.



The input location *must* be an object within an Amazon S3 bucket. For output location, you can choose a secure Amazon S3 service-managed bucket or you can specify your own Amazon S3 bucket.

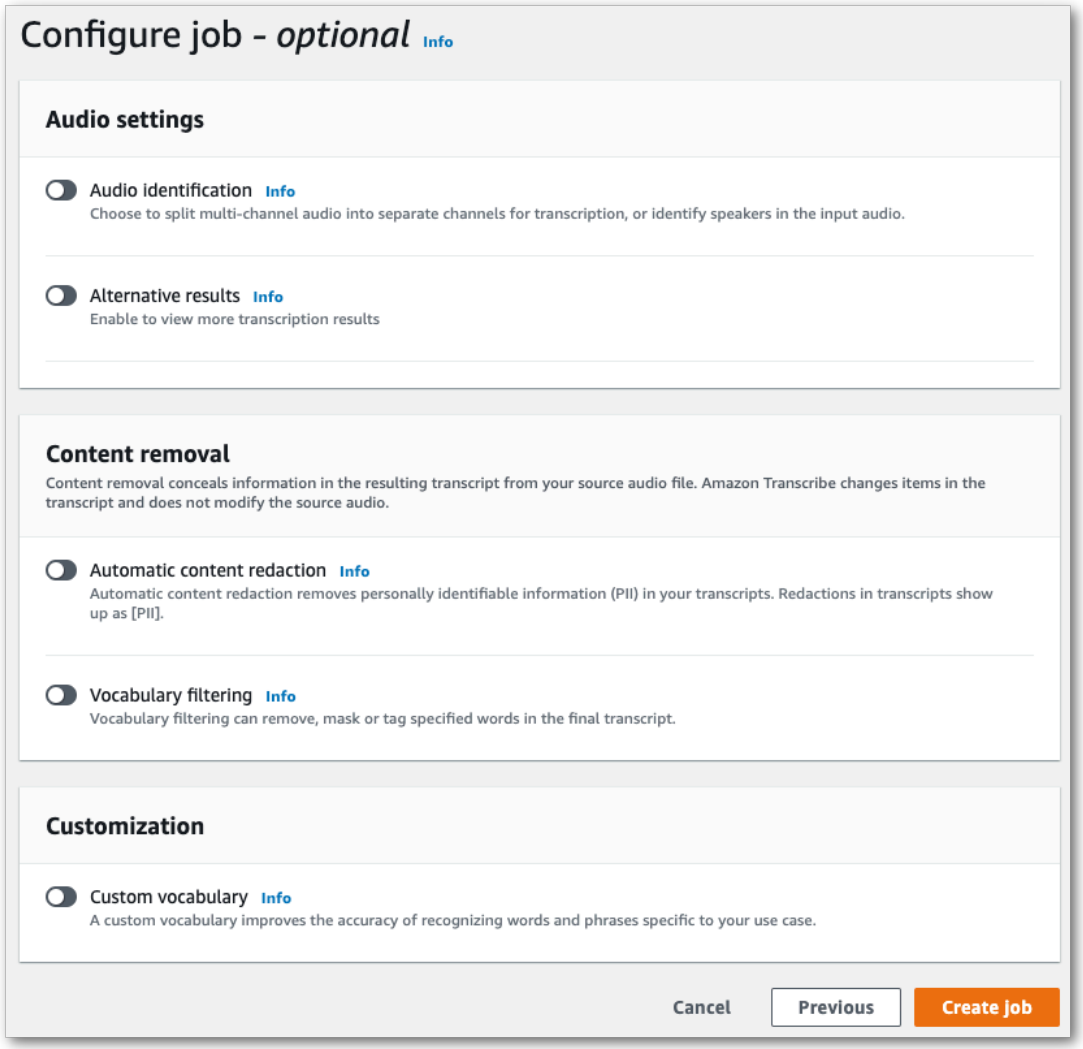
If you choose a service-managed bucket, you can view a transcript preview in the AWS Management Console and you can download your transcript from the job details page (see below).

If you choose your own Amazon S3 bucket, you cannot see a preview in the AWS Management Console and must go to the Amazon S3 bucket to download your transcript.



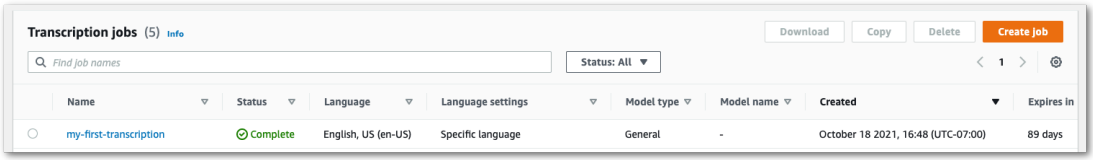
Select **Next**.

3. Select any desired options on the **Configure job** page. If you want to use Custom vocabularies or Custom language models with your transcription, you must create these before starting your transcription job.



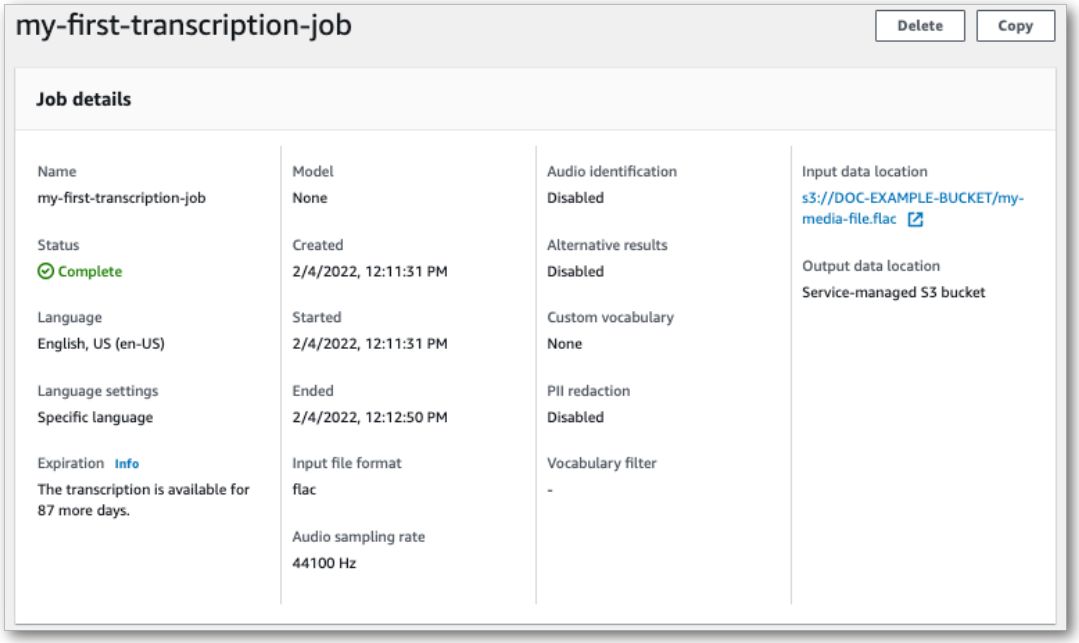
Select **Create job**.

4. You're now on the **Transcription jobs** page. Here you can see the status of the transcription job. Once complete, select your transcription.



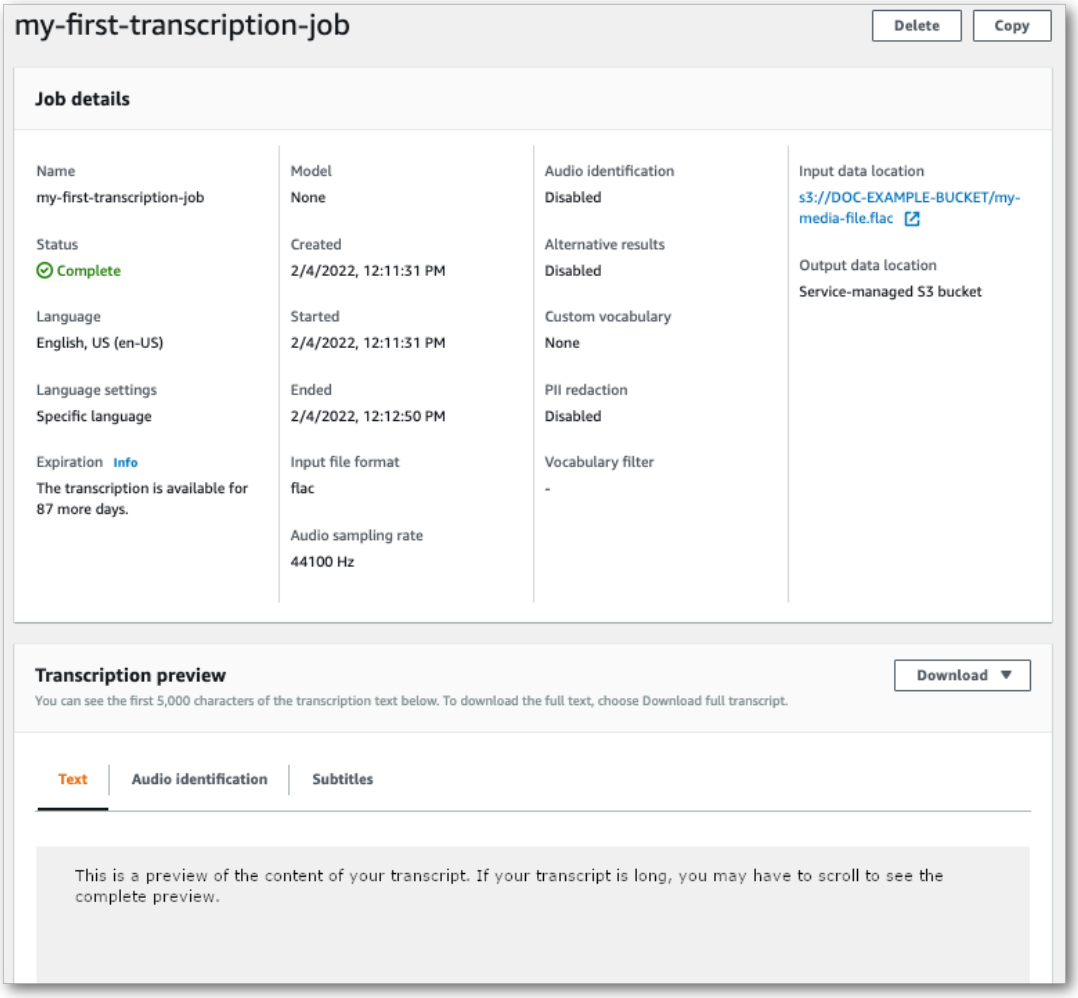
5. You're now viewing the **Job details** page for your transcription. Here you can view all of the options you specified when setting up your transcription job.

   To view your transcript, select the linked filepath in the right column under **Output data location**. This takes you to the Amazon S3 output folder you specified. Select your output file, which now has a .json extension.



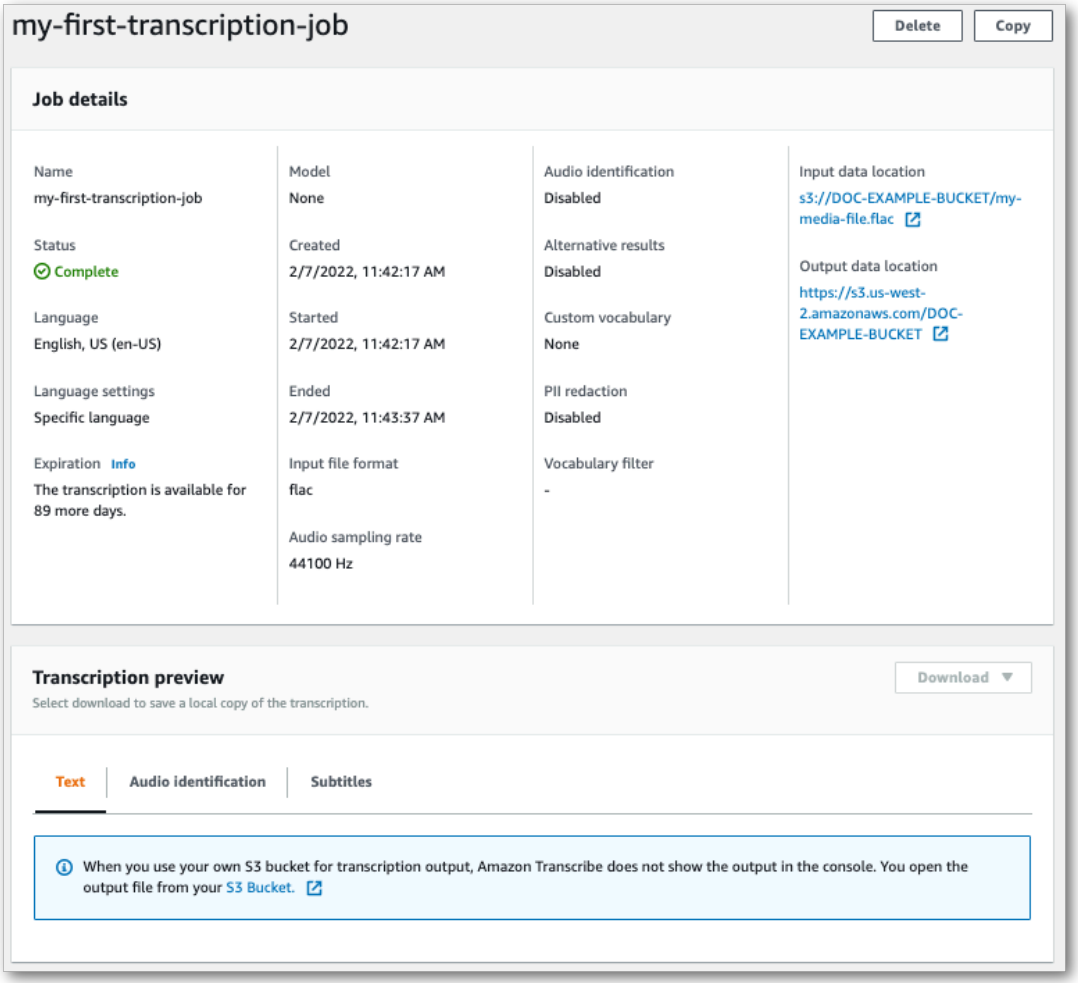
6. How you download your transcript depends on whether you chose a service-managed Amazon S3 bucket or your own Amazon S3 bucket.

   a. If you chose a service-managed bucket, you can see a **Transcription preview** pane on your transcription job's information page, along with a **Download** button.



Select **Download** and choose **Download transcript**.

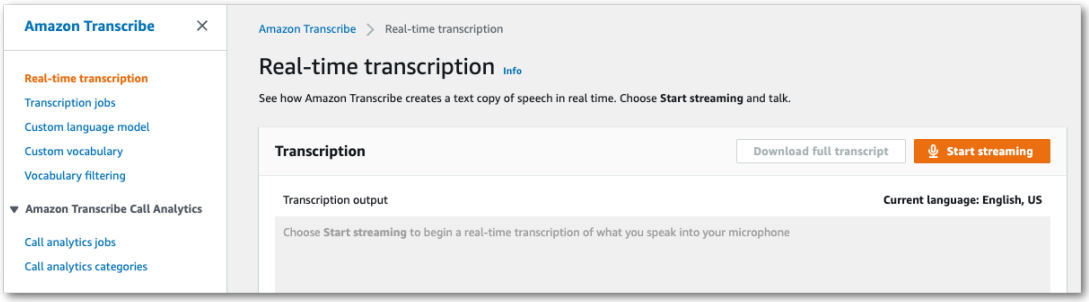
b. If you chose your own Amazon S3 bucket, you don't see any text in the **Transcription preview** pane on your transcription job's information page. Instead, you see a blue information box with a link to the Amazon S3 bucket you chose.



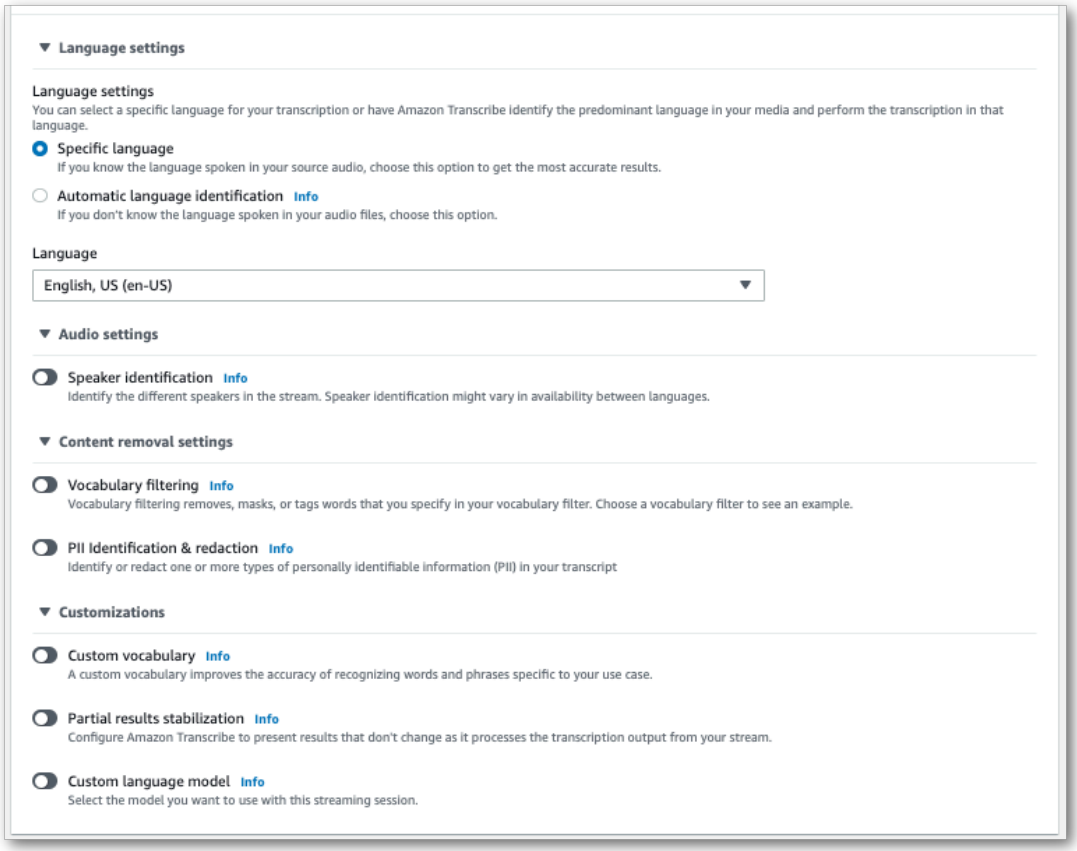
To access your transcript, go to the specified Amazon S3 bucket using the link under **Output data location** in the **Job details** pane or the **S3 Bucket** link within the blue information box in the **Transcription preview** pane.

Streaming transcriptions

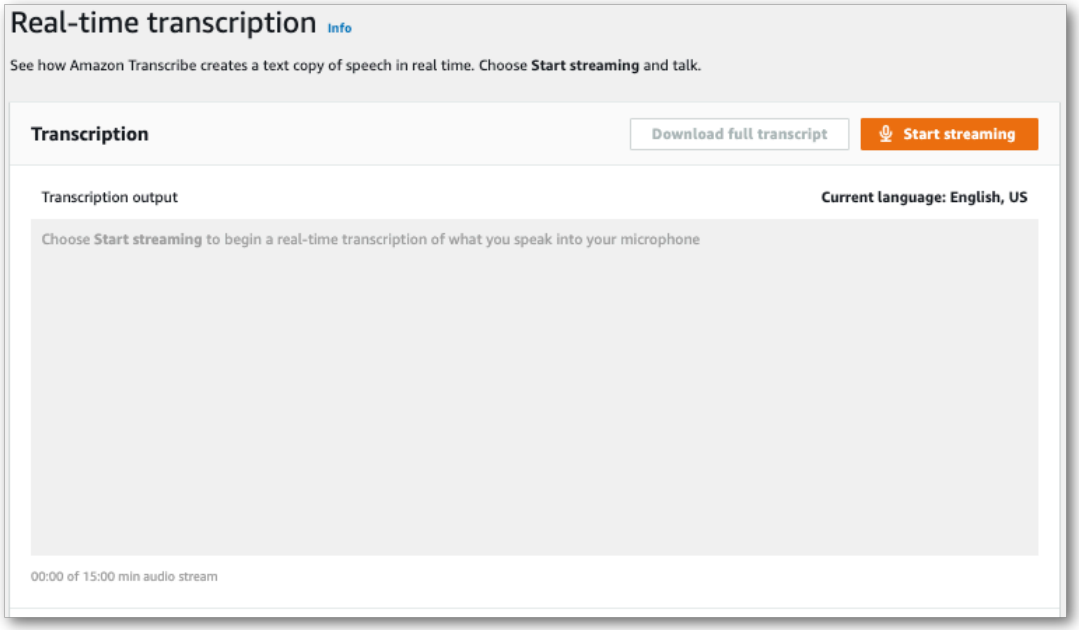
1. From the AWS Management Console, select **Real-time transcription** in the left navigation pane. This takes you to the main streaming page where you can select options before starting your stream.



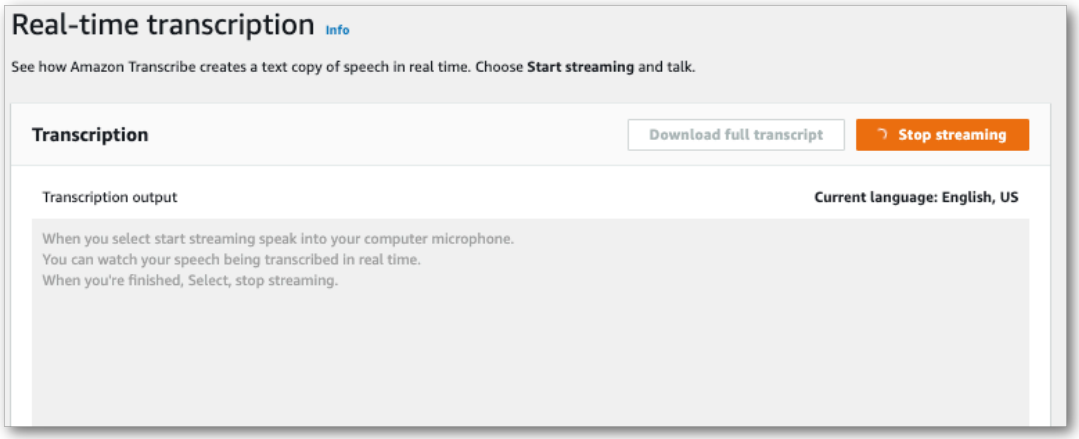
2. Below the **Transcription output** box, you have the option to select various language and audio settings.



3. After you've selected the appropriate settings, scroll to the top of the page and choose **Start streaming**, then begin speaking into your computer microphone. You can see your speech transcribed in real time.



4. When you're finished, select **Stop streaming**.



You can now download your transcript by selecting **Download full transcript**.

Transcribing with the AWS CLI

When using the AWS CLI to start a transcription, you can run all commands at the CLI level. Or you can run the command you want to use, followed by the AWS Region and the location of a JSON file that contains a request body. Examples throughout this guide show both methods; however, this section focuses on the former method.

The AWS CLI does not support streaming transcriptions.

Before you continue, make sure you've:

- Uploaded your media file into an Amazon S3 bucket. If you're unsure how to create an Amazon S3 bucket or upload your file, refer to Create your first Amazon S3 bucket and Upload an object to your bucket.
- Installed the AWS CLI.

You can find all AWS CLI commands for Amazon Transcribe in the AWS CLI Command Reference.

Starting a new transcription job

To start a new transcription, use the `start-transcription-job` command.

1. In a terminal window, type the following:

   ```
   aws transcribe start-transcription-job \
   ```

   A '>' appears on the next line, and you can now continue adding required parameters, as described in the next step.

   You can also omit the '\' and append all parameters, separating each with a space.

2. With the `start-transcription-job` command, you must include `region`, `transcription-job-name`, `media`, and either `language-code` or `identify-language`.

   If you want to specify an output location, include `output-bucket-name` in your request; if you want to specify a sub-folder of the specified output bucket, also include `output-key`.

   ```
   aws transcribe start-transcription-job \
    --region us-west-2 \
    --transcription-job-name my-first-transcription-job \
    --media MediaFileUri=s3://amzn-s3-demo-bucket/my-input-files/my-media-file.flac \
    --language-code en-US
   ```

   If appending all parameters, this request looks like:

   ```
   aws transcribe start-transcription-job --region us-west-2 --transcription-job-
   name my-first-transcription-job --media MediaFileUri=s3://amzn-s3-demo-bucket/my-
   input-files/my-media-file.flac --language-code en-US
   ```

If you choose not to specify an output bucket using `output-bucket-name`, Amazon Transcribe places your transcription output in a service-managed bucket. Transcripts stored in a service-managed bucket expire after 90 days.

Amazon Transcribe responds with:

```
{
    "TranscriptionJob": {
        "TranscriptionJobName": "my-first-transcription-job",
        "TranscriptionJobStatus": "IN_PROGRESS",
        "LanguageCode": "en-US",
        "Media": {
            "MediaFileUri": "s3://amzn-s3-demo-bucket/my-input-files/my-media-
file.flac"
        },
        "StartTime": "2022-03-07T15:03:44.246000-08:00",
        "CreationTime": "2022-03-07T15:03:44.229000-08:00"
    }
}
```

Your transcription job is successful if `TranscriptionJobStatus` changes from `IN_PROGRESS` to `COMPLETED`. To see the updated `TranscriptionJobStatus`, use the `get-transcription-job` or `list-transcription-job` command, as shown in the following section.

Getting the status of a transcription job

To get information about your transcription job, use the `get-transcription-job` command.

The only required parameters for this command are the AWS Region where the job is located and the name of the job.

```
aws transcribe get-transcription-job \
 --region us-west-2 \
 --transcription-job-name my-first-transcription-job
```

Amazon Transcribe responds with:

```
{
    "TranscriptionJob": {
        "TranscriptionJobName": "my-first-transcription-job",
```

```
        "TranscriptionJobStatus": "COMPLETED",
        "LanguageCode": "en-US",
        "MediaSampleRateHertz": 48000,
        "MediaFormat": "flac",
        "Media": {
            "MediaFileUri": "s3://amzn-s3-demo-bucket/my-input-files/my-media-
file.flac"
        },
        "Transcript": {
            "TranscriptFileUri": "https://s3.the-URI-where-your-job-is-located.json"
        },
        "StartTime": "2022-03-07T15:03:44.246000-08:00",
        "CreationTime": "2022-03-07T15:03:44.229000-08:00",
        "CompletionTime": "2022-03-07T15:04:01.158000-08:00",
        "Settings": {
            "ChannelIdentification": false,
            "ShowAlternatives": false
        }
    }
}
```

If you've selected your own Amazon S3 bucket for your transcription output, this bucket is listed with `TranscriptFileUri`. If you've selected a service-managed bucket, a temporary URI is provided; use this URI to download your transcript.

> **Note**
>
> Temporary URIs for service-managed Amazon S3 buckets are only valid for 15 minutes. If you get an `AccesDenied` error when using the URI, run the `get-transcription-job` request again to get a new temporary URI.

Listing your transcription jobs

To list all your transcription jobs in a given AWS Region, use the `list-transcription-jobs` command.

The only required parameter for this command is the AWS Region in which your transcription jobs are located.

```
aws transcribe list-transcription-jobs \
```

```
 --region us-west-2
```

Amazon Transcribe responds with:

```
{
    "NextToken": "A-very-long-string",
    "TranscriptionJobSummaries": [
        {
            "TranscriptionJobName": "my-first-transcription-job",
            "CreationTime": "2022-03-07T15:03:44.229000-08:00",
            "StartTime": "2022-03-07T15:03:44.246000-08:00",
            "CompletionTime": "2022-03-07T15:04:01.158000-08:00",
            "LanguageCode": "en-US",
            "TranscriptionJobStatus": "COMPLETED",
            "OutputLocationType": "SERVICE_BUCKET"
        }
    ]
}
```

Deleting your transcription job

To delete your transcription job, use the `delete-transcription-job` command.

The only required parameters for this command are the AWS Region where the job is located and the name of the job.

```
aws transcribe delete-transcription-job \
 --region us-west-2 \
 --transcription-job-name my-first-transcription-job
```

To confirm your delete request is successful, you can run the `list-transcription-jobs` command. Your job should no longer appear in the list.

Transcribing with the AWS SDKs

You can use SDKs for both batch and streaming transcriptions. If you're transcribing a media file located in an Amazon S3 bucket, you're performing a batch transcription. If you're transcribing a real-time stream of audio data, you're performing a streaming transcription.

For a list of the programming languages you can use with Amazon Transcribe, see Supported programming languages. Note that streaming transcriptions are not supported with all AWS SDKs.

To view supported media formats and other media requirements and constraints, see Data input and output.

For more information on all available AWS SDKs and builder tools, refer to Tools to Build on AWS.

> **Tip**
>
> For additional examples using the AWS SDKs, including feature-specific, scenario, and cross-service examples, refer to the Code examples for Amazon Transcribe using AWS SDKs chapter.
> You can also find SDK code samples in these GitHub repositories:
>
> - AWS Code Examples
> - Amazon Transcribe Examples

Batch transcriptions

You can create batch transcriptions using the URI of a media file located in an Amazon S3 bucket. If you're unsure how to create an Amazon S3 bucket or upload your file, refer to Create your first S3 bucket and Upload an object to your bucket.

Java

```
import software.amazon.awssdk.auth.credentials.AwsCredentialsProvider;
import software.amazon.awssdk.auth.credentials.DefaultCredentialsProvider;
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.transcribe.TranscribeClient;
import software.amazon.awssdk.services.transcribe.model.*;
import software.amazon.awssdk.services.transcribestreaming.model.LanguageCode;

public class TranscribeDemoApp {
    private static final Region REGION = Region.US_WEST_2;
    private static TranscribeClient client;

    public static void main(String args[]) {

        client = TranscribeClient.builder()
                .credentialsProvider(getCredentials())
                .region(REGION)
                .build();
```

```
        String transcriptionJobName = "my-first-transcription-job";
        String mediaType = "flac"; // can be other types
        Media myMedia = Media.builder()
                .mediaFileUri("s3://amzn-s3-demo-bucket/my-input-files/my-media-
file.flac")
                .build();

        String outputS3BucketName = "s3://amzn-s3-demo-bucket";
        // Create the transcription job request
        StartTranscriptionJobRequest request =
 StartTranscriptionJobRequest.builder()
                .transcriptionJobName(transcriptionJobName)
                .languageCode(LanguageCode.EN_US.toString())
                .mediaSampleRateHertz(16000)
                .mediaFormat(mediaType)
                .media(myMedia)
                .outputBucketName(outputS3BucketName)
                .build();

        // send the request to start the transcription job
        StartTranscriptionJobResponse startJobResponse =
 client.startTranscriptionJob(request);

        System.out.println("Created the transcription job");
        System.out.println(startJobResponse.transcriptionJob());

        // Create the get job request
        GetTranscriptionJobRequest getJobRequest =
 GetTranscriptionJobRequest.builder()
                .transcriptionJobName(transcriptionJobName)
                .build();

        // send the request to get the transcription job including the job status
        GetTranscriptionJobResponse getJobResponse =
 client.getTranscriptionJob(getJobRequest);

        System.out.println("Get the transcription job request");
        System.out.println(getJobResponse.transcriptionJob());
    }

    private static AwsCredentialsProvider getCredentials() {
        return DefaultCredentialsProvider.create();
    }
```

```
}
```

JavaScript

```
const { TranscribeClient, StartTranscriptionJobCommand } = require("@aws-sdk/client-
transcribe"); // CommonJS import

const region = "us-west-2";
const credentials = {
  "accessKeyId": "",
  "secretAccessKey": "",
};

const input = {
  TranscriptionJobName: "my-first-transcription-job",
  LanguageCode: "en-US",
  Media: {
    MediaFileUri: "s3://amzn-s3-demo-bucket/my-input-files/my-media-file.flac"
  },
  OutputBucketName: "amzn-s3-demo-bucket",
};

async function startTranscriptionRequest() {
  const transcribeConfig = {
    region,
    credentials
  };
  const transcribeClient = new TranscribeClient(transcribeConfig);
  const transcribeCommand = new StartTranscriptionJobCommand(input);
  try {
    const transcribeResponse = await transcribeClient.send(transcribeCommand);
    console.log("Transcription job created, the details:");
    console.log(transcribeResponse.TranscriptionJob);
  } catch(err) {
    console.log(err);
  }
}

startTranscriptionRequest();
```

Python

```
import time
import boto3

def transcribe_file(job_name, file_uri, transcribe_client):
    transcribe_client.start_transcription_job(
        TranscriptionJobName = job_name,
        Media = {
            'MediaFileUri': file_uri
        },
        MediaFormat = 'flac',
        LanguageCode = 'en-US'
    )

    max_tries = 60
    while max_tries > 0:
        max_tries -= 1
        job = transcribe_client.get_transcription_job(TranscriptionJobName =
 job_name)
        job_status = job['TranscriptionJob']['TranscriptionJobStatus']
        if job_status in ['COMPLETED', 'FAILED']:
            print(f"Job {job_name} is {job_status}.")
            if job_status == 'COMPLETED':
                print(
                    f"Download the transcript from\n"
                    f"\t{job['TranscriptionJob']['Transcript']
['TranscriptFileUri']}.")
            break
        else:
            print(f"Waiting for {job_name}. Current status is {job_status}.")
        time.sleep(10)


def main():
    transcribe_client = boto3.client('transcribe', region_name = 'us-west-2')
    file_uri = 's3://amzn-s3-demo-bucket/my-input-files/my-media-file.flac'
    transcribe_file('Example-job', file_uri, transcribe_client)


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

Streaming transcriptions

You can create streaming transcriptions using a streamed media file or a live media stream.

Note that the standard AWS SDK for Python (Boto3) is not supported for Amazon Transcribe streaming. To start a streaming transcription using Python, use this async Python SDK for Amazon Transcribe.

Java

The following example is a Java program that transcribes streaming audio.

To run this example, note the following:

- You must use the AWS SDK for Java 2.x.
- Clients must use Java 1.8 to be compatible with the AWS SDK for Java 2.x.
- The sample rate you specify must match the actual sample rate of your audio stream.

See also: Retry client for Amazon Transcribe streaming (Java SDK). This code manages the connection to Amazon Transcribe and retries sending data when there are errors on the connection. For example, if there is a transient error on the network, this client resends the request that failed.

```
public class TranscribeStreamingDemoApp {
    private static final Region REGION = Region.US_WEST_2;
    private static TranscribeStreamingAsyncClient client;

    public static void main(String args[]) throws URISyntaxException,
 ExecutionException, InterruptedException, LineUnavailableException {

        client = TranscribeStreamingAsyncClient.builder()
                .credentialsProvider(getCredentials())
                .region(REGION)
                .build();

        CompletableFuture<Void> result =
 client.startStreamTranscription(getRequest(16_000),
                new AudioStreamPublisher(getStreamFromMic()),
                getResponseHandler());

        result.get();
```

```
        client.close();
    }

    private static InputStream getStreamFromMic() throws LineUnavailableException {

        // Signed PCM AudioFormat with 16,000 Hz, 16 bit sample size, mono
        int sampleRate = 16000;
        AudioFormat format = new AudioFormat(sampleRate, 16, 1, true, false);
        DataLine.Info info = new DataLine.Info(TargetDataLine.class, format);

        if (!AudioSystem.isLineSupported(info)) {
            System.out.println("Line not supported");
            System.exit(0);
        }

        TargetDataLine line = (TargetDataLine) AudioSystem.getLine(info);
        line.open(format);
        line.start();

        InputStream audioStream = new AudioInputStream(line);
        return audioStream;
    }

    private static AwsCredentialsProvider getCredentials() {
        return DefaultCredentialsProvider.create();
    }

    private static StartStreamTranscriptionRequest getRequest(Integer
mediaSampleRateHertz) {
        return StartStreamTranscriptionRequest.builder()
                .languageCode(LanguageCode.EN_US.toString())
                .mediaEncoding(MediaEncoding.PCM)
                .mediaSampleRateHertz(mediaSampleRateHertz)
                .build();
    }

    private static StartStreamTranscriptionResponseHandler getResponseHandler() {
        return StartStreamTranscriptionResponseHandler.builder()
                .onResponse(r -> {
                    System.out.println("Received Initial response");
                })
                .onError(e -> {
                    System.out.println(e.getMessage());
                    StringWriter sw = new StringWriter();
```

```
                    e.printStackTrace(new PrintWriter(sw));
                    System.out.println("Error Occurred: " + sw.toString());
                })
                .onComplete(() -> {
                    System.out.println("=== All records stream successfully ===");
                })
                .subscriber(event -> {
                    List<Result> results = ((TranscriptEvent)
 event).transcript().results();
                    if (results.size() > 0) {
                        if (!
results.get(0).alternatives().get(0).transcript().isEmpty()) {

 System.out.println(results.get(0).alternatives().get(0).transcript());
                        }
                    }
                })
                .build();
    }

    private InputStream getStreamFromFile(String myMediaFileName) {
        try {
            File inputFile = new
 File(getClass().getClassLoader().getResource(myMediaFileName).getFile());
            InputStream audioStream = new FileInputStream(inputFile);
            return audioStream;
        } catch (FileNotFoundException e) {
            throw new RuntimeException(e);
        }
    }

    private static class AudioStreamPublisher implements Publisher<AudioStream> {
        private final InputStream inputStream;
        private static Subscription currentSubscription;


        private AudioStreamPublisher(InputStream inputStream) {
            this.inputStream = inputStream;
        }

        @Override
        public void subscribe(Subscriber<? super AudioStream> s) {

            if (this.currentSubscription == null) {
```

```
                this.currentSubscription = new SubscriptionImpl(s, inputStream);
            } else {
                this.currentSubscription.cancel();
                this.currentSubscription = new SubscriptionImpl(s, inputStream);
            }
            s.onSubscribe(currentSubscription);
        }
    }

    public static class SubscriptionImpl implements Subscription {
        private static final int CHUNK_SIZE_IN_BYTES = 1024 * 1;
        private final Subscriber<? super AudioStream> subscriber;
        private final InputStream inputStream;
        private ExecutorService executor = Executors.newFixedThreadPool(1);
        private AtomicLong demand = new AtomicLong(0);

        SubscriptionImpl(Subscriber<? super AudioStream> s, InputStream inputStream)
{
            this.subscriber = s;
            this.inputStream = inputStream;
        }

        @Override
        public void request(long n) {
            if (n <= 0) {
                subscriber.onError(new IllegalArgumentException("Demand must be
positive"));
            }

            demand.getAndAdd(n);

            executor.submit(() -> {
                try {
                    do {
                        ByteBuffer audioBuffer = getNextEvent();
                        if (audioBuffer.remaining() > 0) {
                            AudioEvent audioEvent =
audioEventFromBuffer(audioBuffer);
                            subscriber.onNext(audioEvent);
                        } else {
                            subscriber.onComplete();
                            break;
                        }
                    } while (demand.decrementAndGet() > 0);
```

```
                } catch (Exception e) {
                    subscriber.onError(e);
                }
            });
        }

        @Override
        public void cancel() {
            executor.shutdown();
        }

        private ByteBuffer getNextEvent() {
            ByteBuffer audioBuffer = null;
            byte[] audioBytes = new byte[CHUNK_SIZE_IN_BYTES];

            int len = 0;
            try {
                len = inputStream.read(audioBytes);

                if (len <= 0) {
                    audioBuffer = ByteBuffer.allocate(0);
                } else {
                    audioBuffer = ByteBuffer.wrap(audioBytes, 0, len);
                }
            } catch (IOException e) {
                throw new UncheckedIOException(e);
            }

            return audioBuffer;
        }

        private AudioEvent audioEventFromBuffer(ByteBuffer bb) {
            return AudioEvent.builder()
                    .audioChunk(SdkBytes.fromByteBuffer(bb))
                    .build();
        }
    }
}
```

JavaScript

```
const {
  TranscribeStreamingClient,
```

```
  StartStreamTranscriptionCommand,
} = require("@aws-sdk/client-transcribe-streaming");
const { createReadStream } = require("fs");
const { join } = require("path");

const audio = createReadStream(join(__dirname, "my-media-file.flac"),
 { highWaterMark: 1024 * 16});

const LanguageCode = "en-US";
const MediaEncoding = "pcm";
const MediaSampleRateHertz = "16000";
const credentials = {
  "accessKeyId": "",
  "secretAccessKey": "",
};
async function startRequest() {
  const client = new TranscribeStreamingClient({
    region: "us-west-2",
    credentials
  });

  const params = {
    LanguageCode,
    MediaEncoding,
    MediaSampleRateHertz,
    AudioStream: (async function* () {
      for await (const chunk of audio) {
        yield {AudioEvent: {AudioChunk: chunk}};
      }
    })(),
  };
  const command = new StartStreamTranscriptionCommand(params);
  // Send transcription request
  const response = await client.send(command);
  // Start to print response
  try {
    for await (const event of response.TranscriptResultStream) {
      console.log(JSON.stringify(event));
    }
  } catch(err) {
    console.log("error")
    console.log(err)
  }
}
```

```
startRequest();
```

Python

The following example is a Python program that transcribes streaming audio.

To run this example, note the following:

- You must use this SDK for Python .
- The sample rate you specify must match the actual sample rate of your audio stream.

```
import asyncio
# This example uses aiofile for asynchronous file reads.
# It's not a dependency of the project but can be installed
# with `pip install aiofile`.
import aiofile

from amazon_transcribe.client import TranscribeStreamingClient
from amazon_transcribe.handlers import TranscriptResultStreamHandler
from amazon_transcribe.model import TranscriptEvent

"""
Here's an example of a custom event handler you can extend to
process the returned transcription results as needed. This
handler will simply print the text out to your interpreter.
"""
class MyEventHandler(TranscriptResultStreamHandler):
    async def handle_transcript_event(self, transcript_event: TranscriptEvent):
        # This handler can be implemented to handle transcriptions as needed.
        # Here's an example to get started.
        results = transcript_event.transcript.results
        for result in results:
            for alt in result.alternatives:
                print(alt.transcript)


async def basic_transcribe():
    # Set up our client with your chosen Region
    client = TranscribeStreamingClient(region = "us-west-2")

    # Start transcription to generate async stream
    stream = await client.start_stream_transcription(
```

```
        language_code = "en-US",
        media_sample_rate_hz = 16000,
        media_encoding = "pcm",
    )

    async def write_chunks():
        # NOTE: For pre-recorded files longer than 5 minutes, the sent audio
        # chunks should be rate limited to match the real-time bitrate of the
        # audio stream to avoid signing issues.
        async with aiofile.AIOFile('filepath/my-media-file.flac', 'rb') as afp:
            reader = aiofile.Reader(afp, chunk_size = 1024 * 16)
            async for chunk in reader:
                await stream.input_stream.send_audio_event(audio_chunk = chunk)
        await stream.input_stream.end_stream()

    # Instantiate our handler and start processing events
    handler = MyEventHandler(stream.output_stream)
    await asyncio.gather(write_chunks(), handler.handle_events())

loop = asyncio.get_event_loop()
loop.run_until_complete(basic_transcribe())
loop.close()
```

C++

Refer to the Code examples chapter for the streaming C++ SDK example .

Using this service with an AWS SDK

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

For examples specific to this service, see Code examples for Amazon Transcribe using AWS SDKs.

Transcribing with HTTP or WebSockets

Amazon Transcribe supports HTTP for both batch (HTTP/1.1) and streaming (HTTP/2) transcriptions. WebSockets are supported for streaming transcriptions.

If you're transcribing a media file located in an Amazon S3 bucket, you're performing a batch transcription. If you're transcribing a real-time stream of audio data, you're performing a streaming transcription.

Both HTTP and WebSockets require you to authenticate your request using AWS Signature Version 4 headers. Refer to Signing AWS API requests for more information.

Batch transcriptions

You can make a batch HTTP request using the following headers:

- host
- x-amz-target
- content-type
- x-amz-content-sha256
- x-amz-date
- authorization

Here's an example of a `StartTranscriptionJob` request:

```
POST /transcribe HTTP/1.1
host: transcribe.us-west-2.amazonaws.com
x-amz-target: com.amazonaws.transcribe.Transcribe.StartTranscriptionJob
content-type: application/x-amz-json-1.1
x-amz-content-sha256: string
x-amz-date: YYYYMMDDTHHMMSSZ
authorization: AWS4-HMAC-SHA256 Credential=access-key/YYYYMMSS/us-west-2/transcribe/
aws4_request, SignedHeaders=content-type;host;x-amz-content-sha256;x-amz-date;x-amz-
target;x-amz-security-token, Signature=string

{
    "TranscriptionJobName": "my-first-transcription-job",
    "LanguageCode": "en-US",
    "Media": {
        "MediaFileUri": "s3://amzn-s3-demo-bucket/my-input-files/my-media-file.flac"
    },
    "OutputBucketName": "amzn-s3-demo-bucket",
    "OutputKey": "my-output-files/"
}
```

Additional operations and parameters are listed in the API Reference; parameters common to all AWS API operations are listed in the Common Parameters section. Other signature elements are detailed in Elements of an AWS Signature Version 4 request.

Streaming transcriptions

Streaming transcriptions using HTTP/2 and WebSockets are more involved than using SDKs. We recommend reviewing the Setting up a streaming transcription section before setting up your first stream.

For more information on these methods, refer to Setting up an HTTP/2 stream or Setting up a WebSocket stream.

> **Note**
>
> We strongly recommend using an SDK for streaming transcriptions. For a list of supported SDKs, refer to Supported programming languages.

Transcribing streaming audio

Using Amazon Transcribe streaming, you can produce real-time transcriptions for your media content. Unlike batch transcriptions, which involve uploading media files, streaming media is delivered to Amazon Transcribe in real time. Amazon Transcribe then returns a transcript, also in real time.

Streaming can include pre-recorded media (movies, music, and podcasts) and real-time media (live news broadcasts). Common streaming use cases for Amazon Transcribe include live closed captioning for sporting events and real-time monitoring of call center audio.

Streaming content is delivered as a series of sequential data packets, or 'chunks,' that Amazon Transcribe transcribes instantaneously. The advantages of using streaming over batch include real-time speech-to-text capabilities in your applications and faster transcription times. However, this increased speed may have accuracy limitations in some cases.

Amazon Transcribe offers the following options for streaming:

- SDKs (preferred)
- HTTP/2
- WebSockets
- AWS Management Console

To transcribe streaming audio in the AWS Management Console, speak into your computer microphone.

> **Tip**
>
> For SDK code examples, refer to the AWS Samples repository on GitHub.

Audio formats supported for streaming transcriptions are:

- FLAC
- OPUS-encoded audio in an Ogg container
- PCM (only signed 16-bit little-endian audio formats, which does **not** include WAV)

Lossless formats (FLAC or PCM) are recommended.

> **Note**
>
> Streaming transcriptions are not supported with all languages. Refer to the 'Data input' column in the supported languages table for details.

To view the Amazon Transcribe Region availability for streaming transcriptions, see: Amazon Transcribe Endpoints and Quotas.

Best practices

The following recommendations improve streaming transcription efficiency:

- If possible, use PCM-encoded audio.
- Ensure that your stream is as close to real-time as possible.
- Latency depends on the size of your audio chunks. If you're able to specify chunk size with your audio type (such as with PCM), set each chunk to between 50 ms and 200 ms. You can calculate the audio chunk size by the following formula:

```
chunk_size_in_bytes = chunk_duration_in_millisecond / 1000 * audio_sample_rate * 2
```

- Use a uniform chunk size.
- Make sure you correctly specify the number of audio channels.
- With single-channel PCM audio, each sample consists of two bytes, so each chunk should consist of an even number of bytes.
- With dual-channel PCM audio, each sample consists of four bytes, so each chunk should be a multiple of 4 bytes.
- When your audio stream contains no speech, encode and send the same amount of silence. For example, silence for PCM is a stream of zero bytes.
- Make sure you specify the correct sampling rate for your audio. If possible, record at a sampling rate of 16,000 Hz; this provides the best compromise between quality and data volume sent over the network. Note that most high-end microphones record at 44,100 Hz or 48,000 Hz.

Handling LimitExceededException errors

As with any distributed system, Amazon Transcribe has protection mechanisms that detect resource over-consumption and react accordingly. `LimitExceededException` errors can occur when one of these mechanisms is triggered. There are three distinct causes for this error:

Concurrent stream service quota exceeded

This is the most common cause. It occurs when you exceed your concurrent stream service quota. To resolve this error, retry with exponential backoff. If you consistently hit this limit, request a service quota increase through the Service Quotas console. You can also contact the AWS Support Center for assistance. For more information about retry strategies, see Retry behavior in the *AWS SDKs and Tools Reference Guide*.

Maximum session duration exceeded

This error occurs when a stream exceeds the maximum allowed session duration. This is a hard limit that cannot be increased. To continue transcribing, start a new streaming session.

Number of concurrent streams increased too quickly

This is a rare cause. It can occur if you increase your number of concurrent streams too quickly, for example during load tests. This is a system-level protection mechanism with no adjustable quota. To resolve this error, retry with exponential backoff and gradually increase your number of concurrent streams. For more information about retry strategies, see Retry behavior in the *AWS SDKs and Tools Reference Guide*. You can also visit AWS re:Post or contact AWS Premium Support.

Streaming and partial results

Because streaming works in real time, transcripts are produced in *partial results*. Amazon Transcribe breaks up the incoming audio stream based on natural speech segments, such as a change in speaker or a pause in the audio. The transcription is returned to your application in a stream of transcription events, with each response containing more transcribed speech until an entire segment is transcribed.

An approximation of this is shown in the following code block. You can view this process in action by signing into the AWS Management Console, selecting **Real-time transcription**, and speaking into your microphone. Watch the **Transcription output** pane as you speak.

In this example, each line is the partial result of an audio segment.

```
The
The Amazon.
The Amazon is
The Amazon is the law.
The Amazon is the largest
The Amazon is the largest ray
The Amazon is the largest rain for
The Amazon is the largest rainforest.
The Amazon is the largest rainforest on the
The Amazon is the largest rainforest on the planet.
```

These partial results are present in your transcription output within the `Results` objects. Also in this object block is an **IsPartial** field. If this field is true, your transcription segment is not yet complete. You can view the difference between an incomplete and a complete segment below:

```
"IsPartial": true (incomplete segment)

"Transcript": "The Amazon is the largest rainforest."

"EndTime": 4.545,
"IsPartial": true,
"ResultId": "12345a67-8bc9-0de1-2f34-a5b678c90d12",
"StartTime": 0.025


"IsPartial": false (complete segment)

"Transcript": "The Amazon is the largest rainforest on the planet."

"EndTime": 6.025,
"IsPartial": false,
"ResultId": "34567e89-0fa1-2bc3-4d56-78e90123456f",
"StartTime": 0.025
```

Each word within a *complete* segment has an associated confidence score, which is a value between 0 and 1. A larger value indicates a greater likelihood that the word is correctly transcribed.

> **Tip**
>
> The `StartTime` and `EndTime` of an audio segment can be used to synchronize transcription output with video dialogue.

If you're running an application that requires low latency, you may want to use partial-result stabilization.

Partial-result stabilization

Amazon Transcribe starts returning transcription results as soon as you start streaming your audio. It returns these partial results incrementally until it generates a finished result at the level of a natural speech segment. A natural speech segment is continuous speech that contains a pause or a change in speaker.

Amazon Transcribe continues outputting partial results until it generates the final transcription result for a speech segment. Because speech recognition may revise words as it gains more context, streaming transcriptions can change slightly with each new partial result output.

This process gives you two options for each speech segment:

- Wait for the finished segment
- Use the segment's partial results

Partial result stabilization changes how Amazon Transcribe produces the final transcription result for each complete segment. When activated, only the last few words from the partial results can change. Because of this, transcription accuracy may be affected. However, your transcript is returned faster than without partial-results stabilization. This reduction in latency may be beneficial when subtitling videos or generating captions for live streams.

The following examples show how the same audio stream is handled when partial-results stabilization is not activated and when it is. Note that you can set the stability level to low, medium, or high. Low stability provides the highest accuracy. High stability transcribes faster, but with slightly lower accuracy.

"Transcript":	"EndTime":	"IsPartial":
Partial-result stabilization not enabled		
The The The Amazon. The Amazon is The Amazon is the law. The Amazon is the largest The Amazon is the largest ray The Amazon is the largest rain for The Amazon is the largest rainforest. The Amazon is the largest rainforest on the The Amazon is the largest rainforest on the planet. The Amazon is the largest rainforest on the planet. The Amazon is the largest rainforest on the planet.	0.545 1.045 1.545 2.045 2.545 3.045 3.545 4.045 4.545 5.045 5.545 6.025 6.025	true true true true true true true true true true true true false
Partial-result stabilization enabled (high stability)		
The The The Amazon. The Amazon is The Amazon is the large The Amazon is the largest The Amazon is the largest rainfall.	0.515 1.015 1.515 2.015 2.515 3.015 3.515 4.015 4.515 5.015	true true true true true true true true true true

"Transcript":	"EndTime":	"IsPartial":

`The Amazon is the largest rain forest.`
`The Amazon is the largest rain forest on`
`The Amazon is the largest rain forest on the planet.`
`The Amazon is the largest rain forest on the planet.`
`The Amazon is the largest rain forest on the planet.`
`The Amazon is the largest rain forest on the planet.`
`The Amazon is the largest rain forest on the planet.`
	`5.515`
	`6.015`
	`6.335`
	`6.335`
		`true`
		`true`
		`true`
		`false`

When you activate partial-result stabilization, Amazon Transcribe uses a `Stable` field to indicate whether an item is stable, where 'item' refers to a transcribed word or punctuation mark. Values for `Stable` are `true` or `false`. Items flagged as `false` (not stable) are more likely to change as your segment is transcribed. Conversely, items flagged as `true` (stable) won't change.

You can choose to render non-stable words so your captions align with speech. Even if captions change slightly as context is added, this is a better user experience than periodic text bursts, which may or may not align with speech.

You can also choose to display non-stable words in a different format, such as italics, to indicate to viewers that these words may change. Displaying partial results limits the amount of text displayed at a given time. This can be important when you're dealing with space constraints, as with video captions.

Dive deeper with the AWS Machine Learning Blog

To learn more about improving accuracy with real-time transcriptions, see:

- Improve the streaming transcription experience with Amazon Transcribe partial results stabilization
- "What was that?" Increasing subtitle accuracy for live broadcasts using Amazon Transcribe

Partial-result stabilization example output

The following example output shows `Stable` flags for an incomplete segment (`"IsPartial": true`). You can see that the words "*to*" and "*Amazon*" are not stable and therefore could change before the segment is finalized.

```
"Transcript": {
    "Results": [
        {
            "Alternatives": [
                {
                    "Items": [
                        {
                            "Content": "Welcome",
                            "EndTime": 2.4225,
                            "Stable": true,
                            "StartTime": 1.65,
                            "Type": "pronunciation",
                            "VocabularyFilterMatch": false
                        },
                        {
                            "Content": "to",
                            "EndTime": 2.8325,
                            "Stable": false,
                            "StartTime": 2.4225,
                            "Type": "pronunciation",
                            "VocabularyFilterMatch": false
                        },
                        {
                            "Content": "Amazon",
                            "EndTime": 3.635,
                            "Stable": false,
                            "StartTime": 2.8325,
                            "Type": "pronunciation",
                            "VocabularyFilterMatch": false
```

```
                        },
                        {
                            "Content": ".",
                            "EndTime": 3.635,
                            "Stable": false,
                            "StartTime": 3.635,
                            "Type": "punctuation",
                            "VocabularyFilterMatch": false
                        }
                    ],
                    "Transcript": "Welcome to Amazon."
                }
            ],
            "EndTime": 4.165,
            "IsPartial": true,
            "ResultId": "12345a67-8bc9-0de1-2f34-a5b678c90d12",
            "StartTime": 1.65
        }
    ]
}
```

Setting up a streaming transcription

This section expands on the main streaming section. It's intended to provide information for users who want to set up their stream with HTTP/2 or WebSockets directly, rather than with an AWS SDK. The information in this section can also be used to build your own SDK.

> **Important**
>
> We strongly recommend using SDKs rather than using HTTP/2 and WebSockets directly. SDKs are the simplest and most reliable method for transcribing data streams. To start streaming using an AWS SDK, see Transcribing with the AWS SDKs.

Setting up an HTTP/2 stream

The key components for an HTTP/2 protocol for streaming transcription requests with Amazon Transcribe are:

- A header frame. This contains the HTTP/2 headers for your request, and a signature in the authorization header that Amazon Transcribe uses as a seed signature to sign the data frames.

- One or more message frames in event stream encoding that contain metadata and raw audio bytes.
- An end frame. This is a signed message in event stream encoding with an empty body.

> **Note**
>
> Amazon Transcribe only supports one stream per HTTP/2 session. If you attempt to use multiple streams, your transcription request fails.

1. Attach the following policy to the IAM role that makes the request. See Adding IAM policies for more information.

   JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Sid": "myTranscribeHttp2Policy",
            "Effect": "Allow",
            "Action": "transcribe:StartStreamTranscription",
            "Resource": "*"
        }
    ]
}
```

2. To start the session, send an HTTP/2 request to Amazon Transcribe.

```
POST /stream-transcription HTTP/2
host: transcribestreaming.us-west-2.amazonaws.com
X-Amz-Target: com.amazonaws.transcribe.Transcribe.StartStreamTranscription
Content-Type: application/vnd.amazon.eventstream
X-Amz-Content-Sha256: string
X-Amz-Date: YYYYMMDDTHHMMSSZ
Authorization: AWS4-HMAC-SHA256 Credential=access-key/YYYYMMDD/us-west-2/
transcribe/aws4_request, SignedHeaders=content-type;host;x-amz-content-sha256;x-
amz-date;x-amz-target;x-amz-security-token, Signature=string
x-amzn-transcribe-language-code: en-US
x-amzn-transcribe-media-encoding: flac
```

```
x-amzn-transcribe-sample-rate: 16000
transfer-encoding: chunked
```

Additional operations and parameters are listed in the API Reference; parameters common to all AWS API operations are listed in the Common Parameters section.

Amazon Transcribe sends the following response:

```
HTTP/2.0 200
x-amzn-transcribe-language-code: en-US
x-amzn-transcribe-media-encoding: flac
x-amzn-transcribe-sample-rate: 16000
x-amzn-request-id: 8a08df7d-5998-48bf-a303-484355b4ab4e
x-amzn-transcribe-session-id: b4526fcf-5eee-4361-8192-d1cb9e9d6887
content-type: application/json
```

3. Create an audio event that contains your audio data. Combine the headers—described in the following table—with a chunk of audio bytes in an event-encoded message. To create the payload for the event message, use a buffer in raw-byte format.

Header name byte length	Header name (string)	Header value type	Value string byte length	Value string (UTF-8)
13	:content-type	7	24	application/ octet-stream
11	:event-type	7	10	AudioEvent
13	:message-type	7	5	event

Binary data in this example request are base64-encoded. In an actual request, data are raw bytes.

```
:content-type: "application/vnd.amazon.eventstream"
:event-type: "AudioEvent"
:message-type: "event"
UklGRjzxPQBXQVZFZm10IBAAAAABAAEAgD4AAAB9AAACABAAZGF0YVTwPQAAAAAAAAAAAAAAAAD//wIA/
f8EAA==
```

4. Create an audio message that contains your audio data.

   a. Your audio message data frame contains event-encoding headers that include the current date and a signature for the audio chunk and the audio event.

Header name byte length	Header name (string)	Header value type	Value string byte length	Value
16	:chunk-signature	6	varies	generated signature
5	:date	8	8	timestamp

   Binary data in this request are base64-encoded. In an actual request, data are raw bytes.

```
:date: 2019-01-29T01:56:17.291Z
:chunk-signature: signature

AAAA0gAAAIKVoRFcTTcjb250ZW50LXR5cGUHABhhcHBsaWNhdGlvbi9vY3RldC1zdHJlYW0LOmV2ZW50LXR5
cGUHAApBdWRpb0V2ZW50DTptZXNzYWdlLXR5cGUHAAVldmVudAxDb256ZW50LVR5cGUHABphcHBsaWNhdGlv
bi94LWFtei1qc29uLTEuMVJJRkY88T0AV0FWRWZtdCAQAAAAAQABAIA
+AAAAfQAAAgAQAGRhdGFU8D0AAAAA
AAAAAAAAAAAA//8CAP3/BAC7QLFf
```

   b. Construct a string to sign, as outlined in Create a string to sign for Signature Version 4. Your string follows this format:

```
String stringToSign =
"AWS4-HMAC-SHA256" +
"\n" +
DateTime +
"\n" +
Keypath +
"\n" +
Hex(priorSignature) +
"\n" +
HexHash(nonSignatureHeaders) +
"\n" +
HexHash(payload);
```

- **DateTime**: The date and time the signature is created. The format is YYYYMMDDTHHMMSSZ, where YYYY=year, MM=month, DD=day, HH=hour, MM=minute, SS=seconds, and 'T' and 'Z' are fixed characters. For more information, refer to Handling Dates in Signature Version 4.
- **Keypath**: The signature scope in the format `date/region/service/aws4_request`. For example, `20220127/us-west-2/transcribe/aws4_request`.
- **Hex**: A function that encodes input into a hexadecimal representation.
- **priorSignature**: The signature for the previous frame. For the first data frame, use the signature of the header frame.
- **HexHash**: A function that first creates a SHA-256 hash of its input and then uses the Hex function to encode the hash.
- **nonSignatureHeaders**: The DateTime header encoded as a string.
- **payload**: The byte buffer containing the audio event data.

c. Derive a signing key from your AWS secret access key and use it to sign the `stringToSign`. For a greater degree of protection, the derived key is specific to the date, service, and AWS Region. For more information, see Calculate the signature for AWSSignature Version 4.

Make sure you implement the `GetSignatureKey` function to derive your signing key. If you have not yet derived a signing key, refer to Examples of how to derive a signing key for Signature Version 4.

```
String signature = HMACSHA256(derivedSigningKey, stringToSign);
```

- **HMACSHA256**: A function that creates a signature using the SHA-256 hash function.
- **derivedSigningKey**: The Signature Version 4 signing key.
- **stringToSign**: The string you calculated for the data frame.

After you've calculated the signature for the data frame, construct a byte buffer containing the date, signature, and audio event payload. Send the byte array to Amazon Transcribe for transcription.

5. To indicate the audio stream is complete, send an end frame (an empty data frame) that contains only the date and signature. You construct this end frame the same way that you construct a data frame.

Amazon Transcribe responds with a stream of transcription events, sent to your application. This response is event stream encoded. It contains the standard prelude and the following headers.

Header name byte length	Header name (string)	Header value type	Value string byte length	Value string (UTF-8)
13	:content-type	7	16	application/json
11	:event-type	7	15	TranscriptEvent
13	:message-type	7	5	event

The events are sent in raw-byte format. In this example, the bytes are base64-encoded.

```
AAAAUwAAAEP1RHpYBTpkYXRlCAAAAWiXUkMLEDpjaHVuay1zaWduYXR1cmUGACCt6Zy+uymwEK2SrLp/
zVBI
5eGn83jdBwCaRUBJA+eaDafqjqI=
```

To see the transcription results, decode the raw bytes using event stream encoding.

```
:content-type: "application/vnd.amazon.eventstream"
:event-type: "TranscriptEvent"
:message-type: "event"

{
    "Transcript":
        {
            "Results":
                [
                    results
                ]
        }
}
```

6. To end your stream, send an empty audio event to Amazon Transcribe. Create the audio event exactly like any other, except with an empty payload. Sign the event and include the signature in the `:chunk-signature` header, as follows:

```
:date: 2019-01-29T01:56:17.291Z
:chunk-signature: signature
```

Handling HTTP/2 streaming errors

If an error occurs when processing your media stream, Amazon Transcribe sends an exception response. The response is event stream encoded.

The response contains the standard prelude and the following headers:

Header name byte length	Header name (string)	Header value type	Value string byte length	Value string (UTF-8)
13	:content-type	7	16	application/json
11	:event-type	7	19	BadRequestException
13	:message-type	7	9	exception

When the exception response is decoded, it contains the following information:

```
:content-type: "application/vnd.amazon.eventstream"
:event-type: "BadRequestException"
:message-type: "exception"

Exception message
```

Setting up a WebSocket stream

The key components for a WebSocket protocol for streaming transcription requests with Amazon Transcribe are:

- The upgrade request. This contains the query parameters for your request, and a signature that Amazon Transcribe uses as a seed signature to sign the data frames.

- One or more message frames in event stream encoding that contain metadata and raw audio bytes.
- An end frame. This is a signed message in event stream encoding with an empty body.

> **Note**
>
> Amazon Transcribe only supports one stream per WebSocket session. If you attempt to use multiple streams, your transcription request fails.

1. Attach the following policy to the IAM role that makes the request. See Adding IAM policies for more information.

   JSON

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

2. To start the session, create a presigned URL in the following format. Line breaks have been added for readability.

   ```
   GET wss://transcribestreaming.us-west-2.amazonaws.com:8443/stream-transcription-
   websocket?
   &X-Amz-Algorithm=AWS4-HMAC-SHA256
   &X-Amz-Credential=access-key%2FYYYYMMDD%2Fus-west-2%2Ftranscribe%2Faws4_request
   &X-Amz-Date=YYYYMMDDTHHMMSSZ
   &X-Amz-Expires=300
   &X-Amz-Security-Token=security-token
   &X-Amz-Signature=string
   &X-Amz-SignedHeaders=content-type%3Bhost%3Bx-amz-date
   &language-code=en-US
   ```

```
&media-encoding=flac
&sample-rate=16000
```

> **Note**
>
> The maximum value for `X-Amz-Expires` is 300 (5 minutes).

Additional operations and parameters are listed in the API Reference; parameters common to all AWS API operations are listed in the Common Parameters section.

To construct the URL for your request and create the Signature Version 4 signature, refer to the following steps. Examples are in pseudocode.

a. Create a canonical request. A canonical request is a string that includes information from your request in a standardized format. This ensures that when AWS receives the request, it can calculate the same signature you created for your URL. For more information, see Create a Canonical Request for Signature Version 4.

```
# HTTP verb
method = "GET"
# Service name
service = "transcribe"
# Region
region = "us-west-2"
# Amazon Transcribe streaming endpoint
endpoint = "wss://transcribestreaming.us-west-2.amazonaws.com:8443"
# Host
host = "transcribestreaming.us-west-2.amazonaws.com:8443"
# Date and time of request
amz-date = YYYYMMDDTHHMMSSZ
# Date without time for credential scope
datestamp = YYYYMMDD
```

b. Create a canonical URI, which is the part of the URI between the domain and the query string.

```
canonical_uri = "/stream-transcription-websocket"
```

c. Create the canonical headers and signed headers. Note the trailing \n in the canonical headers.

- Append the lowercase header name followed by a colon (:).
- Append a comma-separated list of values for that header. Do not sort values in headers that have multiple values.
- Append a new line (\n).

```
canonical_headers = "host:" + host + "\n"
signed_headers = "host"
```

d. Match the algorithm to the hashing algorithm. Use SHA-256.

```
algorithm = "AWS4-HMAC-SHA256"
```

e. Create the credential scope, which scopes the derived key to the date, AWS Region, and service. For example, *20220127*/*us-west-2*/transcribe/aws4_request.

```
credential_scope = datestamp + "/" + region + "/" + service + "/" +
 "aws4_request"
```

f. Create the canonical query string. Query string values must be URI-encoded and sorted by name.

- Sort the parameter names by character code point in ascending order. Parameters with duplicate names should be sorted by value. For example, a parameter name that begins with the uppercase letter F precedes a parameter name that begins with the lowercase letter b.
- Do not URI-encode any of the unreserved characters that RFC 3986 defines: A-Z, a-z, 0-9, hyphen (-), underscore (_), period (.), and tilde (~).
- Percent-encode all other characters with %XY, where X and Y are hexadecimal characters (0-9 and uppercase A-F). For example, the space character must be encoded as %20 (don't include '+', as some encoding schemes do); extended UTF-8 characters must be in the form %XY%ZA%BC.
- Double-encode any equals (=) characters in parameter values.

```
canonical_querystring  = "X-Amz-Algorithm=" + algorithm
canonical_querystring += "&X-Amz-Credential="+ URI-encode(access key + "/" +
 credential_scope)
canonical_querystring += "&X-Amz-Date=" + amz_date
canonical_querystring += "&X-Amz-Expires=300"
canonical_querystring += "&X-Amz-Security-Token=" + token
canonical_querystring += "&X-Amz-SignedHeaders=" + signed_headers
canonical_querystring += "&language-code=en-US&media-encoding=flac&sample-
rate=16000"
```

g. Create a hash of the payload. For a GET request, the payload is an empty string.

```
payload_hash = HashSHA256(("").Encode("utf-8")).HexDigest()
```

h. Combine the following elements to create the canonical request.

```
canonical_request = method + '\n'
   + canonical_uri + '\n'
   + canonical_querystring + '\n'
   + canonical_headers + '\n'
   + signed_headers + '\n'
   + payload_hash
```

3. Create the string to sign, which contains meta information about your request. You use the string to sign in the next step when you calculate the request signature. For more information, see Create a String to Sign for Signature Version 4.

```
string_to_sign=algorithm + "\n"
   + amz_date + "\n"
   + credential_scope + "\n"
   + HashSHA256(canonical_request.Encode("utf-8")).HexDigest()
```

4. Calculate the signature. To do this, derive a signing key from your AWS secret access key. For a greater degree of protection, the derived key is specific to the date, service, and AWS Region. Use this derived key to sign the request. For more information, see Calculate the Signature for AWS Signature Version 4.

   Make sure you implement the `GetSignatureKey` function to derive your signing key. If you have not yet derived a signing key, refer to Examples of how to derive a signing key for Signature Version 4.

```
#Create the signing key
signing_key = GetSignatureKey(secret_key, datestamp, region, service)

# Sign the string_to_sign using the signing key
signature = HMAC.new(signing_key, (string_to_sign).Encode("utf-8"),
 Sha256()).HexDigest
```

The function `HMAC(key, data)` represents an HMAC-SHA256 function that returns results in binary format.

5. Add signing information to the request and create the request URL.

   After you calculate the signature, add it to the query string. For more information, see Add the Signature to the Request.

   First, add the authentication information to the query string.

   ```
   canonical_querystring += "&X-Amz-Signature=" + signature
   ```

   Second, create the URL for the request.

   ```
   request_url = endpoint + canonical_uri + "?" + canonical_querystring
   ```

   Use the request URL with your WebSocket library to make the request to Amazon Transcribe.

6. The request to Amazon Transcribe must include the following headers. Typically these headers are managed by your WebSocket client library.

   ```
   Host: transcribestreaming.us-west-2.amazonaws.com:8443
   Connection: Upgrade
   Upgrade: websocket
   Origin: URI-of-WebSocket-client
   Sec-WebSocket-Version: 13
   Sec-WebSocket-Key: randomly-generated-string
   ```

7. When Amazon Transcribe receives your WebSocket request, it responds with a WebSocket upgrade response. Typically your WebSocket library manages this response and sets up a socket for communications with Amazon Transcribe.

The following is the response from Amazon Transcribe. Line breaks have been added for readability.

```
HTTP/1.1 101 WebSocket Protocol Handshake

Connection: upgrade
Upgrade: websocket
websocket-origin: wss://transcribestreaming.us-west-2.amazonaws.com:8443
websocket-location: transcribestreaming.us-west-2.amazonaws.com:8443/stream-
transcription-websocket?
&X-Amz-Algorithm=AWS4-HMAC-SHA256
&X-Amz-Credential=AKIAIOSFODNN7EXAMPLE%2F20220208%2Fus-west-2%2Ftranscribe
%2Faws4_request
&X-Amz-Date=20220208T235959Z
&X-Amz-Expires=300
&X-Amz-Signature=Signature Version 4 signature
&X-Amz-SignedHeaders=host
&language-code=en-US
&session-id=String
&media-encoding=flac
&sample-rate=16000
x-amzn-RequestId: RequestId
Strict-Transport-Security: max-age=31536000
sec-websocket-accept: hash-of-the-Sec-WebSocket-Key-header
```

8. Make your WebSocket streaming request.

   After the WebSocket connection is established, the client can start sending a sequence of audio frames, each encoded using event stream encoding.

   Each data frame contains three headers combined with a chunk of raw audio bytes; the following table describes these headers.

Header name byte length	Header name (string)	Header value type	Value string byte length	Value string (UTF-8)
13	:content-type	7	24	application/ octet-stream
11	:event-type	7	10	AudioEvent

Header name byte length	Header name (string)	Header value type	Value string byte length	Value string (UTF-8)
13	:message-type	7	5	event

9. To end the data stream, send an empty audio chunk in an event stream encoded message.

   The response contains event stream encoded raw bytes in the payload. It contains the standard prelude and the following headers.

Header name byte length	Header name (string)	Header value type	Value string byte length	Value string (UTF-8)
13	:content-type	7	16	application/json
11	:event-type	7	15	TranscriptEvent
13	:message-type	7	5	event

   When you decode the binary response, you end up with a JSON structure containing the transcription results.

Handling WebSocket streaming errors

If an exception occurs while processing your request, Amazon Transcribe responds with a terminal WebSocket frame containing an event stream encoded response. This response contains the headers described in the following table; the body of the response contains a descriptive error message. After sending the exception response, Amazon Transcribe sends a close frame.

Header name byte length	Header name (string)	Header value type	Value string byte length	Value string (UTF-8)
13	:content-type	7	16	application/json
15	:exception-type	7	varies	varies, see below
13	:message-type	7	9	exception

The `exception-type` header contains one of the following values:

- **`BadRequestException`**: There was a client error when the stream was created, or an error occurred while streaming data. Make sure that your client is ready to accept data and try your request again.
- **`InternalFailureException`**: Amazon Transcribe had a problem during the handshake with the client. Try your request again.
- **`LimitExceededException`**: The client exceeded the concurrent stream limit. For more information, see Amazon Transcribe Limits. Reduce the number of streams that you're transcribing.
- **`UnrecognizedClientException`**: The WebSocket upgrade request was signed with an incorrect access key or secret key. Make sure you're correctly creating the access key and try your request again.

Amazon Transcribe can also return any of the common service errors. For a list, see Common Errors.

Event stream encoding

Amazon Transcribe uses a format called event stream encoding for streaming transcriptions.

Event stream encoding provides bidirectional communication between a client and a server. Data frames sent to the Amazon Transcribe streaming service are encoded in this format. The response from Amazon Transcribe also uses this encoding.

Each message consists of two sections: the prelude and the data. The prelude consists of:

1. The total byte length of the message
2. The combined byte length of all headers

The data section consists of:

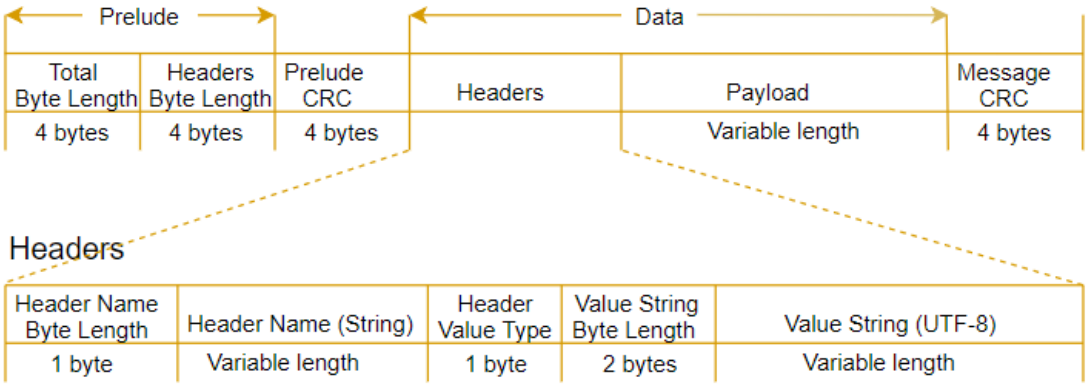
1. Headers
2. Payload

Each section ends with a 4-byte big-endian integer cyclic redundancy check (CRC) checksum. The message CRC checksum is for both the prelude section and the data section. Amazon Transcribe

uses CRC32 (often referred to as GZIP CRC32) to calculate both CRCs. For more information about CRC32, see *GZIP file format specification version 4.3*.

Total message overhead, including the prelude and both checksums, is 16 bytes.

The following diagram shows the components that make up a message and a header. There are multiple headers per message.



Each message contains the following components:

- **Prelude**: Consists of two, 4-byte fields, for a fixed total of 8 bytes.
  - *First 4 bytes*: The big-endian integer byte-length of the entire message, inclusive of this 4-byte length field.
  - *Second 4 bytes*: The big-endian integer byte-length of the 'headers' portion of the message, excluding the 'headers' length field itself.
- **Prelude CRC**: The 4-byte CRC checksum for the prelude portion of the message, excluding the CRC itself. The prelude has a separate CRC from the message CRC. That ensures that Amazon Transcribe can detect corrupted byte-length information immediately without causing errors, such as buffer overruns.
- **Headers**: Metadata annotating the message; for example, message type and content type. Messages have multiple headers, which are key:value pairs, where the key is a UTF-8 string. Headers can appear in any order in the 'headers' portion of the message, and each header can appear only once.
- **Payload**: The audio content to be transcribed.
- **Message CRC**: The 4-byte CRC checksum from the start of the message to the start of the checksum. That is, everything in the message except the CRC itself.

The header frame is the authorization frame for the streaming transcription. Amazon Transcribe uses the authorization header's value as the seed for generating a chain of authorization headers for the data frames in the request.

Each header contains the following components; there are multiple headers per frame.

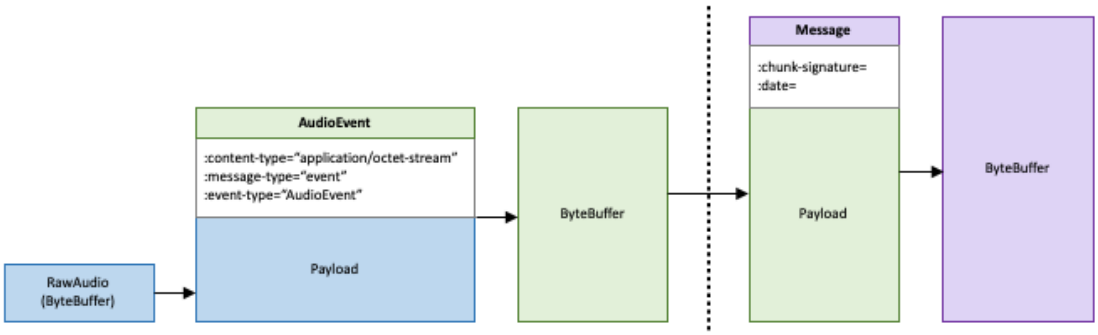
- **Header name byte-length**: The byte-length of the header name.
- **Header name**: The name of the header that indicates the header type. For valid values, see the following frame descriptions.
- **Header value type**: A number indicating the header value. The following list shows the possible values for the header and what they indicate.
  - 0 – TRUE
  - 1 – FALSE
  - 2 – BYTE
  - 3 – SHORT
  - 4 – INTEGER
  - 5 – LONG
  - 6 – BYTE ARRAY
  - 7 – STRING
  - 8 – TIMESTAMP
  - 9 – UUID
- **Value string byte length**: The byte length of the header value string.
- **Header value**: The value of the header string. Valid values for this field depend on the type of header. See Setting up an HTTP/2 stream or Setting up a WebSocket stream for more information.

Data frames

Each streaming request contains one or more data frames. There are two steps to creating a data frame:

1. Combine raw audio data with metadata to create the payload of your request.
2. Combine the payload with a signature to form the event message that is sent to Amazon Transcribe.

The following diagram shows how this works.



Job queueing

Using job queueing, you can submit more transcription job requests than can be concurrently processed. Without job queueing, once you reach the quota of allowed concurrent requests, you must wait until one or more requests are completed before submitting a new request.

Job queueing is optional for both transcription job and post-call analytics job requests.

If you enable job queueing, Amazon Transcribe creates a queue that contains all requests that exceed your limit. As soon as a request is completed, a new request is pulled from your queue and processed. Queued requests are processed in a FIFO (first in, first out) order.

You can add up to 10,000 jobs to your queue. If you exceed this limit, you get a `LimitExceededConcurrentJobException` error. To maintain optimal performance, Amazon Transcribe only uses up to 90 percent of your quota (a bandwidth ratio of 0.9) to process queued jobs. Note that these are default values that can be increased upon request.

> **Tip**
>
> You can find a list of default limits and quotas for Amazon Transcribe resources in the AWS General Reference. Some of these defaults can be increased upon request.

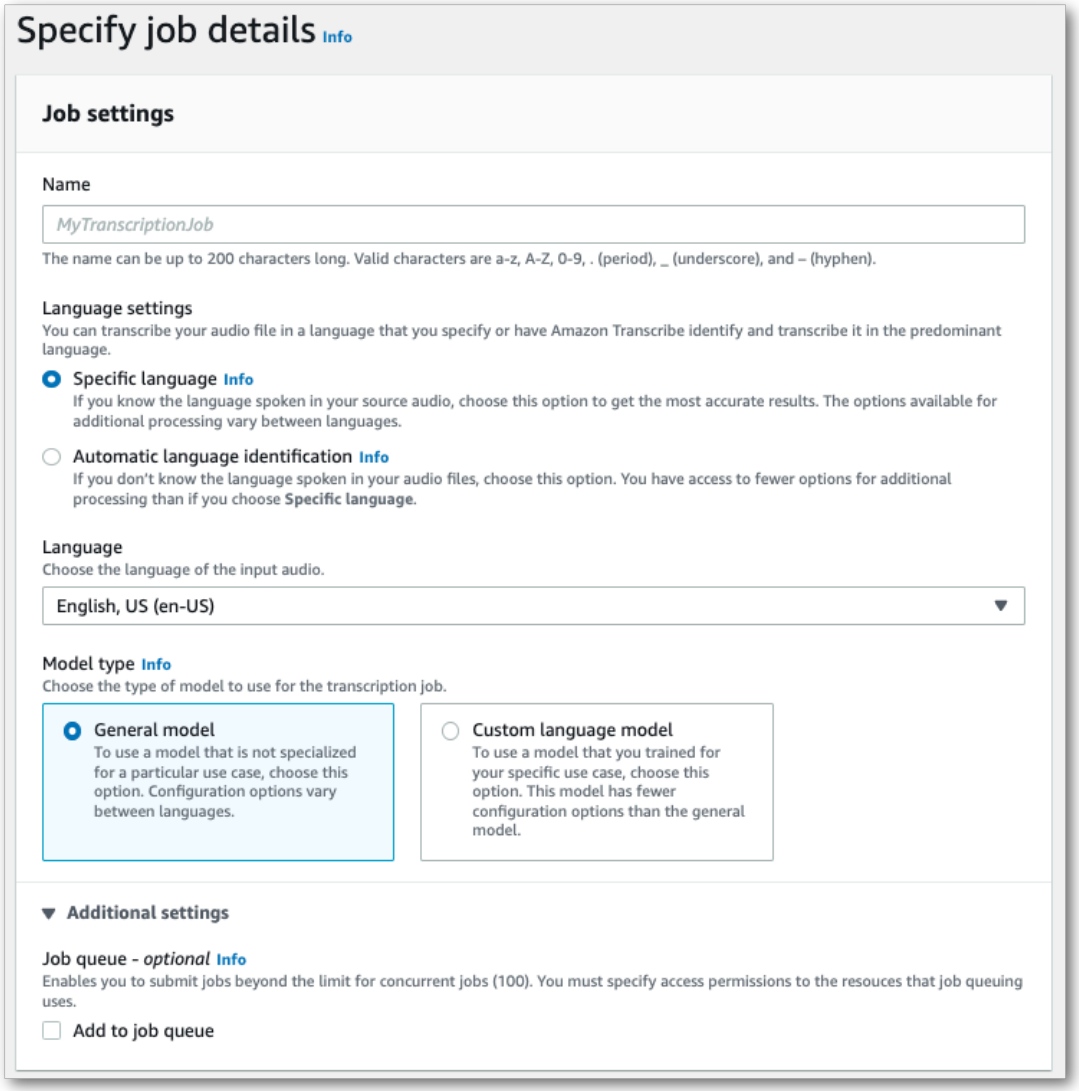
If you enable job queueing but don't exceed the quota for concurrent requests, all requests are processed concurrently.

Enabling job queueing

You can enable job queueing using the **AWS Management Console**, **AWS CLI**, or **AWS SDKs**; see the following for examples; see the following for examples:

AWS Management Console

1. Sign in to the AWS Management Console.
2. In the navigation pane, choose **Transcription jobs**, then select **Create job** (top right). This opens the **Specify job details** page.
3. In the **Job Settings** box, there is an **Additional settings** panel. If you expand this panel, you can select the **Add to job queue** box to enable job queueing.



4. Fill in any other fields you want to include on the **Specify job details** page, then select **Next**. This takes you to the **Configure job - *optional* page**.
5. Select **Create job** to run your transcription job.

AWS CLI

This example uses the start-transcription-job command and `job-execution-settings` parameter with the `AllowDeferredExecution` sub-parameter. Note that when you include `AllowDeferredExecution` in your request, you must also include `DataAccessRoleArn`.

For more information, see `StartTranscriptionJob` and `JobExecutionSettings`.

```
aws transcribe start-transcription-job \
```

```
--region us-west-2 \
--transcription-job-name my-first-transcription-job \
--media MediaFileUri=s3://amzn-s3-demo-bucket/my-input-files/my-media-file.flac \
--output-bucket-name amzn-s3-demo-bucket \
--output-key my-output-files/ \
--language-code en-US \
--job-execution-settings
 AllowDeferredExecution=true,DataAccessRoleArn=arn:aws:iam::111122223333:role/
ExampleRole
```

Here's another example using the start-transcription-job command, and a request body that enables queueing.

```
aws transcribe start-transcription-job \
--region us-west-2 \
--cli-input-json file://my-first-queueing-request.json
```

The file *my-first-queueing-request.json* contains the following request body.

```
{
  "TranscriptionJobName": "my-first-transcription-job",
  "Media": {
        "MediaFileUri": "s3://amzn-s3-demo-bucket/my-input-files/my-media-file.flac"
  },
  "OutputBucketName": "amzn-s3-demo-bucket",
  "OutputKey": "my-output-files/",
  "LanguageCode": "en-US",
  "JobExecutionSettings": {
        "AllowDeferredExecution": true,
        "DataAccessRoleArn": "arn:aws:iam::111122223333:role/ExampleRole"
  }
}
```

AWS SDK for Python (Boto3)

This example uses the AWS SDK for Python (Boto3) to enable job queueing using the `AllowDeferredExecution` argument for the start_transcription_job method. Note that when you include `AllowDeferredExecution` in your request, you must also include `DataAccessRoleArn`. For more information, see `StartTranscriptionJob` and `JobExecutionSettings`.

For additional examples using the AWS SDKs, including feature-specific, scenario, and cross-service examples, refer to the Code examples for Amazon Transcribe using AWS SDKs chapter.

```
from __future__ import print_function
import time
import boto3
transcribe = boto3.client('transcribe', 'us-west-2')
job_name = "my-first-queueing-request"
job_uri = "s3://amzn-s3-demo-bucket/my-input-files/my-media-file.flac"
transcribe.start_transcription_job(
    TranscriptionJobName = job_name,
    Media = {
        'MediaFileUri': job_uri
    },
    OutputBucketName = 'amzn-s3-demo-bucket',
    OutputKey = 'my-output-files/',
    LanguageCode = 'en-US',
    JobExecutionSettings = {
        'AllowDeferredExecution': True,
        'DataAccessRoleArn': 'arn:aws:iam::111122223333:role/ExampleRole'
   }
)

while True:
    status = transcribe.get_transcription_job(TranscriptionJobName = job_name)
    if status['TranscriptionJob']['TranscriptionJobStatus'] in ['COMPLETED', 'FAILED']:
        break
    print("Not ready yet...")
    time.sleep(5)
print(status)
```

You can view the progress of a queued job via the AWS Management Console or by submitting a GetTranscriptionJob request. When a job is queued, the `Status` is `QUEUED`. The status changes to `IN_PROGRESS` once your job starts processing, then changes to `COMPLETED` or `FAILED` when processing is finished.

Tagging resources

A tag is a custom metadata label that you can add to a resource to make it easier to identify, organize, and find in a search. Tags are comprised of two individual parts: A tag key and a tag value. This is referred to as a key:value pair.

A tag key typically represents a larger category, while a tag value represents a subset of that category. For example you could have *tag key=Color* and *tag value=Blue*, which would produce the key:value pair `Color:Blue`. Note that you can set the value of a tag to an empty string, but you can't set the value of a tag to null. Omitting the tag value is the same as using an empty string.

> **Tip**
>
> AWS Billing and Cost Management can use tags to separate your bills into dynamic categories. For example, if you add tags to represent different departments within your company, such as `Department:Sales` or `Department:Legal`, AWS can provide you with your cost distribution per department.

In Amazon Transcribe, you can tag the following resources:

- Transcription jobs
- Medical transcription jobs
- Call analytics post call transcription jobs
- Custom vocabularies
- Custom medical vocabularies
- Custom vocabulary filters
- Call analytics categories
- Custom language models

Tag keys can be up to 128 characters in length and tag values can be up to 256 characters in length; both are case sensitive. Amazon Transcribe supports up to 50 tags per resource. For a given resource, each tag key must be unique with only one value. Note that your tags cannot begin with `aws:` because AWS reserves this prefix for system-generated tags. You cannot add, modify, or delete `aws:*` tags, and they don't count against your tags-per-resource limit.

API operations specific to resource tagging

ListTagsForResource, TagResource, UntagResource

To use the tagging APIs, you must include an Amazon Resource Name (ARN) with your request. ARNs have the format `arn:partition:service:region:account-id:resource-type/resource-id`. For example, the ARN associated with a transcription job may look like: `arn:aws:transcribe:us-west-2:111122223333:transcription-job/my-transcription-job-name`.

To learn more about tagging, including best practices, see Tagging AWS resources.

Tag-based access control

You can use tags to control access within your AWS accounts. For tag-based access control, you provide tag information in the condition element of an IAM policy. You can then use tags and their associated tag condition key to control access to:

- **Resources:** Control access to your Amazon Transcribe resources based on the tags you've assigned to those resources.
  - Use the `aws:ResourceTag/key-name` condition key to specify which tag key:value pair must be attached to the resource.
- **Requests:** Control which tags can be passed in a request.
  - Use the `aws:RequestTag/key-name` condition key to specify which tags can be added, modified, or removed from an IAM user or role.
- **Authorization processes:** Control tag-based access for any part of your authorization process.
  - Use the `aws:TagKeys/` condition key to control whether specific tag keys can be used on a resource, in a request, or by a principal. In this case, the key value doesn't matter.

For an example tag-based access control policy, see Viewing transcription jobs based on tags.

For more detailed information on tag-based access control, see Controlling access to AWS resources using tags.

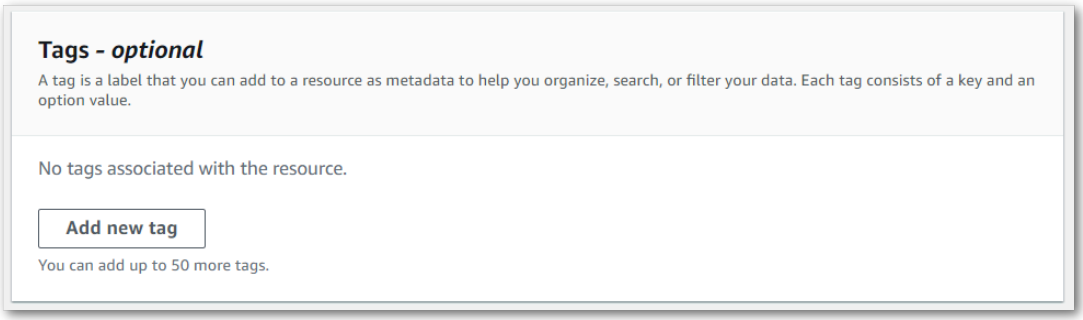
Adding tags to your Amazon Transcribe resources

You can add tags before or after you run your Amazon Transcribe job. Using the existing **Create*** and **Start*** APIs, you can add add tags with your transcription request.

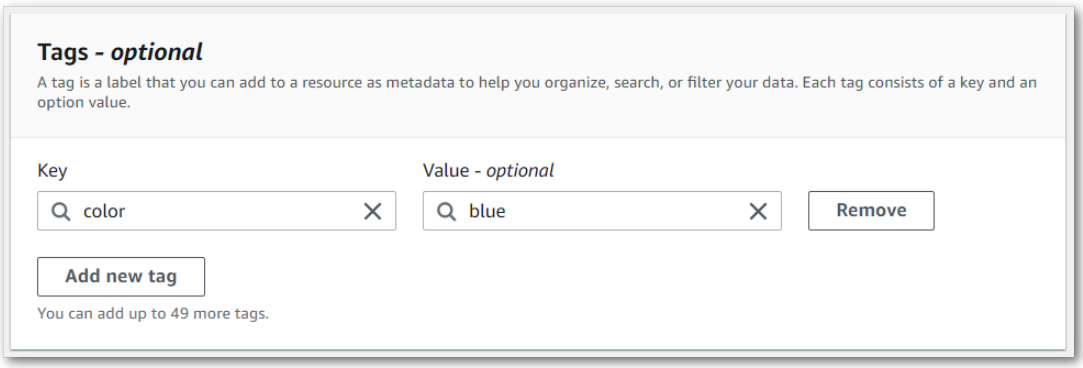
You can add, modify or delete tags using the **AWS Management Console**, **AWS CLI**, or **AWS SDKs**; see the following for examples:

AWS Management Console

1. Sign in to the AWS Management Console.
2. In the navigation pane, choose **Transcription jobs**, then select **Create job** (top right). This opens the **Specify job details** page.
3. Scroll to the bottom of the **Specify job details** page to find the **Tags - *optional*** box and select **Add new tag**.



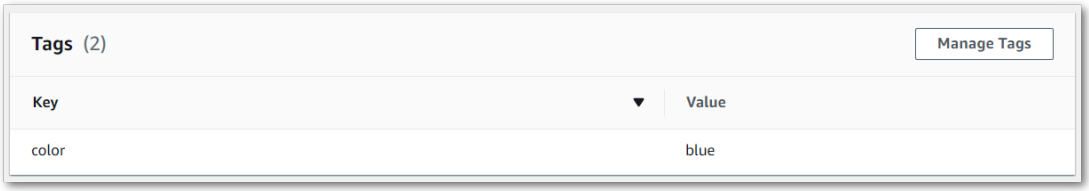
4. Enter information for the **Key** field and, optionally, the **Value** field.



5. Fill in any other fields you want to include on the **Specify job details** page, then select **Next**. This takes you to the **Configure job - *optional* page.**

   Select **Create job** to run your transcription job.

6. You can view the tags associated with a transcription job by navigating to the **Transcription jobs** page, selecting a transcription job, and scrolling to the bottom of that job's information page. If you want to edit your tags, you can do so by selecting **Manage tags**.



AWS CLI

This example uses the start-transcription-job command and Tags parameter. For more information, see StartTranscriptionJob and Tag.

```
aws transcribe start-transcription-job \
--region us-west-2 \
--transcription-job-name my-first-transcription-job \
--media MediaFileUri=s3://amzn-s3-demo-bucket/my-input-files/my-media-file.flac \
--output-bucket-name amzn-s3-demo-bucket \
--output-key my-output-files/ \
--language-code en-US \
--tags Key=color,Value=blue Key=shape,Value=square
```

Here's another example using the start-transcription-job command, and a request body that adds tags to that job.

```
aws transcribe start-transcription-job \
--region us-west-2 \
--cli-input-json file://filepath/my-first-tagging-job.json
```

The file *my-first-tagging-job.json* contains the following request body.

```
{
  "TranscriptionJobName": "my-first-transcription-job",
  "Media": {
        "MediaFileUri": "s3://amzn-s3-demo-bucket/my-input-files/my-media-file.flac"
  },
  "OutputBucketName": "amzn-s3-demo-bucket",
  "OutputKey": "my-output-files/",
  "LanguageCode": "en-US",
```

```
  "Tags": [
        {
           "Key": "color",
           "Value": "blue"
        },
        {
           "Key": "shape",
           "Value": "square"
        }
  ]
}
```

AWS SDK for Python (Boto3)

The following example uses the AWS SDK for Python (Boto3) to add a tag by using the `Tags` argument for the start_transcription_job method. For more information, see `StartTranscriptionJob` and `Tag`.

For additional examples using the AWS SDKs, including feature-specific, scenario, and cross-service examples, refer to the Code examples for Amazon Transcribe using AWS SDKs chapter.

```
from __future__ import print_function
import time
import boto3
transcribe = boto3.client('transcribe', 'us-west-2')
job_name = "my-first-transcription-job"
job_uri = "s3://amzn-s3-demo-bucket/my-input-files/my-media-file.flac"
transcribe.start_transcription_job(
    TranscriptionJobName = job_name,
    Media = {
        'MediaFileUri': job_uri
    },
    OutputBucketName = 'amzn-s3-demo-bucket',
    OutputKey = 'my-output-files/',
    LanguageCode = 'en-US',
    Tags = [
        {
            'Key':'color',
            'Value':'blue'
        }
    ]
)
```

```
while True:
    status = transcribe.get_transcription_job(TranscriptionJobName = job_name)
    if status['TranscriptionJob']['TranscriptionJobStatus'] in ['COMPLETED', 'FAILED']:
        break
    print("Not ready yet...")
    time.sleep(5)
print(status)
```

Partitioning speakers (diarization)

With speaker diarization, you can distinguish between different speakers in your transcription output. Amazon Transcribe can differentiate between a maximum of 30 unique speakers and labels the text from each unique speaker with a unique value (`spk_0` through `spk_29`).

In addition to the standard transcript sections (`transcripts` and `items`), requests with speaker partitioning enabled include a `speaker_labels` section. This section is grouped by speaker and contains information on each utterance, including speaker label and timestamps.

```
"speaker_labels": {
    "channel_label": "ch_0",
    "speakers": 2,
    "segments": [
         {
            "start_time": "4.87",
            "speaker_label": "spk_0",
            "end_time": "6.88",
            "items": [
                {
                    "start_time": "4.87",
                    "speaker_label": "spk_0",
                    "end_time": "5.02"
                },
        ...
        {
            "start_time": "8.49",
            "speaker_label": "spk_1",
            "end_time": "9.24",
            "items": [
                {
                    "start_time": "8.49",
                    "speaker_label": "spk_1",
                    "end_time": "8.88"
                },
```

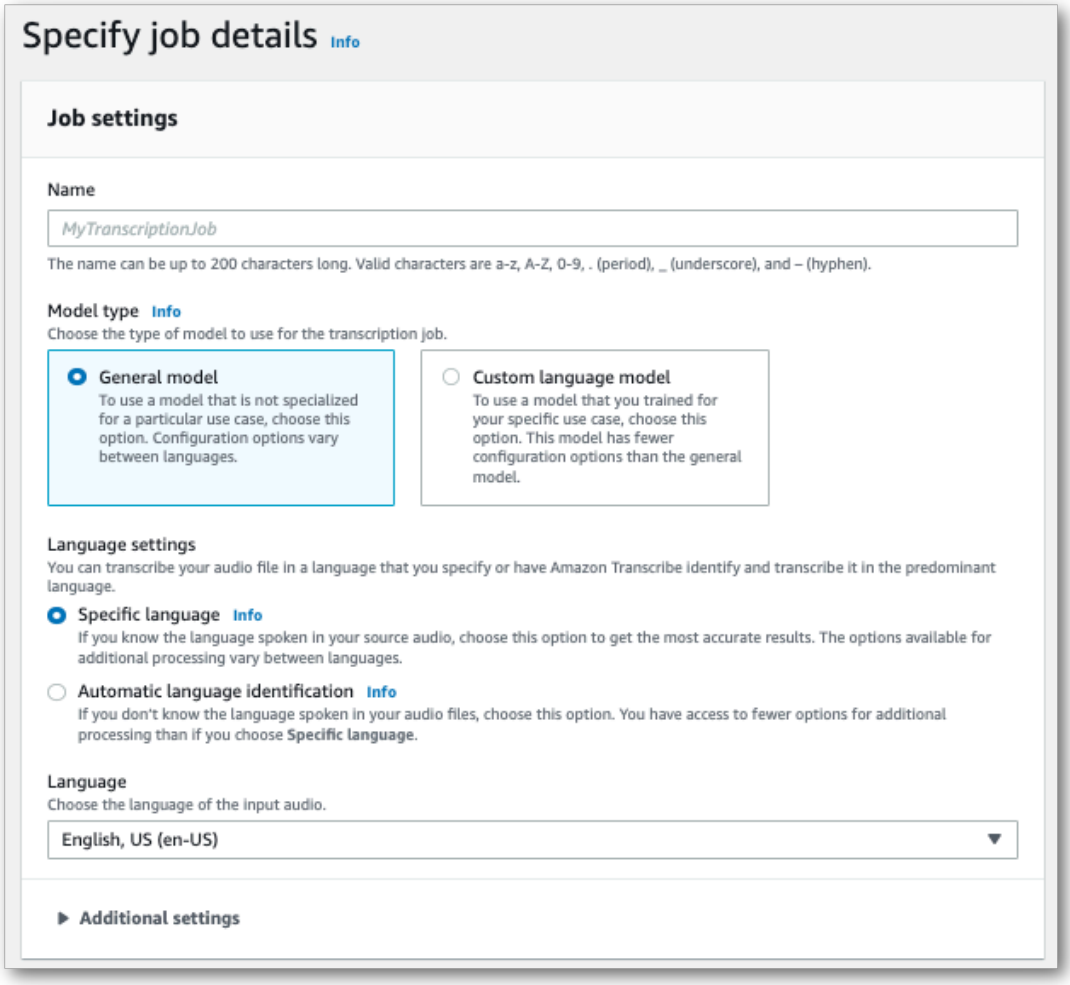
To view a complete example transcript with speaker partitioning (for two speakers), see Example diarization output (batch).

Partitioning speakers in a batch transcription

To partition speakers in a batch transcription, see the following examples:

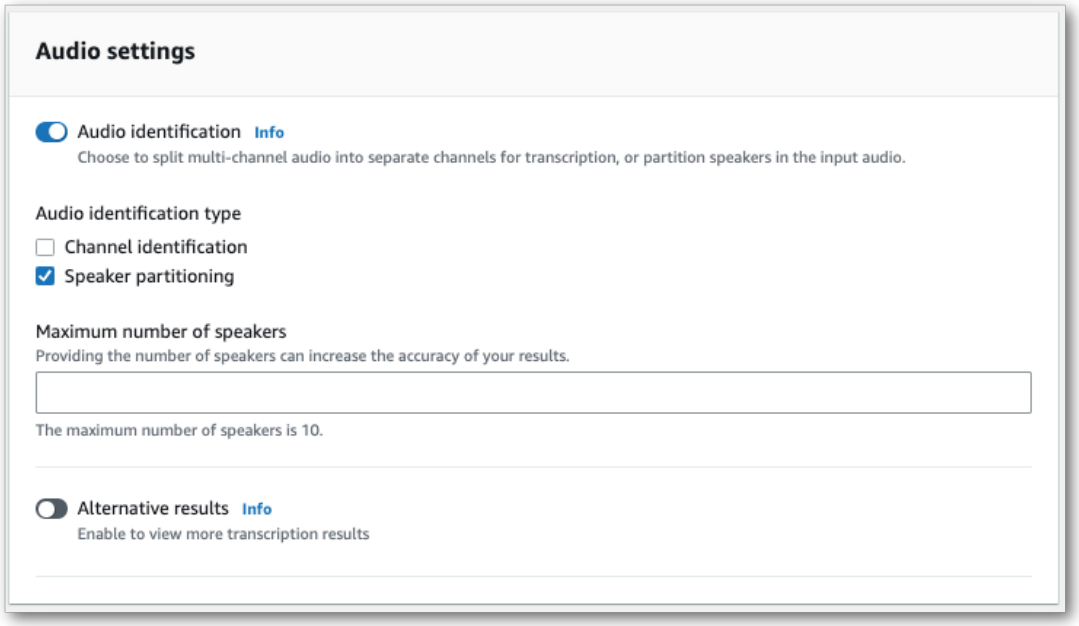
AWS Management Console

1. Sign in to the AWS Management Console.
2. In the navigation pane, choose **Transcription jobs**, then select **Create job** (top right). This opens the **Specify job details** page.



3. Fill in any fields you want to include on the **Specify job details** page, then select **Next**. This takes you to the **Configure job - *optional*** page.

   To enable speaker partitioning, in **Audio settings**, choose **Audio identification**. Then choose **Speaker partitioning** and specify the number of speakers.



4. Select **Create job** to run your transcription job.

AWS CLI

This example uses the start-transcription-job. For more information, see StartTranscriptionJob.

```
aws transcribe start-transcription-job \
--region us-west-2 \
--transcription-job-name my-first-transcription-job \
--media MediaFileUri=s3://amzn-s3-demo-bucket/my-input-files/my-media-file.flac \
--output-bucket-name amzn-s3-demo-bucket \
--output-key my-output-files/ \
--language-code en-US \
--settings ShowSpeakerLabels=true,MaxSpeakerLabels=3
```

Here's another example using the start-transcription-job command, and a request body that enables speaker partitioning with that job.

```
aws transcribe start-transcription-job \
--region us-west-2 \
--cli-input-json file://my-first-transcription-job.json
```

The file *my-first-transcription-job.json* contains the following request body.

```
{
  "TranscriptionJobName": "my-first-transcription-job",
  "Media": {
        "MediaFileUri": "s3://amzn-s3-demo-bucket/my-input-files/my-media-file.flac"
  },
  "OutputBucketName": "amzn-s3-demo-bucket",
  "OutputKey": "my-output-files/",
  "LanguageCode": "en-US",
  "ShowSpeakerLabels": 'TRUE',
  "MaxSpeakerLabels": 3
 }
```

AWS SDK for Python (Boto3)

This example uses the AWS SDK for Python (Boto3) to identify channels using the start_transcription_job method. For more information, see StartTranscriptionJob.

```
from __future__ import print_function
import time
import boto3
transcribe = boto3.client('transcribe', 'us-west-2')
job_name = "my-first-transcription-job"
job_uri = "s3://amzn-s3-demo-bucket/my-input-files/my-media-file.flac"
transcribe.start_transcription_job(
    TranscriptionJobName = job_name,
    Media = {
        'MediaFileUri': job_uri
    },
    OutputBucketName = 'amzn-s3-demo-bucket',
    OutputKey = 'my-output-files/',
    LanguageCode = 'en-US',
    Settings = {
        'ShowSpeakerLabels': True,
        'MaxSpeakerLabels': 3
    }
)

while True:
    status = transcribe.get_transcription_job(TranscriptionJobName = job_name)
    if status['TranscriptionJob']['TranscriptionJobStatus'] in ['COMPLETED', 'FAILED']:
        break
    print("Not ready yet...")
    time.sleep(5)
```

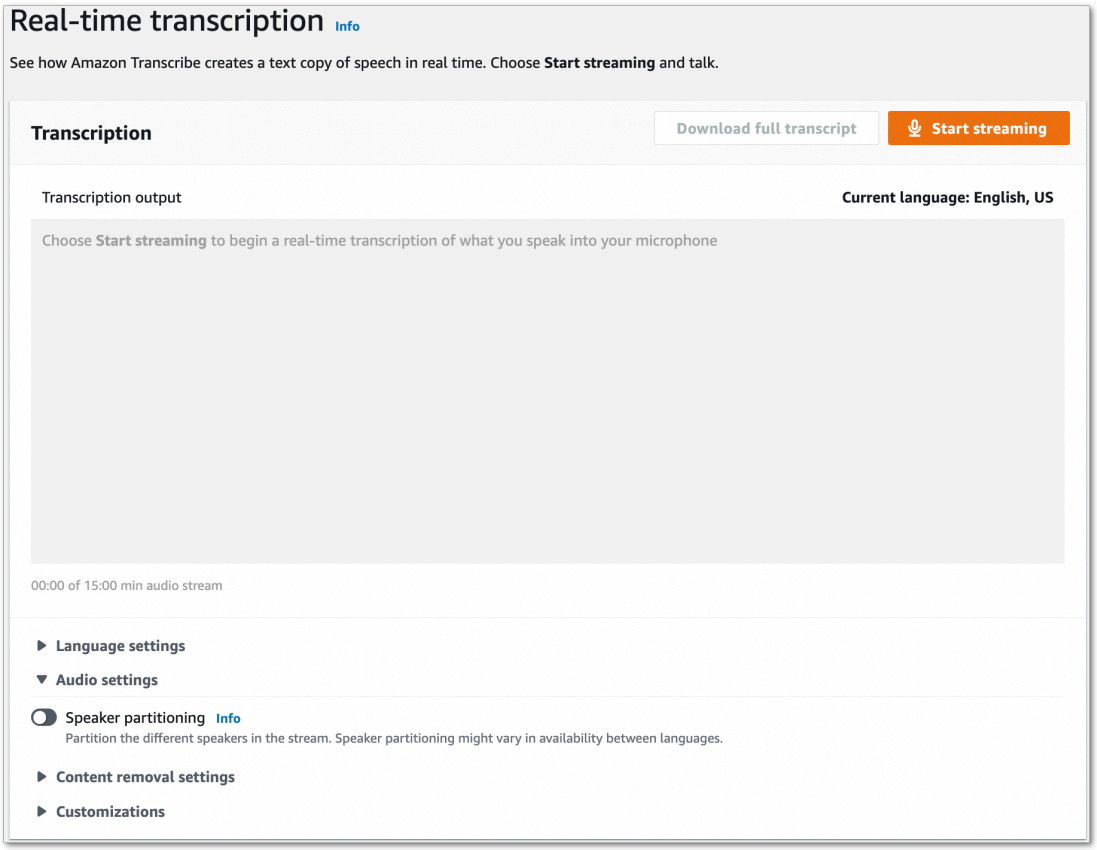
```
print(status)
```

Partitioning speakers in a streaming transcription

To partition speakers in a streaming transcription, see the following examples:

Streaming transcriptions

1. Sign in to the AWS Management Console.
2. In the navigation pane, choose **Real-time transcription**. Scroll down to **Audio settings** and expand this field if it is minimized.



3. Toggle on **Speaker partitioning**.



4. You're now ready to transcribe your stream. Select **Start streaming** and begin speaking. To end your dictation, select **Stop streaming**.

HTTP/2 stream

This example creates an HTTP/2 request that partitions speakers in your transcription output. For more information on using HTTP/2 streaming with Amazon Transcribe, see Setting up an HTTP/2 stream. For more detail on parameters and headers specific to Amazon Transcribe, see StartStreamTranscription.

```
POST /stream-transcription HTTP/2
host: transcribestreaming.us-west-2.amazonaws.com
X-Amz-Target: com.amazonaws.transcribe.Transcribe.StartStreamTranscription
Content-Type: application/vnd.amazon.eventstream
X-Amz-Content-Sha256: string
X-Amz-Date: 20220208T235959Z
Authorization: AWS4-HMAC-SHA256 Credential=access-key/20220208/us-west-2/transcribe/
aws4_request, SignedHeaders=content-type;host;x-amz-content-sha256;x-amz-date;x-amz-
target;x-amz-security-token, Signature=string
x-amzn-transcribe-language-code: en-US
x-amzn-transcribe-media-encoding: flac
x-amzn-transcribe-sample-rate: 16000
x-amzn-transcribe-show-speaker-label: true
transfer-encoding: chunked
```

Parameter definitions can be found in the API Reference; parameters common to all AWS API operations are listed in the Common Parameters section.

WebSocket stream

This example creates a presigned URL that separates speakers in your transcription output. Line breaks have been added for readability. For more information on using WebSocket streams with Amazon Transcribe, see Setting up a WebSocket stream. For more detail on parameters, see StartStreamTranscription.

```
GET wss://transcribestreaming.us-west-2.amazonaws.com:8443/stream-transcription-
websocket?
&X-Amz-Algorithm=AWS4-HMAC-SHA256
&X-Amz-Credential=AKIAIOSFODNN7EXAMPLE%2F20220208%2Fus-
west-2%2Ftranscribe%2Faws4_request
&X-Amz-Date=20220208T235959Z
```

```
&X-Amz-Expires=300
&X-Amz-Security-Token=security-token
&X-Amz-Signature=string
&X-Amz-SignedHeaders=content-type%3Bhost%3Bx-amz-date
&language-code=en-US
&specialty=PRIMARYCARE
&type=DICTATION
&media-encoding=flac
&sample-rate=16000
&show-speaker-label=true
```

Parameter definitions can be found in the API Reference; parameters common to all AWS API operations are listed in the Common Parameters section.

Example diarization output (batch)

Here's an output example for a batch transcription with diarization enabled.

```
{
    "jobName": "my-first-transcription-job",
    "accountId": "111122223333",
    "results": {
        "transcripts": [
            {
                "transcript": "I've been on hold for an hour. Sorry about that."
            }
        ],
        "speaker_labels": {
            "channel_label": "ch_0",
            "speakers": 2,
            "segments": [
                {
                    "start_time": "4.87",
                    "speaker_label": "spk_0",
                    "end_time": "6.88",
                    "items": [
                        {
                            "start_time": "4.87",
                            "speaker_label": "spk_0",
                            "end_time": "5.02"
                        },
                        {
                            "start_time": "5.02",
```

```
                        "speaker_label": "spk_0",
                        "end_time": "5.17"
                    },
                    {
                        "start_time": "5.17",
                        "speaker_label": "spk_0",
                        "end_time": "5.29"
                    },
                    {
                        "start_time": "5.29",
                        "speaker_label": "spk_0",
                        "end_time": "5.64"
                    },
                    {
                        "start_time": "5.64",
                        "speaker_label": "spk_0",
                        "end_time": "5.84"
                    },
                    {
                        "start_time": "6.11",
                        "speaker_label": "spk_0",
                        "end_time": "6.26"
                    },
                    {
                        "start_time": "6.26",
                        "speaker_label": "spk_0",
                        "end_time": "6.88"
                    }
                ]
            },
            {
                "start_time": "8.49",
                "speaker_label": "spk_1",
                "end_time": "9.24",
                "items": [
                    {
                        "start_time": "8.49",
                        "speaker_label": "spk_1",
                        "end_time": "8.88"
                    },
                    {
                        "start_time": "8.88",
                        "speaker_label": "spk_1",
                        "end_time": "9.05"
```

```
                    },
                    {
                        "start_time": "9.05",
                        "speaker_label": "spk_1",
                        "end_time": "9.24"
                    }
                ]
            }
        ]
    },
    "items": [
        {
            "id": 0,
            "start_time": "4.87",
            "speaker_label": "spk_0",
            "end_time": "5.02",
            "alternatives": [
                {
                    "confidence": "1.0",
                    "content": "I've"
                }
            ],
            "type": "pronunciation"
        },
        {
            "id": 1,
            "start_time": "5.02",
            "speaker_label": "spk_0",
            "end_time": "5.17",
            "alternatives": [
                {
                    "confidence": "1.0",
                    "content": "been"
                }
            ],
            "type": "pronunciation"
        },
        {
            "id": 2,
            "start_time": "5.17",
            "speaker_label": "spk_0",
            "end_time": "5.29",
            "alternatives": [
                {
```

```
                    "confidence": "1.0",
                    "content": "on"
                }
            ],
            "type": "pronunciation"
        },
        {
            "id": 3,
            "start_time": "5.29",
            "speaker_label": "spk_0",
            "end_time": "5.64",
            "alternatives": [
                {
                    "confidence": "1.0",
                    "content": "hold"
                }
            ],
            "type": "pronunciation"
        },
        {
            "id": 4,
            "start_time": "5.64",
            "speaker_label": "spk_0",
            "end_time": "5.84",
            "alternatives": [
                {
                    "confidence": "1.0",
                    "content": "for"
                }
            ],
            "type": "pronunciation"
        },
        {
            "id": 5,
            "start_time": "6.11",
            "speaker_label": "spk_0",
            "end_time": "6.26",
            "alternatives": [
                {
                    "confidence": "1.0",
                    "content": "an"
                }
            ],
            "type": "pronunciation"
```

```
            },
            {
                "id": 6,
                "start_time": "6.26",
                "speaker_label": "spk_0",
                "end_time": "6.88",
                "alternatives": [
                    {
                        "confidence": "1.0",
                        "content": "hour"
                    }
                ],
                "type": "pronunciation"
            },
            {
                "id": 7,
                "speaker_label": "spk_0",
                "alternatives": [
                    {
                        "confidence": "0.0",
                        "content": "."
                    }
                ],
                "type": "punctuation"
            },
            {
                "id": 8,
                "start_time": "8.49",
                "speaker_label": "spk_1",
                "end_time": "8.88",
                "alternatives": [
                    {
                        "confidence": "1.0",
                        "content": "Sorry"
                    }
                ],
                "type": "pronunciation"
            },
            {
                "id": 9,
                "start_time": "8.88",
                "speaker_label": "spk_1",
                "end_time": "9.05",
                "alternatives": [
```

```
                    {
                        "confidence": "0.902",
                        "content": "about"
                    }
                ],
                "type": "pronunciation"
            },
            {
                "id": 10,
                "start_time": "9.05",
                "speaker_label": "spk_1",
                "end_time": "9.24",
                "alternatives": [
                    {
                        "confidence": "1.0",
                        "content": "that"
                    }
                ],
                "type": "pronunciation"
            },
            {
                "id": 11,
                "speaker_label": "spk_1",
                "alternatives": [
                    {
                        "confidence": "0.0",
                        "content": "."
                    }
                ],
                "type": "punctuation"
            }
        ],
        "audio_segments": [
            {
                "id": 0,
                "transcript": "I've been on hold for an hour.",
                "start_time": "4.87",
                "end_time": "6.88",
                "speaker_label": "spk_0",
                "items": [
                    0,
                    1,
                    2,
                    3,
```

```
                    4,
                    5,
                    6,
                    7
                ]
            },
            {
                "id": 1,
                "transcript": "Sorry about that.",
                "start_time": "8.49",
                "end_time": "9.24",
                "speaker_label": "spk_1",
                "items": [
                    8,
                    9,
                    10,
                    11
                ]
            }
        ]
    },
    "status": "COMPLETED"
}
```

Transcribing multi-channel audio

If your audio has two channels, you can use channel identification to transcribe the speech from each channel separately. Amazon Transcribe doesn't currently support audio with more than two channels.

In your transcript, channels are assigned the labels `ch_0` and `ch_1`.

In addition to the standard transcript sections (`transcripts` and `items`), requests with channel identification enabled include a `channel_labels` section. This section contains each utterance or punctuation mark, grouped by channel, and its associated channel label, timestamps, and confidence score.

```
"channel_labels": {
    "channels": [
        {
            "channel_label": "ch_0",
            "items": [
                {
                    "channel_label": "ch_0",
                    "start_time": "4.86",
                    "end_time": "5.01",
                    "alternatives": [
                        {
                            "confidence": "1.0",
                            "content": "I've"
                        }
                    ],
                    "type": "pronunciation"
                },
                ...
            "channel_label": "ch_1",
            "items": [
                {
                    "channel_label": "ch_1",
                    "start_time": "8.5",
                    "end_time": "8.89",
                    "alternatives": [
                        {
                            "confidence": "1.0",
                            "content": "Sorry"
```

```
                    }
                ],
                "type": "pronunciation"
            },
            ...
        "number_of_channels": 2
    },
```

Note that if a person on one channel speaks at the same time as a person on a separate channel, timestamps for each channel overlap while the individuals are speaking over each other.

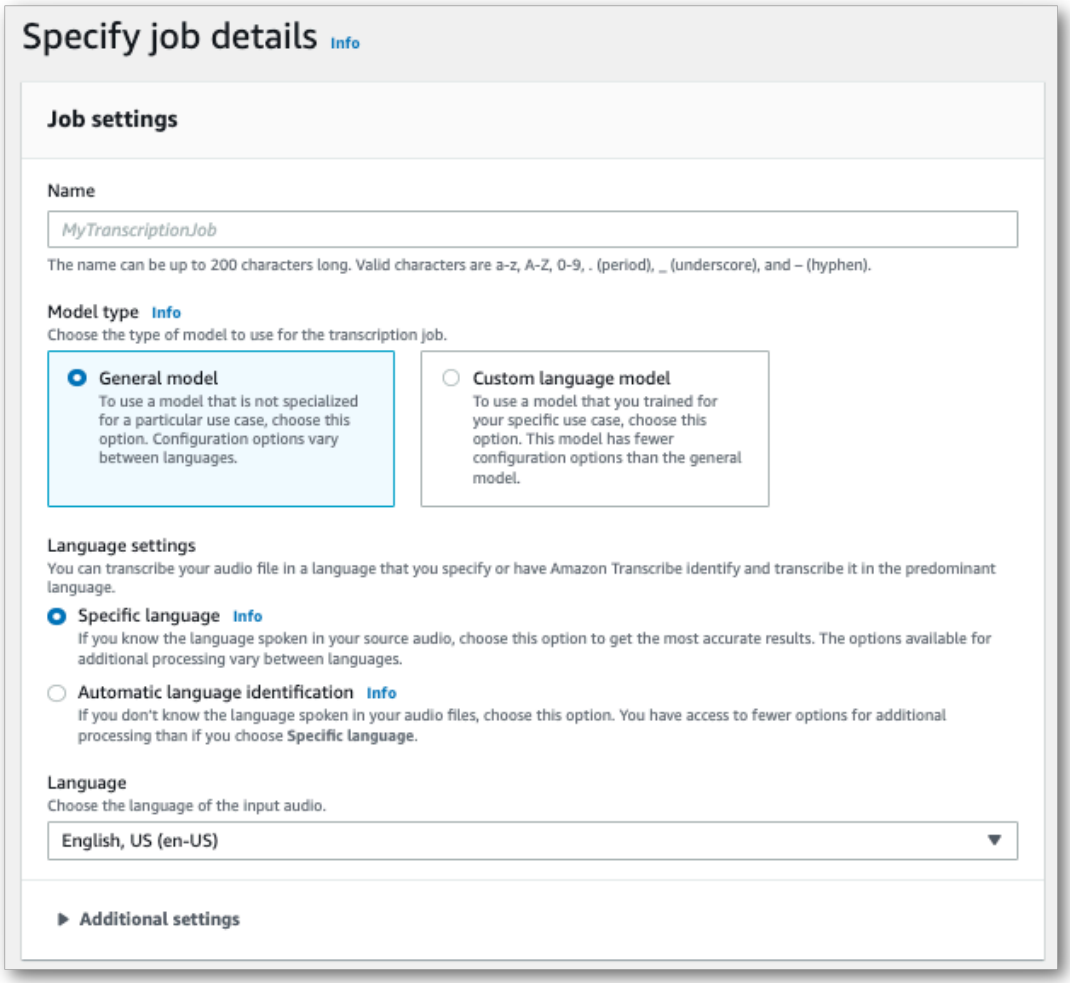
To view a complete example transcript with channel identification, see Example channel identification output (batch).

Using channel identification in a batch transcription

To identify channels in a batch transcription, you can use the **AWS Management Console**, **AWS CLI**, or **AWS SDKs**; see the following for examples:

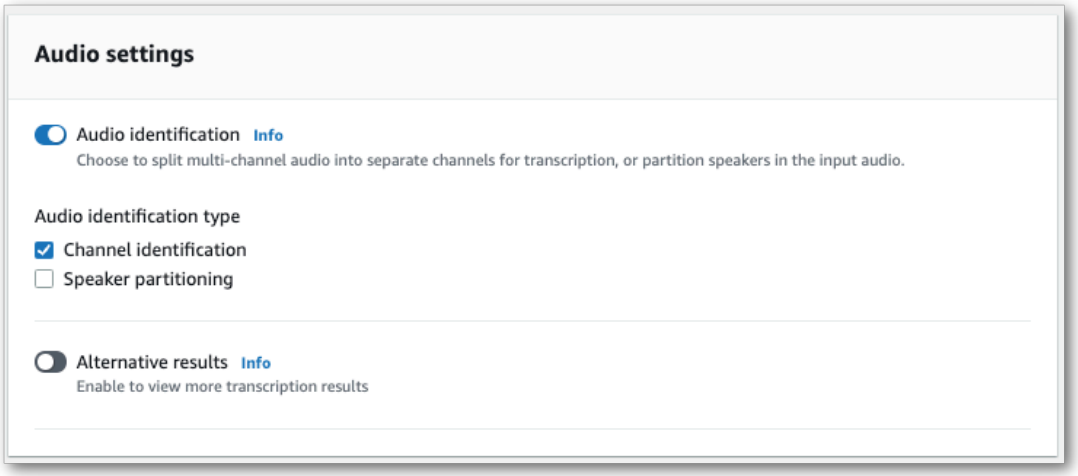
AWS Management Console

1. Sign in to the AWS Management Console.
2. In the navigation pane, choose **Transcription jobs**, then select **Create job** (top right). This opens the **Specify job details** page.



3. Fill in any fields you want to include on the **Specify job details** page, then select **Next**. This takes you to the **Configure job - *optional*** page.

   In the **Audio settings** panel, select **Channel identification** (under the 'Audio identification type' heading).



4. Select **Create job** to run your transcription job.

AWS CLI

This example uses the start-transcription-job. For more information, see StartTranscriptionJob.

```
aws transcribe start-transcription-job \
--region us-west-2 \
--transcription-job-name my-first-transcription-job \
--media MediaFileUri=s3://amzn-s3-demo-bucket/my-input-files/my-media-file.flac \
--output-bucket-name amzn-s3-demo-bucket \
--output-key my-output-files/ \
--language-code en-US \
--settings ChannelIdentification=true
```

Here's another example using the start-transcription-job command, and a request body that enables channel identification with that job.

```
aws transcribe start-transcription-job \
--region us-west-2 \
--cli-input-json file://my-first-transcription-job.json
```

The file *my-first-transcription-job.json* contains the following request body.

```
{
    "TranscriptionJobName": "my-first-transcription-job",
```

```
    "Media": {
        "MediaFileUri": "s3://amzn-s3-demo-bucket/my-input-files/my-media-file.flac"
  },
    "OutputBucketName": "amzn-s3-demo-bucket",
    "OutputKey": "my-output-files/",
    "LanguageCode": "en-US",
    "Settings": {
        "ChannelIdentification": true
    }
}
```

AWS SDK for Python (Boto3)

This example uses the AWS SDK for Python (Boto3) to identify channels using the start_transcription_job method. For more information, see StartTranscriptionJob.

```
from __future__ import print_function
import time
import boto3
transcribe = boto3.client('transcribe', 'us-west-2')
job_name = "my-first-transcription-job"
job_uri = "s3://amzn-s3-demo-bucket/my-input-files/my-media-file.flac"
transcribe.start_transcription_job(
    TranscriptionJobName = job_name,
    Media = {
        'MediaFileUri': job_uri
    },
    OutputBucketName = 'amzn-s3-demo-bucket',
    OutputKey = 'my-output-files/',
    LanguageCode = 'en-US',
    Settings = {
        'ChannelIdentification':True
    }
)

while True:
    status = transcribe.get_transcription_job(TranscriptionJobName = job_name)
    if status['TranscriptionJob']['TranscriptionJobStatus'] in ['COMPLETED', 'FAILED']:
        break
    print("Not ready yet...")
    time.sleep(5)
print(status)
```

Using channel identification in a streaming transcription

To identify channels in a streaming transcription, you can use **HTTP/2** or **WebSockets**; see the following for examples:

HTTP/2 stream

This example creates an HTTP/2 request that separates channels in your transcription output. For more information on using HTTP/2 streaming with Amazon Transcribe, see Setting up an HTTP/2 stream. For more detail on parameters and headers specific to Amazon Transcribe, see StartStreamTranscription.

```
POST /stream-transcription HTTP/2
host: transcribestreaming.us-west-2.amazonaws.com
X-Amz-Target: com.amazonaws.transcribe.Transcribe.StartStreamTranscription
Content-Type: application/vnd.amazon.eventstream
X-Amz-Content-Sha256: string
X-Amz-Date: 20220208T235959Z
Authorization: AWS4-HMAC-SHA256 Credential=access-key/20220208/us-west-2/transcribe/
aws4_request, SignedHeaders=content-type;host;x-amz-content-sha256;x-amz-date;x-amz-
target;x-amz-security-token, Signature=string
x-amzn-transcribe-language-code: en-US
x-amzn-transcribe-media-encoding: flac
x-amzn-transcribe-sample-rate: 16000
x-amzn-channel-identification: TRUE
transfer-encoding: chunked
```

Parameter definitions can be found in the API Reference; parameters common to all AWS API operations are listed in the Common Parameters section.

WebSocket stream

This example creates a presigned URL that separates channels in your transcription output. Line breaks have been added for readability. For more information on using WebSocket streams with Amazon Transcribe, see Setting up a WebSocket stream. For more detail on parameters, see StartStreamTranscription.

```
GET wss://transcribestreaming.us-west-2.amazonaws.com:8443/stream-transcription-
websocket?
&X-Amz-Algorithm=AWS4-HMAC-SHA256
```

```
&X-Amz-Credential=AKIAIOSFODNN7EXAMPLE%2F20220208%2Fus-
west-2%2Ftranscribe%2Faws4_request
&X-Amz-Date=20220208T235959Z
&X-Amz-Expires=300
&X-Amz-Security-Token=security-token
&X-Amz-Signature=string
&X-Amz-SignedHeaders=content-type%3Bhost%3Bx-amz-date
&language-code=en-US
&specialty=PRIMARYCARE
&type=DICTATION
&media-encoding=flac
&sample-rate=16000
&channel-identification=TRUE
```

Parameter definitions can be found in the API Reference; parameters common to all AWS API operations are listed in the Common Parameters section.

Example channel identification output (batch)

Here's an output example for a batch transcription with channel identification enabled.

```
{
    "jobName": "my-first-transcription-job",
    "accountId": "111122223333",
    "results": {
        "transcripts": [
            {
                "transcript": "I've been on hold for an hour. Sorry about that."
            }
        ],
        "channel_labels": {
            "channels": [
                {
                    "channel_label": "ch_0",
                    "items": [
                        {
                            "channel_label": "ch_0",
                            "start_time": "4.86",
                            "end_time": "5.01",
                            "alternatives": [
                                {
                                    "confidence": "1.0",
                                    "content": "I've"
```

```
                        }
                    ],
                    "type": "pronunciation"
                },
                {
                    "channel_label": "ch_0",
                    "start_time": "5.01",
                    "end_time": "5.16",
                    "alternatives": [
                        {
                            "confidence": "1.0",
                            "content": "been"
                        }
                    ],
                    "type": "pronunciation"
                },
                {
                    "channel_label": "ch_0",
                    "start_time": "5.16",
                    "end_time": "5.28",
                    "alternatives": [
                        {
                            "confidence": "1.0",
                            "content": "on"
                        }
                    ],
                    "type": "pronunciation"
                },
                {
                    "channel_label": "ch_0",
                    "start_time": "5.28",
                    "end_time": "5.62",
                    "alternatives": [
                        {
                            "confidence": "1.0",
                            "content": "hold"
                        }
                    ],
                    "type": "pronunciation"
                },
                {
                    "channel_label": "ch_0",
                    "start_time": "5.62",
                    "end_time": "5.83",
```

```
                        "alternatives": [
                            {
                                "confidence": "1.0",
                                "content": "for"
                            }
                        ],
                        "type": "pronunciation"
                    },
                    {
                        "channel_label": "ch_0",
                        "start_time": "6.1",
                        "end_time": "6.25",
                        "alternatives": [
                            {
                                "confidence": "1.0",
                                "content": "an"
                            }
                        ],
                        "type": "pronunciation"
                    },
                    {
                        "channel_label": "ch_0",
                        "start_time": "6.25",
                        "end_time": "6.87",
                        "alternatives": [
                            {
                                "confidence": "1.0",
                                "content": "hour"
                            }
                        ],
                        "type": "pronunciation"
                    },
                    {
                        "channel_label": "ch_0",
                        "language_code": "en-US",
                        "alternatives": [
                            {
                                "confidence": "0.0",
                                "content": "."
                            }
                        ],
                        "type": "punctuation"
                    }
                ]
```

```
                },
                {
                "channel_label": "ch_1",
                    "items": [
                        {
                            "channel_label": "ch_1",
                            "start_time": "8.5",
                            "end_time": "8.89",
                            "alternatives": [
                                {
                                    "confidence": "1.0",
                                    "content": "Sorry"
                                }
                            ],
                            "type": "pronunciation"
                        },
                        {
                            "channel_label": "ch_1",
                            "start_time": "8.89",
                            "end_time": "9.06",
                            "alternatives": [
                                {
                                    "confidence": "0.9176",
                                    "content": "about"
                                }
                            ],
                            "type": "pronunciation"
                        },
                        {
                            "channel_label": "ch_1",
                            "start_time": "9.06",
                            "end_time": "9.25",
                            "alternatives": [
                                {
                                    "confidence": "1.0",
                                    "content": "that"
                                }
                            ],
                            "type": "pronunciation"
                        },
                        {
                            "channel_label": "ch_1",
                            "alternatives": [
                                {
```

```
                                "confidence": "0.0",
                                "content": "."
                            }
                        ],
                        "type": "punctuation"
                    }
                ]
            }
        ],
        "number_of_channels": 2
    },
    "items": [
        {
            "id": 0,
            "channel_label": "ch_0",
            "start_time": "4.86",
            "end_time": "5.01",
            "alternatives": [
                {
                    "confidence": "1.0",
                    "content": "I've"
                }
            ],
            "type": "pronunciation"
        },
        {
            "id": 1,
            "channel_label": "ch_0",
            "start_time": "5.01",
            "end_time": "5.16",
            "alternatives": [
                {
                    "confidence": "1.0",
                    "content": "been"
                }
            ],
            "type": "pronunciation"
        },
        {
            "id": 2,
            "channel_label": "ch_0",
            "start_time": "5.16",
            "end_time": "5.28",
            "alternatives": [
```

```
                {
                    "confidence": "1.0",
                    "content": "on"
                }
            ],
            "type": "pronunciation"
        },
        {
            "id": 3,
            "channel_label": "ch_0",
            "start_time": "5.28",
            "end_time": "5.62",
            "alternatives": [
                {
                    "confidence": "1.0",
                    "content": "hold"
                }
            ],
            "type": "pronunciation"
        },
        {
            "id": 4,
            "channel_label": "ch_0",
            "start_time": "5.62",
            "end_time": "5.83",
            "alternatives": [
                {
                    "confidence": "1.0",
                    "content": "for"
                }
            ],
            "type": "pronunciation"
        },
        {
            "id": 5,
            "channel_label": "ch_0",
            "start_time": "6.1",
            "end_time": "6.25",
            "alternatives": [
                {
                    "confidence": "1.0",
                    "content": "an"
                }
            ],
```

```
            "type": "pronunciation"
        },
        {
            "id": 6,
            "channel_label": "ch_0",
            "start_time": "6.25",
            "end_time": "6.87",
            "alternatives": [
                {
                    "confidence": "1.0",
                    "content": "hour"
                }
            ],
            "type": "pronunciation"
        },
        {
            "id": 7,
            "channel_label": "ch_0",
            "alternatives": [
                {
                    "confidence": "0.0",
                    "content": "."
                }
            ],
            "type": "punctuation"
        },
        {
            "id": 8,
            "channel_label": "ch_1",
            "start_time": "8.5",
            "end_time": "8.89",
            "alternatives": [
                {
                    "confidence": "1.0",
                    "content": "Sorry"
                }
            ],
            "type": "pronunciation"
        },
        {
            "id": 9,
            "channel_label": "ch_1",
            "start_time": "8.89",
            "end_time": "9.06",
```

```
                "alternatives": [
                    {
                        "confidence": "0.9176",
                        "content": "about"
                    }
                ],
                "type": "pronunciation"
            },
            {
                "id": 10,
                "channel_label": "ch_1",
                "start_time": "9.06",
                "end_time": "9.25",
                "alternatives": [
                    {
                        "confidence": "1.0",
                        "content": "that"
                    }
                ],
                "type": "pronunciation"
            },
            {
                "id": 11,
                "channel_label": "ch_1",
                "alternatives": [
                    {
                        "confidence": "0.0",
                        "content": "."
                    }
                ],
                "type": "punctuation"
            }
        ],
        "audio_segments": [
            {
                "id": 0,
                "transcript": "I've been on hold for an hour.",
                "start_time": "4.86",
                "end_time": "6.87",
                "channel_label": "ch_0",
                "items": [
                    0,
                    1,
                    2,
```

```
                    3,
                    4,
                    5,
                    6,
                    7
                ]
            },
            {
                "id": 1,
                "transcript": "Sorry about that.",
                "start_time": "8.5",
                "end_time": "9.25",
                "channel_label": "ch_1",
                "items": [
                    8,
                    9,
                    10,
                    11
                ]
            }
        ]
    },
    "status": "COMPLETED"
}
```

Identifying the dominant languages in your media

Amazon Transcribe is able to automatically identify the languages spoken in your media without you having to specify a language code.

Batch language identification can identify the dominant language spoken in your media file or, if your media contains multiple languages, it can identify all languages spoken. To improve language identification accuracy, you can optionally provide a list of two or more languages you think may be present in your media.

Streaming language identification can identify one language per channel (a maximum of two channels are supported) or, if your stream contains multiple languages, it can identify all languages spoken. Streaming requests must have a minimum of two additional language options included in your request. Providing language options allows for faster language identification. The faster Amazon Transcribe is able to identify the language, the less change there is of data loss in the first few seconds of your stream.

> **Important**
>
> Batch and streaming transcriptions support different languages. Refer to the **Data input** column in the supported languages table for details. Note that Swedish and Vietnamese are not currently supported with language identification.

To learn about monitoring and events with language identification, refer to Language identification events.

Language identification with batch transcription jobs

Use batch language identification to automatically identify the language, or languages, in your media file.

If your media contains only one language, you can enable single-language identification, which identifies the dominant language spoken in your media file and creates your transcript using only this language.

If your media contains more than one language, you can enable multi-language identification, which identifies all languages spoken in your media file and creates your transcript using each

identified language. Note that a multi-lingual transcript is produced. You can use other services, such as Amazon Translate, to translate your transcript.

Refer to the supported languages table for a complete list of supported languages and associated language codes.

For best results, ensure that your media file contains at least 30 seconds of speech.

For usage examples with the AWS Management Console, AWS CLI, and AWS Python SDK, see Using language identification with batch transcriptions.

Identifying languages in multi-language audio

Multi-language identification is intended for multi-lingual media files, and provides you with a transcript that reflects all supported languages spoken in your media. This means that if speakers change languages mid-conversation, or if each participant is speaking a different language, your transcription output detects and transcribes each language correctly. For example, if your media contains a bilingual speaker who is alternating between US English (`en-US`) and Hindi (`hi-IN`), multi-language identification can identify and transcribe spoken US English as `en-US` and spoken Hindi as `hi-IN`.

This differs from single-language identification, where only one dominant language is used to create a transcript. In this case, any spoken language that is not the dominant language is transcribed incorrectly.

Note

Redaction and custom language models are not currently supported with multi-language identification.

Note

The following languages are currently supported with multi-language identification: en-AB, en-AU, en-GB, en-IE, en-IN, en-NZ, en-US, en-WL, en-ZA, es-ES, es-US, fr-CA, fr-FR, zh-CN, zh-TW, pt-BR, pt-PT, de-CH, de-DE, af-ZA, ar-AE, da-DK, he-IL, hi-IN, id-ID, fa-IR, it-IT, ja-JP, ko-KR, ms-MY, nl-NL, ru-RU, ta-IN, te-IN, th-TH, tr-TR

Multi-language transcripts provide a summary of detected languages and the total time that each language is spoken in your media. Here's an example:

```
"results": {
        "transcripts": [
            {
                "transcript": "welcome to Amazon transcribe. ## ## ###### ### #### ####
 ## #### ### ###################"
            }
        ],

    ...

        "language_codes": [
            {
                "language_code": "en-US",
                "duration_in_seconds": 2.45
            },
            {
                "language_code": "hi-IN",
                "duration_in_seconds": 5.325
            },
            {
                "language_code": "ja-JP",
                "duration_in_seconds": 4.15
            }
        ]
}
```

Improving language identification accuracy

With language identification, you have the option to include a list of languages you think may be present in your media. Including language options (`LanguageOptions`) restricts Amazon Transcribe to using only the languages that you specify when matching your audio to the correct language, which can speed up language identification and improve the accuracy associated with assigning the correct language dialect.

If you choose to include language codes, you must include at least two. There's no limit on the number of language codes you can include, but we recommend using between two and five for optimal efficiency and accuracy.

> **Note**
>
> If you include language codes with your request and none of the language codes you provide match the language, or languages, identified in your audio, Amazon Transcribe selects the closest language match from your specified language codes. It then produces a transcript in that language. For example, if your media is in US English (en-US) and you provide Amazon Transcribe with the language codes zh-CN, fr-FR, and de-DE, Amazon Transcribe is likely to match your media to German (de-DE) and produce a German-language transcription. Mismatching language codes and spoken languages can result in an inaccurate transcript, so we recommend caution when including language codes.

Combining language identification with other Amazon Transcribe features

You can use batch language identification in combination with any other Amazon Transcribe feature. If combining language identification with other features, you are limited to the languages supported with those features. For example, if using language identification with content redaction, you are limited to US English (en-US) or US Spanish (es-US), as this is only language available for redaction. Refer to Supported languages and language-specific features for more information.

> **Important**
>
> If you're using automatic language identification with content redaction enabled and your audio contains languages other than US English (en-US) or US Spanish (es-US), only the US English or US Spanish content is redacted in your transcript. Other languages cannot be redacted and there are no warnings or job failures.

Custom language models, custom vocabularies, and custom vocabulary filters

If you want to add one or more custom language models, custom vocabularies, or custom vocabulary filters to your language identification request, you must include the `LanguageIdSettings` parameter. You can then specify a language code with a corresponding custom language model, custom vocabulary, and custom vocabulary filter. Note that multi-language identification doesn't support custom language models.

It's recommended that you include `LanguageOptions` when using `LanguageIdSettings` to ensure that the correct language dialect is identified. For example, if you specify an `en-US` custom vocabulary, but Amazon Transcribe determines that the language spoken in your media is `en-AU`, your custom vocabulary *is not* applied to your transcription. If you include `LanguageOptions` and specify `en-US` as the only English language dialect, your custom vocabulary *is* applied to your transcription.

For examples of `LanguageIdSettings` in a request, refer to Option 2 in the **AWS CLI** and **AWS SDKs** dropdown panels in the Using language identification with batch transcriptions section.

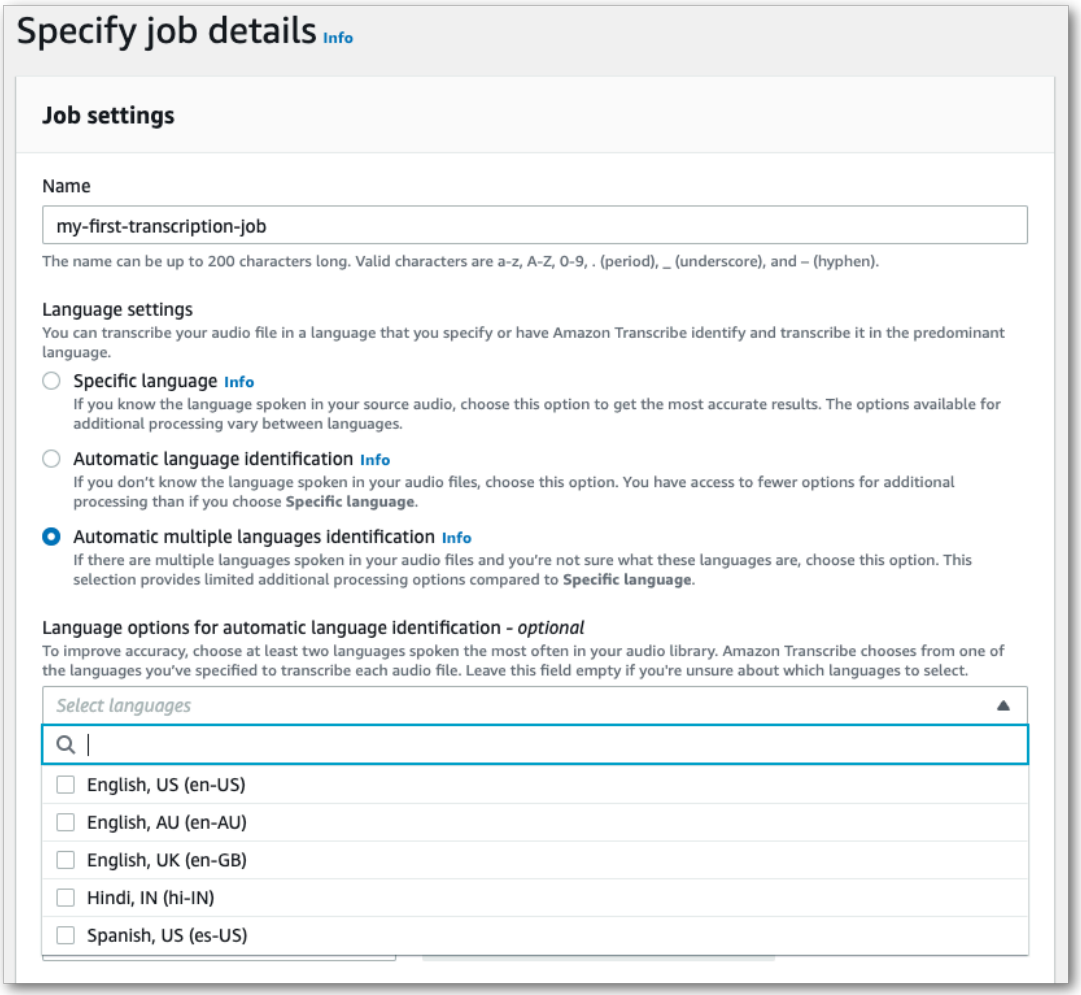
Using language identification with batch transcriptions

You can use automatic language identification in a batch transcription job using the **AWS Management Console**, **AWS CLI**, or **AWS SDKs**; see the following for examples:

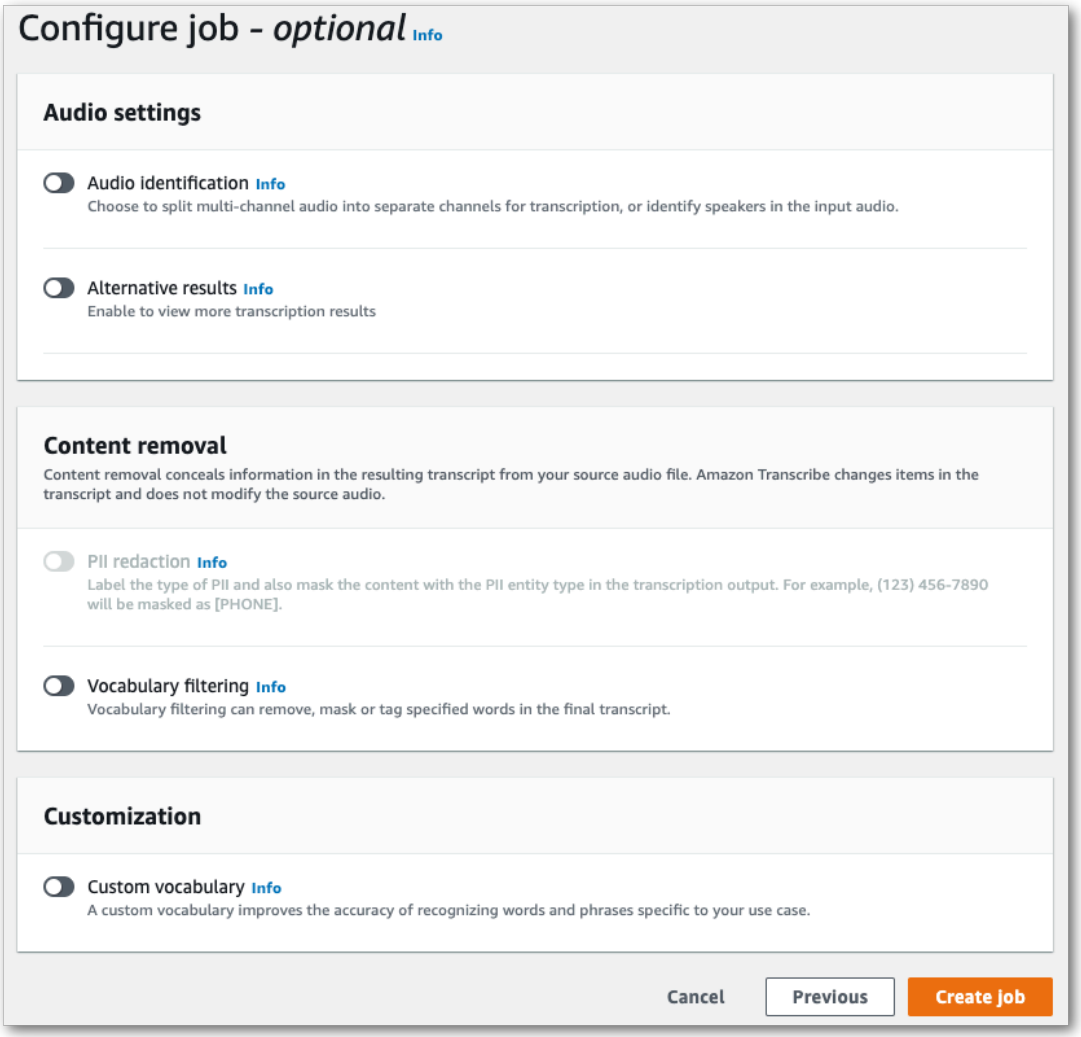
AWS Management Console

1. Sign in to the AWS Management Console.
2. In the navigation pane, choose **Transcription jobs**, then select **Create job** (top right). This opens the **Specify job details** page.
3. In the **Job settings** panel, find the **Language settings** section and select **Automatic language identification** or **Automatic multiple languages identification**.

   You have the option to select multiple language options (from the *Select languages* dropdown box) if you know which languages are present in your audio file. Providing language options can improve accuracy, but is not required.



4. Fill in any other fields you want to include on the **Specify job details** page, then select **Next**. This takes you to the **Configure job - *optional*** page.



5. Select **Create job** to run your transcription job.

AWS CLI

This example uses the start-transcription-job command and `IdentifyLanguage` parameter. For more information, see `StartTranscriptionJob` and `LanguageIdSettings`.

Option 1: Without the `language-id-settings` parameter. Use this option if you are **not** including a custom language model, custom vocabulary, or custom vocabulary filter in your request. `language-options` is optional, but recommended.

```
aws transcribe start-transcription-job \
--region us-west-2 \
--transcription-job-name my-first-transcription-job \
--media MediaFileUri=s3://amzn-s3-demo-bucket/my-input-files/my-media-file.flac \
```

```
--output-bucket-name amzn-s3-demo-bucket \
--output-key my-output-files/ \
--identify-language \  (or --identify-multiple-languages) \
--language-options "en-US" "hi-IN"
```

Option 2: With the `language-id-settings` parameter. Use this option if you **are** including a custom language model, custom vocabulary, or custom vocabulary filter in your request.

```
aws transcribe start-transcription-job \
--region us-west-2 \
--transcription-job-name my-first-transcription-job \
--media MediaFileUri=s3://amzn-s3-demo-bucket/my-input-files/my-media-file.flac \
--output-bucket-name amzn-s3-demo-bucket \
--output-key my-output-files/ \
--identify-language \  (or --identify-multiple-languages)
--language-options "en-US" "hi-IN" \
--language-id-settings en-US=VocabularyName=my-en-US-vocabulary,en-
US=VocabularyFilterName=my-en-US-vocabulary-filter,en-US=LanguageModelName=my-en-US-
language-model,hi-IN=VocabularyName=my-hi-IN-vocabulary,hi-IN=VocabularyFilterName=my-
hi-IN-vocabulary-filter
```

Here's another example using the start-transcription-job command, and a request body that identifies the language.

```
aws transcribe start-transcription-job \
--region us-west-2 \
--cli-input-json file://filepath/my-first-language-id-job.json
```

The file *my-first-language-id-job.json* contains the following request body.

Option 1: Without the `LanguageIdSettings` parameter. Use this option if you are **not** including a custom language model, custom vocabulary, or custom vocabulary filter in your request. `LanguageOptions` is optional, but recommended.

```
{
  "TranscriptionJobName": "my-first-transcription-job",
  "Media": {
        "MediaFileUri": "s3://amzn-s3-demo-bucket/my-input-files/my-media-file.flac"
   },
  "OutputBucketName": "amzn-s3-demo-bucket",
  "OutputKey": "my-output-files/",
```

```
  "IdentifyLanguage": true,  (or "IdentifyMultipleLanguages": true),
  "LanguageOptions": [
        "en-US", "hi-IN"
  ]
}
```

Option 2: With the `LanguageIdSettings` parameter. Use this option if you **are** including a custom language model, custom vocabulary, or custom vocabulary filter in your request.

```
{
   "TranscriptionJobName": "my-first-transcription-job",
   "Media": {
        "MediaFileUri": "s3://amzn-s3-demo-bucket/my-input-files/my-media-file.flac"
   },
   "OutputBucketName": "amzn-s3-demo-bucket",
   "OutputKey": "my-output-files/",
   "IdentifyLanguage": true,  (or "IdentifyMultipleLanguages": true)
   "LanguageOptions": [
        "en-US", "hi-IN"
   ],
   "LanguageIdSettings": {
         "en-US" : {
            "LanguageModelName": "my-en-US-language-model",
            "VocabularyFilterName": "my-en-US-vocabulary-filter",
            "VocabularyName": "my-en-US-vocabulary"
         },
         "hi-IN": {
             "VocabularyName": "my-hi-IN-vocabulary",
             "VocabularyFilterName": "my-hi-IN-vocabulary-filter"
         }
    }
}
```

AWS SDK for Python (Boto3)

This example uses the AWS SDK for Python (Boto3) to identify your file's language using the `IdentifyLanguage` argument for the start_transcription_job method. For more information, see `StartTranscriptionJob` and `LanguageIdSettings`.

For additional examples using the AWS SDKs, including feature-specific, scenario, and cross-service examples, refer to the Code examples for Amazon Transcribe using AWS SDKs chapter.

Option 1: Without the `LanguageIdSettings` parameter. Use this option if you are **not** including a custom language model, custom vocabulary, or custom vocabulary filter in your request. `LanguageOptions` is optional, but recommended.

```
from __future__ import print_function
import time
import boto3
transcribe = boto3.client('transcribe', 'us-west-2')
job_name = "my-first-transcription-job"
job_uri = "s3://amzn-s3-demo-bucket/my-input-files/my-media-file.flac"
transcribe.start_transcription_job(
    TranscriptionJobName = job_name,
    Media = {
        'MediaFileUri': job_uri
    },
    OutputBucketName = 'amzn-s3-demo-bucket',
    OutputKey = 'my-output-files/',
    MediaFormat = 'flac',
    IdentifyLanguage = True,  (or IdentifyMultipleLanguages = True),
    LanguageOptions = [
        'en-US', 'hi-IN'
    ]
)

while True:
    status = transcribe.get_transcription_job(TranscriptionJobName = job_name)
    if status['TranscriptionJob']['TranscriptionJobStatus'] in ['COMPLETED', 'FAILED']:
        break
    print("Not ready yet...")
    time.sleep(5)
print(status)
```

Option 2: With the `LanguageIdSettings` parameter. Use this option if you **are** including a custom language model, custom vocabulary, or custom vocabulary filter in your request.

```
from __future__ import print_function
import time
import boto3
transcribe = boto3.client('transcribe')
job_name = "my-first-transcription-job"
job_uri = "s3://amzn-s3-demo-bucket/my-input-files/my-media-file.flac"
transcribe.start_transcription_job(
```

```
    TranscriptionJobName = job_name,
    Media = {
        'MediaFileUri': job_uri
    },
    OutputBucketName = 'amzn-s3-demo-bucket',
    OutputKey = 'my-output-files/',
    MediaFormat='flac',
    IdentifyLanguage=True,  (or IdentifyMultipleLanguages=True)
    LanguageOptions = [
        'en-US', 'hi-IN'
    ],
    LanguageIdSettings={
        'en-US': {
            'VocabularyName': 'my-en-US-vocabulary',
            'VocabularyFilterName': 'my-en-US-vocabulary-filter',
            'LanguageModelName': 'my-en-US-language-model'
        },
        'hi-IN': {
            'VocabularyName': 'my-hi-IN-vocabulary',
            'VocabularyFilterName': 'my-hi-IN-vocabulary-filter'
        }
    }
)

while True:
    status = transcribe.get_transcription_job(TranscriptionJobName = job_name)
    if status['TranscriptionJob']['TranscriptionJobStatus'] in ['COMPLETED', 'FAILED']:
        break
    print("Not ready yet...")
    time.sleep(5)
print(status)
```

Language identification with streaming transcriptions

Streaming language identification can identify the dominant language spoken in your media stream. Amazon Transcribe requires a minimum of one second of speech to identify the language.

If your stream contains only one language, you can enable single-language identification, which identifies the dominant language spoken in your media file and creates your transcript using only this language.

If your stream contains more than one language, you can enable multi-language identification, which identifies all languages spoken in your stream and creates your transcript using each identified language. Note that a multi-lingual transcript is produced. You can use other services, such as Amazon Transcribe, to translate your transcript.

To use streaming language identification, you must provide at least two language codes, and you can select only one language dialect per language per stream. This means that you cannot select `en-US` and `en-AU` as language options for the same transcription.

You also have the option to select a preferred language from the set of language codes you provide. Adding a preferred language can speed up the language identification process, which is helpful for short audio clips.

Important

If none of the language codes you provide match the language, or languages, identified in your audio, Amazon Transcribe selects the closest language match from your specified language codes. It then produces a transcript in that language. For example, if your media is in US English (`en-US`) and you provide Amazon Transcribe with the language codes `zh-CN`, `fr-FR`, and `de-DE`, Amazon Transcribe is likely to match your media to German (`de-DE`) and produce a German-language transcription. Mismatching language codes and spoken languages can result in an inaccurate transcript, so we recommend caution when including language codes.

If your media contains two channels, Amazon Transcribe can identify the dominant language spoken in each channel. In this case, set the `ChannelIdentification` parameter to `true` and each channel is transcribed separately. Note that the default for this parameter is `false`. If you don't change it, only the first channel is transcribed and only one language is identified.

Streaming language identification can't be combined with custom language models or redaction. If combining language identification with other features, you are limited to the languages supported with those features, and also with streaming transcriptions. Refer to Supported languages.

Note

PCM and FLAC are the only supported audio formats for streaming language identification. For multi-language identification, only PCM is supported.

Identifying languages in multi-language audio

Multi-language identification is intended for multi-lingual streams, and provides you with a transcript that reflects all supported languages spoken in your stream. This means that if speakers change languages mid-conversation, or if each participant is speaking a different language, your transcription output detects and transcribes each language correctly.

For example, if your stream contains a bilingual speaker who is alternating between US English (`en-US`) and Hindi (`hi-IN`), multi-language identification can identify and transcribe spoken US English as `en-US` and spoken Hindi as `hi-IN`. This differs from single-language identification, where only one dominant language is used to create a transcript. In this case, any spoken language that is not the dominant language is transcribed incorrectly.

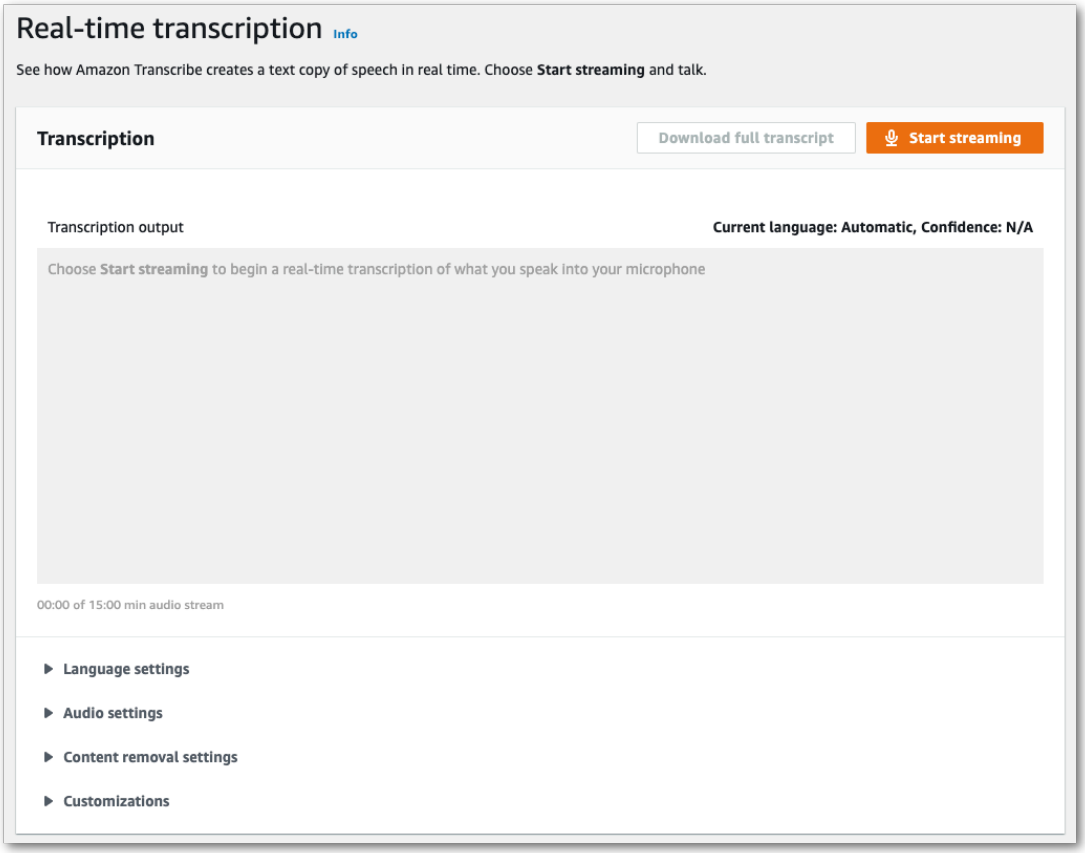
> **Note**
>
> Redaction and custom language models are not currently supported with multi-language identification.

Using language identification with streaming media

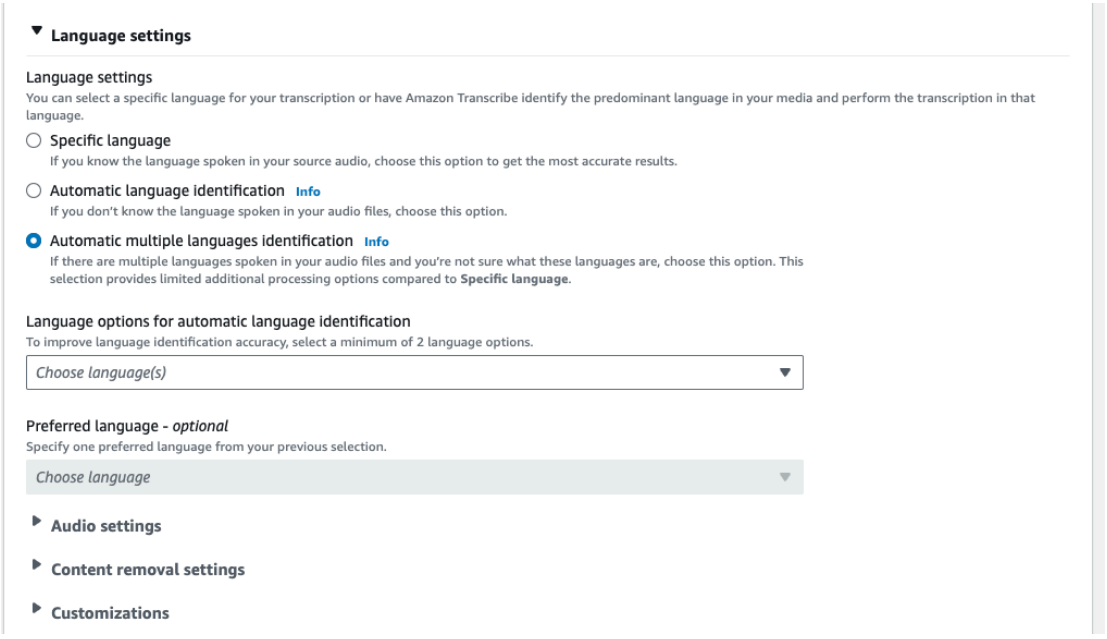
You can use automatic language identification in a streaming transcription using the **AWS Management Console**, **HTTP/2**, or **WebSockets**; see the following for examples:

AWS Management Console

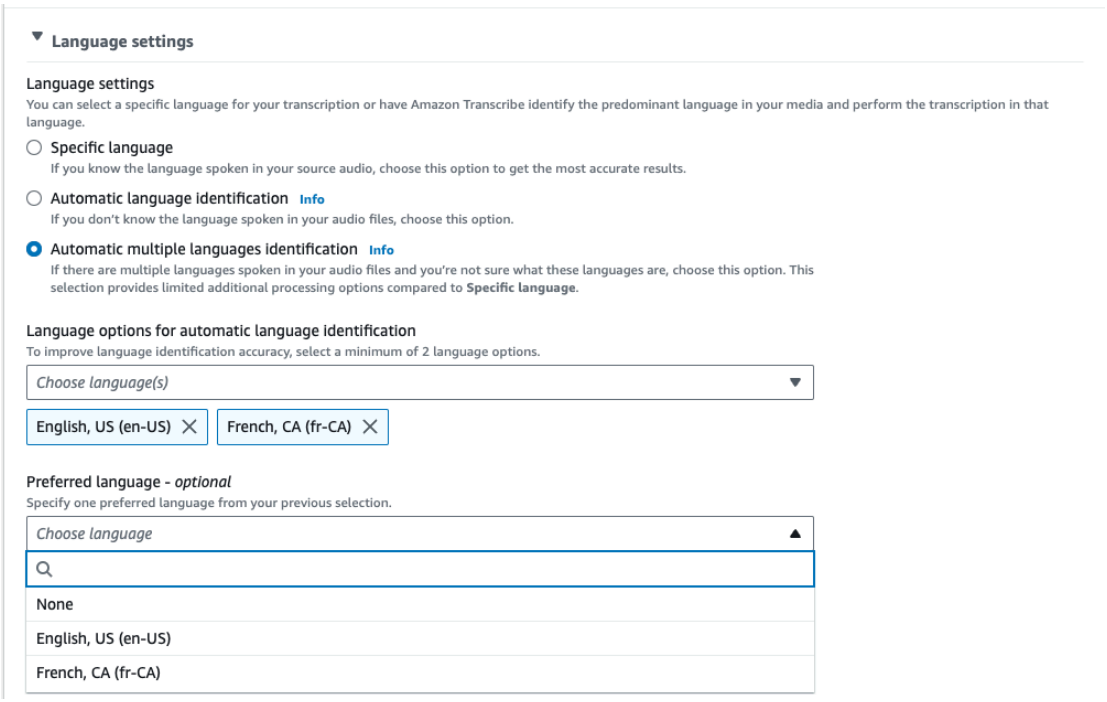
1. Sign in to the AWS Management Console.
2. In the navigation pane, choose **Real-time transcription**. Scroll down to **Language settings** and expand this field if it is minimized.



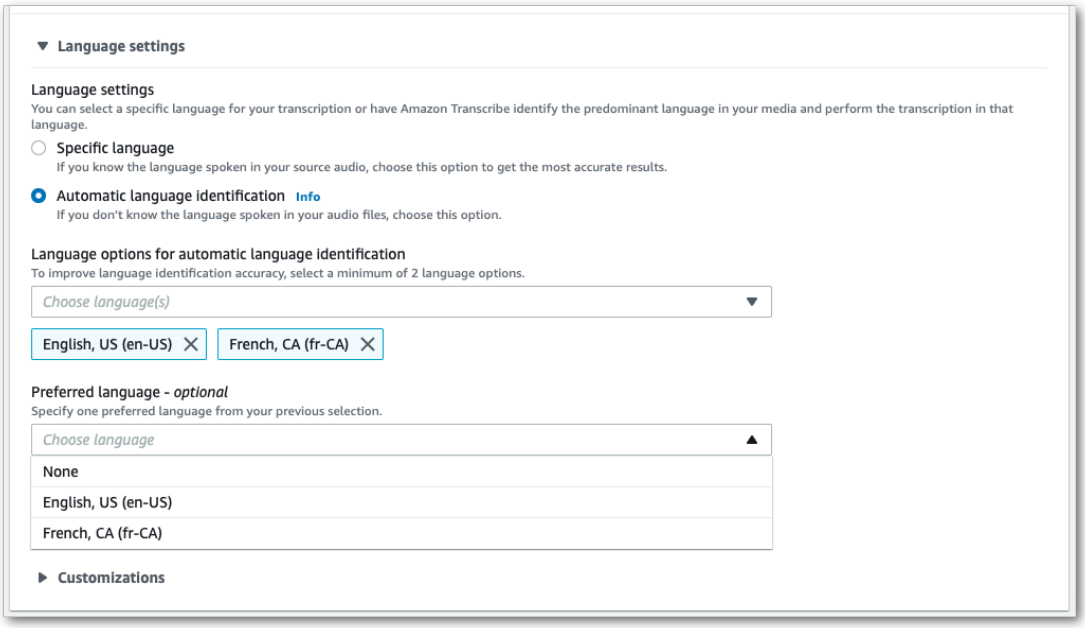
3. Select **Automatic language identification** or **Automatic multiple languages identification**.



4. Provide a minimum of two language codes for your transcription. Note that you can provide only one dialect per language. For example, you cannot select both `en-US` and `en-GB` as language options for the same transcription.



5. (Optional) From the subset of languages you selected in the previous step, you can choose a preferred language for your transcript.



6. You're now ready to transcribe your stream. Select **Start streaming** and begin speaking. To end your dictation, select **Stop streaming**.

HTTP/2 stream

This example creates an HTTP/2 request with language identification enabled. For more information on using HTTP/2 streaming with Amazon Transcribe, see Setting up an HTTP/2 stream. For more detail on parameters and headers specific to Amazon Transcribe, see StartStreamTranscription.

```
POST /stream-transcription HTTP/2
host: transcribestreaming.us-west-2.amazonaws.com
X-Amz-Target: com.amazonaws.transcribe.Transcribe.StartStreamTranscription
Content-Type: application/vnd.amazon.eventstream
X-Amz-Content-Sha256: string
X-Amz-Date: 20220208T235959Z
Authorization: AWS4-HMAC-SHA256 Credential=access-key/20220208/us-west-2/transcribe/
aws4_request, SignedHeaders=content-type;host;x-amz-content-sha256;x-amz-date;x-amz-
target;x-amz-security-token, Signature=string
x-amzn-transcribe-media-encoding: flac
x-amzn-transcribe-sample-rate: 16000
x-amzn-transcribe-identify-language: true
x-amzn-transcribe-language-options: en-US,de-DE
x-amzn-transcribe-preferred-language: en-US
transfer-encoding: chunked
```

This example creates an HTTP/2 request with multiple languages identification enabled. For more information on using HTTP/2 streaming with Amazon Transcribe, see Setting up an HTTP/2 stream. For more detail on parameters and headers specific to Amazon Transcribe, see StartStreamTranscription.

```
POST /stream-transcription HTTP/2
host: transcribestreaming.us-west-2.amazonaws.com
X-Amz-Target: com.amazonaws.transcribe.Transcribe.StartStreamTranscription
Content-Type: application/vnd.amazon.eventstream
X-Amz-Content-Sha256: string
X-Amz-Date: 20220208T235959Z
Authorization: AWS4-HMAC-SHA256 Credential=access-key/20220208/us-west-2/transcribe/
aws4_request, SignedHeaders=content-type;host;x-amz-content-sha256;x-amz-date;x-amz-
target;x-amz-security-token, Signature=string
x-amzn-transcribe-media-encoding: flac
x-amzn-transcribe-sample-rate: 16000
x-amzn-transcribe-identify-multiple-languages: true
x-amzn-transcribe-language-options: en-US,de-DE
x-amzn-transcribe-preferred-language: en-US
```

```
transfer-encoding: chunked
```

If you use `identify-language` or `identify-multiple-languages` in your request, you must also include `language-options`. You cannot use both `language-code` and `identify-language` in the same request.

Parameter definitions can be found in the API Reference; parameters common to all AWS API operations are listed in the Common Parameters section.

WebSocket stream

This example creates a presigned URL that uses language identification in a WebSocket stream. Line breaks have been added for readability. For more information on using WebSocket streams with Amazon Transcribe, see Setting up a WebSocket stream. For more detail on parameters, see `StartStreamTranscription`.

```
GET wss://transcribestreaming.us-west-2.amazonaws.com:8443/stream-transcription-
websocket?
&X-Amz-Algorithm=AWS4-HMAC-SHA256
&X-Amz-Credential=AKIAIOSFODNN7EXAMPLE%2F20220208%2Fus-
west-2%2Ftranscribe%2Faws4_request
&X-Amz-Date=20220208T235959Z
&X-Amz-Expires=300
&X-Amz-Security-Token=security-token
&X-Amz-Signature=string
&X-Amz-SignedHeaders=content-type%3Bhost%3Bx-amz-date
&media-encoding=flac
&sample-rate=16000
&identify-language=true
&language-options=en-US,de-DE
&preferred-language=en-US
```

This example creates a presigned URL that uses multiple languages identification in a WebSocket stream. Line breaks have been added for readability. For more information on using WebSocket streams with Amazon Transcribe, see Setting up a WebSocket stream. For more detail on parameters, see `StartStreamTranscription`.

```
GET wss://transcribestreaming.us-west-2.amazonaws.com:8443/stream-transcription-
websocket?
&X-Amz-Algorithm=AWS4-HMAC-SHA256
```

```
&X-Amz-Credential=AKIAIOSFODNN7EXAMPLE%2F20220208%2Fus-
west-2%2Ftranscribe%2Faws4_request
&X-Amz-Date=20220208T235959Z
&X-Amz-Expires=300
&X-Amz-Security-Token=security-token
&X-Amz-Signature=string
&X-Amz-SignedHeaders=content-type%3Bhost%3Bx-amz-date
&media-encoding=flac
&sample-rate=16000
&identify-multiple-languages=true
&language-options=en-US,de-DE
&preferred-language=en-US
```

If you use `identify-language` or `identify-multiple-languages` in your request, you must also include `language-options`. You cannot use both `language-code` and `identify-language` in the same request.

Parameter definitions can be found in the API Reference; parameters common to all AWS API operations are listed in the Common Parameters section.

Alternative transcriptions

When Amazon Transcribe transcribes audio, it creates different versions of the same transcript and assigns a confidence score to each version. In a typical transcription, you only get the version with the highest confidence score.

If you turn on alternative transcriptions, Amazon Transcribe returns other versions of your transcript that have lower confidence levels. You can choose to have up to 10 alternative transcriptions returned. If you specify a greater number of alternatives than what Amazon Transcribe identifies, only the actual number of alternatives is returned.

All alternatives are located in the same transcription output file and are presented at the segment level. Segments are natural pauses in speech, such as a change in speaker or a pause in the audio.

Alternative transcriptions are only available for batch transcriptions.

Your transcription output is structured as follows. The ellipses (`...`) in the code examples indicate where content has been removed for brevity.

1. A complete final transcription for a given segment.

```
"results": {
    "language_code": "en-US",
    "transcripts": [
        {
            "transcript": "The amazon is the largest rainforest on the planet."
        }
    ],
```

2. A confidence score for each word in the preceding `transcript` section.

```
"items": [
    {
        "start_time": "1.15",
        "end_time": "1.35",
        "alternatives": [
            {
                "confidence": "1.0",
                "content": "The"
            }
        ],
```

```
        "type": "pronunciation"
    },
    {
        "start_time": "1.35",
        "end_time": "2.05",
        "alternatives": [
            {
                "confidence": "1.0",
                "content": "amazon"
            }
        ],
        "type": "pronunciation"
    },
```

3. Your alternative transcriptions are located in the `segments` portion of your transcription output. The alternatives for each segment are ordered by descending confidence score.

```
"segments": [
            {
                "start_time": "1.04",
                "end_time": "5.065",
                "alternatives": [
                    {
                ...
                        "transcript": "The amazon is the largest rain forest on the
 planet.",
                        "items": [
                            {
                             "start_time": "1.15",
                                "confidence": "1.0",
                                "end_time": "1.35",
                                "type": "pronunciation",
                                "content": "The"
                            },
                            ...
                            {
                                "start_time": "3.06",
                                "confidence": "0.0037",
                                "end_time": "3.38",
                                "type": "pronunciation",
                                "content": "rain"
                            },
                            {
```

```
                    "start_time": "3.38",
                    "confidence": "0.0037",
                    "end_time": "3.96",
                    "type": "pronunciation",
                    "content": "forest"
                },
```

4. A status at the end of your transcription output.

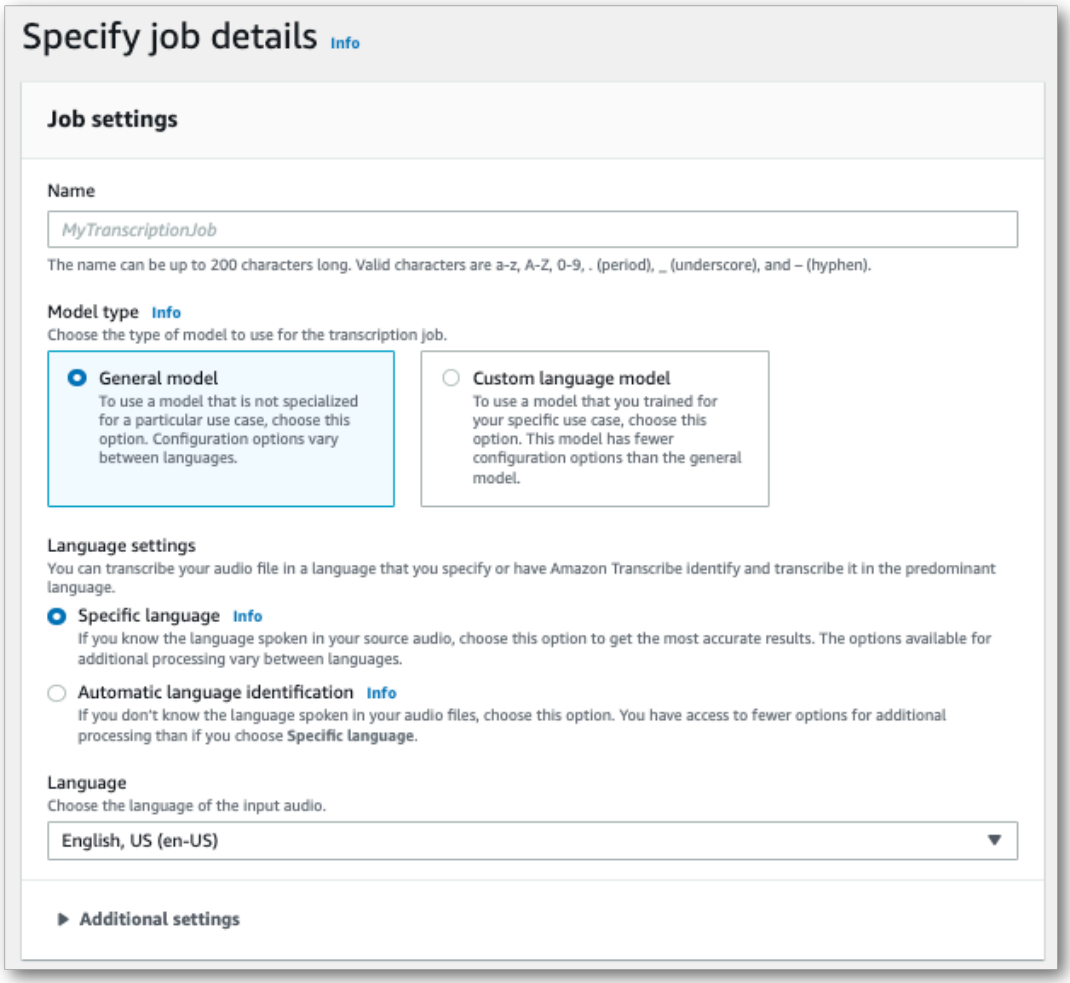
```
"status": "COMPLETED"
}
```

Requesting alternative transcriptions

You can request alternative transcriptions using the **AWS Management Console**, **AWS CLI**, or **AWS SDKs**; see the following for examples:

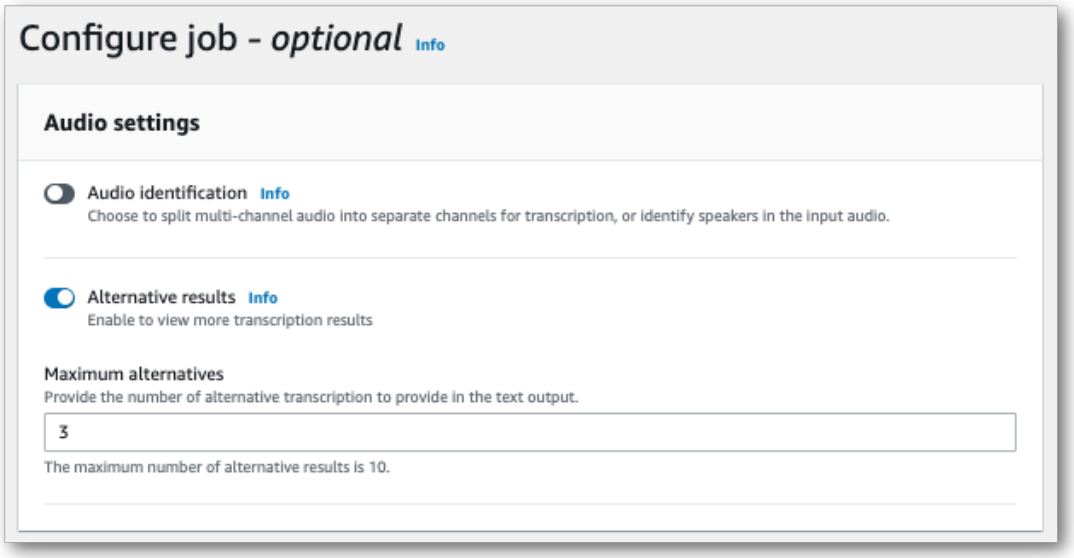
AWS Management Console

1. Sign in to the AWS Management Console.
2. In the navigation pane, choose **Transcription jobs**, then select **Create job** (top right). This opens the **Specify job details** page.



3. Fill in any fields you want to include on the **Specify job details** page, then select **Next**. This takes you to the **Configure job - *optional*** page.

   Select **Alternative results** and specify the maximum number of alternative transcription result you want in your transcript.



4. Select **Create job** to run your transcription job.

AWS CLI

This example uses the start-transcription-job command and `ShowAlternatives` parameter. For more information, see `StartTranscriptionJob` and `ShowAlternatives`.

Note that if you include `ShowAlternatives=true` in your request, you must also include `MaxAlternatives`.

```
aws transcribe start-transcription-job \
--region us-west-2 \
--transcription-job-name my-first-transcription-job \
--media MediaFileUri=s3://amzn-s3-demo-bucket/my-input-files/my-media-file.flac \
--output-bucket-name amzn-s3-demo-bucket \
--output-key my-output-files/ \
--language-code en-US \
--settings ShowAlternatives=true,MaxAlternatives=4
```

Here's another example using the start-transcription-job command, and a request body that includes alternative transcriptions.

```
aws transcribe start-transcription-job \
--region us-west-2 \
--cli-input-json file://filepath/my-first-alt-transcription-job.json
```

The file *my-first-alt-transcription-job.json* contains the following request body.

```
{
  "TranscriptionJobName": "my-first-transcription-job",
  "Media": {
        "MediaFileUri": "s3://amzn-s3-demo-bucket/my-input-files/my-media-file.flac"
   },
  "OutputBucketName": "amzn-s3-demo-bucket",
  "OutputKey": "my-output-files/",
  "LanguageCode": "en-US",
  "Settings": {
        "ShowAlternatives": true,
        "MaxAlternatives": 4
   }
}
```

AWS SDK for Python (Boto3)

The following example uses the AWS SDK for Python (Boto3) to request alternative transcriptions by using the `ShowAlternatives` argument for the start_transcription_job method. For more information, see `StartTranscriptionJob` and `ShowAlternatives`.

For additional examples using the AWS SDKs, including feature-specific, scenario, and cross-service examples, refer to the Code examples for Amazon Transcribe using AWS SDKs chapter.

Note that if you include `'ShowAlternatives':True` in your request, you must also include `MaxAlternatives`.

```
from __future__ import print_function
import time
import boto3
transcribe = boto3.client('transcribe', 'us-west-2')
job_name = "my-first-transcription-job"
job_uri = "s3://amzn-s3-demo-bucket/my-input-files/my-media-file.flac"
transcribe.start_transcription_job(
    TranscriptionJobName = job_name,
    Media = {
        'MediaFileUri': job_uri
    },
    OutputBucketName = 'amzn-s3-demo-bucket',
    OutputKey = 'my-output-files/',
    LanguageCode = 'en-US',
```

```
    Settings = {
        'ShowAlternatives':True,
        'MaxAlternatives':4
    }
)

while True:
    status = transcribe.get_transcription_job(TranscriptionJobName = job_name)
    if status['TranscriptionJob']['TranscriptionJobStatus'] in ['COMPLETED', 'FAILED']:
        break
    print("Not ready yet...")
    time.sleep(5)
print(status)
```

Improving transcription accuracy with custom vocabularies and custom language models

If your media contains domain-specific or non-standard terms, such as brand names, acronyms, technical words, and jargon, Amazon Transcribe might not correctly capture these terms in your transcription output.

To correct transcription inaccuracies and customize your output for your specific use case, you can create Custom vocabularies and Custom language models.

- Custom vocabularies are designed to tune and boost both the recognition and the formatting of specific words in all contexts. This involves supplying Amazon Transcribe with words and, optionally, pronunciation and display forms.

  If Amazon Transcribe is not correctly rendering specific terms in your transcripts, you can create a custom vocabulary file that tells Amazon Transcribe how you want these terms displayed. This word-specific approach is most appropriate for correcting terms like brand names and acronyms.
- Custom language models are designed to capture the context associated with terms. This involves supplying Amazon Transcribe with a large volume of domain-specific text data.

  If Amazon Transcribe is not correctly rendering technical terms or is using the incorrect homophone in your transcripts, you can create a custom language model that teaches Amazon Transcribe your domain-specific language. For example, a custom language model can learn when to use 'floe' (ice floe) versus 'flow' (linear flow).

  This context-aware approach is most appropriate for transcribing large volumes of domain-specific speech. Custom language models can produce significant accuracy improvements over custom vocabularies alone. Although the API accepts both a custom language model and a custom vocabulary in the same request, custom vocabulary effects (such as `DisplayAs` rendering) are not applied to the transcription output when a custom language model is active.

> **Important**
>
> Custom vocabulary features (such as the `DisplayAs` field) are not applied to transcription output when a custom language model is active in the same request. This applies to both batch and streaming transcriptions.

For a video demo on how to create a custom vocabulary using the AWS Management Console, see Using a custom vocabulary.

For a video demo on how to create and use custom language models, see Using Custom Language Models (CLM) to supercharge transcription accuracy.

Dive deeper with the AWS Machine Learning Blog

Custom vocabularies:

- Live transcriptions of F1 races using Amazon Transcribe

Custom language models:

- Building custom language models to supercharge speech-to-text performance Amazon Transcribe
- Boost transcription accuracy of class lectures with custom language models for Amazon Transcribe

Custom vocabularies

Use custom vocabularies to improve transcription accuracy for one or more specific words. These are generally domain-specific terms, such as brand names and acronyms, proper nouns, and words that Amazon Transcribe isn't rendering correctly.

Custom vocabularies can be used with all supported languages. Note that only the characters listed in your language's character set can be used in a custom vocabulary.

Important

You are responsible for the integrity of your own data when you use Amazon Transcribe. Do not enter confidential information, personal information (PII), or protected health information (PHI) into a custom vocabulary.

Considerations when creating a custom vocabulary:

- You can have up to 100 custom vocabulary files per AWS account

- The size limit for each custom vocabulary file is 50 Kb
- If using the API to create your custom vocabulary, your vocabulary file must be in text (*.txt) format. If using the AWS Management Console, your vocabulary file can be in text (*.txt) format or comma-separated value (*.csv) format.
- Each entry within a custom vocabulary cannot exceed 256 characters
- To use a custom vocabulary, it must have been created in the same AWS Region as your transcription.

Tip

You can test your custom vocabulary using the AWS Management Console. Once your custom vocabulary is ready to use, log in to the AWS Management Console, select **Real-time transcription**, scroll to **Customizations**, toggle on **Custom vocabulary**, and select your custom vocabulary from the dropdown list. Then select **start streaming**. Speak some of the words in your custom vocabulary into your microphone to see if they render correctly.

Custom vocabulary tables versus lists

Important

Custom vocabularies in list format are being deprecated. If you're creating a new custom vocabulary, use the table format.

Tables give you more options for—and more control over—the input and output of words within your custom vocabulary. With tables, you must specify multiple categories (Phrase and DisplayAs), allowing you to fine-tune your output.

Lists don't have additional options, so you can only type in entries as you want them to appear in your transcript, replacing all spaces with hyphens.

The AWS Management Console, AWS CLI, and AWS SDKs all use custom vocabulary tables in the same way; lists are used differently for each method and thus may require additional formatting for successful use between methods.

For more information, see Creating a custom vocabulary using a table and Creating a custom vocabulary using a list.

To dive a little deeper and learn how to use Amazon Augmented AI with custom vocabularies, see Start building a human review along with Amazon Transcribe

API operations specific to custom vocabularies

`CreateVocabulary`, `DeleteVocabulary`, `GetVocabulary`, `ListVocabularies`, `UpdateVocabulary`

Creating a custom vocabulary using a table

Using a table format is the preferred way to create your custom vocabulary. Vocabulary tables must consist of four columns (Phrase, SoundsLike, IPA, and DisplayAs), which can be included in any order:

Phrase	SoundsLike	IPA	DisplayAs
Required. Every row in your table must contain an entry in this column. Do not use spaces in this column. If your entry contains multiple words, separate each word with a hyphen (-). For example, **`Andorra-la-Vella`** or **`Los-Angeles`** . For acronyms, any pronounced letters must be separated by	`SoundsLike` is no longer supported for Custom Vocabulary. Please leave the column empty. Any values in this column will be ignored. We will remove the support for this column in the future.	`IPA` is no longer supported for Custom Vocabulary. Please leave the column empty. Any values in this column will be ignored. We will remove the support for this column in the future.	Optional. Rows in this column can be left empty. You can use spaces in this column. Defines the how you want your entry to look in your transcription output. For example, **`Andorra-la-Vella`** in the `Phrase` column is **`Andorra la Vella`** in the `DisplayAs` column.

Phrase	SoundsLike	IPA	DisplayAs
a period. The trailing period also needs to be pronounced. If your acronym is plural, you must use a hyphen between the acronym and the 's'. For example, 'CLI' is **C.L.I.** (not **C.L.I**) and 'ABCs' is **A.B.C.-s** (not **A.B.C-s**). If your phrase consists of both a word and an acronym, these two components must be separated by a hyphen. For example, 'DynamoDB' is **Dynamo-D.B.** . Do not include digits in this column; numbers must be spelled out. For example, 'VX02Q' is **V.X.-zero-two-Q.**.			If a row in this column is empty, Amazon Transcribe uses the contents of the `Phrase` column to determine output. You can include digits (`0-9`) in this column.

Things to note when creating your table:

- Your table must contain all four column headers (Phrase, SoundsLike, IPA, and DisplayAs). The `Phrase` column must contain an entry on each row. The ability to provide pronunciation inputs

through `IPA` and `SoundsLike` is no longer supported and you may leave the column empty. Any values in these columns will be ignored.

- Each column must be TAB or comma (,) delineated; this applies to every row in your custom vocabulary file. If a row contains empty columns, you must still include a delineator (TAB or comma) for each column.
- Spaces are only allowed within the `IPA` and `DisplayAs` columns. Do not use spaces to separate columns.
- `IPA` and `SoundsLike` are no longer supported for Custom Vocabulary. Please leave the column empty. Any values in these column will be ignored. We will remove the support for this column in the future.
- The `DisplayAs` column supports symbols and special characters (for example, C++). All other columns support the characters that are listed on your language's character set page.
- The `Phrase` column has the following formatting rules:
  - Cannot start with a period (`.`), apostrophe (`'`), or hyphen (`-`). For example, **`-hello`** and **`.test`** are invalid.
  - Cannot end with an apostrophe (`'`) or hyphen (`-`). For example, **`hello-`** and **`world'`** are invalid.
  - Cannot contain repeated hyphens (`--`), repeated periods (`..`), or repeated apostrophes (`''`). For example, **`well--known`** and **`can''t`** are invalid.
  - Periods can only be used to denote acronyms (single letters separated by periods, such as **`A.B.C.`**). A period cannot appear after two or more consecutive non-special characters (for example, **`AB.C`** is invalid), and cannot be followed by three or more consecutive non-special characters (for example, **`A.BCD`** is invalid).
  - Only characters listed in your language's character set are supported.
- If you want to include numbers in the `Phrase` column, you must spell them out. Digits (`0-9`) are only supported in the `DisplayAs` column.
- You must save your table as a plaintext (*.txt) file. Both `LF` and `CRLF` line endings are supported.
- You must upload your custom vocabulary file into an Amazon S3 bucket and process it using `CreateVocabulary` before you can include it in a transcription request. Refer to Creating custom vocabulary tables for instructions.

> **Note**
>
> Enter acronyms, or other words whose letters should be pronounced individually, as single letters separated by periods (**`A.B.C.`**). To enter the plural form of an acronym, such as 'ABCs', separate the 's' from the acronym with a hyphen (**`A.B.C.-s`**). You can use upper or lower case letters to define an acronym. Acronyms are not supported in all languages; refer to Supported languages and language-specific features.

Here is a sample custom vocabulary table (where **`[TAB]`** represents a tab character):

```
Phrase[TAB]SoundsLike[TAB]IPA[TAB]DisplayAs
Los-Angeles[TAB][TAB][TAB]Los Angeles
Eva-Maria[TAB][TAB][TAB]
A.B.C.-s[TAB][TAB][TAB]ABCs
Amazon-dot-com[TAB][TAB][TAB]Amazon.com
C.L.I.[TAB][TAB][TAB]CLI
Andorra-la-Vella[TAB][TAB][TAB]Andorra la Vella
Dynamo-D.B.[TAB][TAB][TAB]DynamoDB
V.X.-zero-two[TAB][TAB][TAB]VX02
V.X.-zero-two-Q.[TAB][TAB][TAB]VX02Q
```

For visual clarity, here is the same table with aligned columns. **Do not** add spaces between columns in your custom vocabulary table; your table should look misaligned like the preceding example.

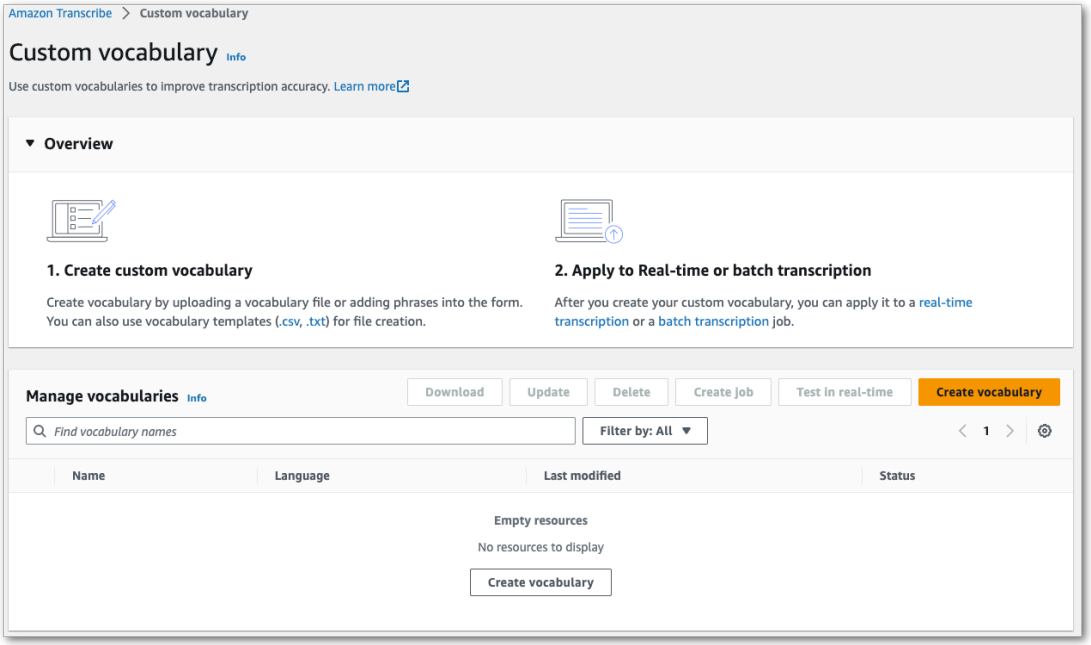
```
Phrase          [TAB]SoundsLike         [TAB]IPA            [TAB]DisplayAs
Los-Angeles     [TAB]                   [TAB]               [TAB]Los Angeles
Eva-Maria       [TAB]                   [TAB]               [TAB]
A.B.C.-s        [TAB]                   [TAB]               [TAB]ABCs
amazon-dot-com  [TAB]                   [TAB]               [TAB]amazon.com
C.L.I.          [TAB]                   [TAB]               [TAB]CLI
Andorra-la-Vella[TAB]                   [TAB]               [TAB]Andorra la Vella
Dynamo-D.B.     [TAB]                   [TAB]               [TAB]DynamoDB
V.X.-zero-two   [TAB]                   [TAB]               [TAB]VX02
V.X.-zero-two-Q.[TAB]                   [TAB]               [TAB]VX02Q
```

Creating custom vocabulary tables

To process a custom vocabulary table for use with Amazon Transcribe, see the following examples:

AWS Management Console

1. Sign in to the AWS Management Console.
2. In the navigation pane, choose **Custom vocabulary**. This opens the **Custom vocabulary** page where you can view existing vocabularies or create a new one.
3. Select **Create vocabulary**.

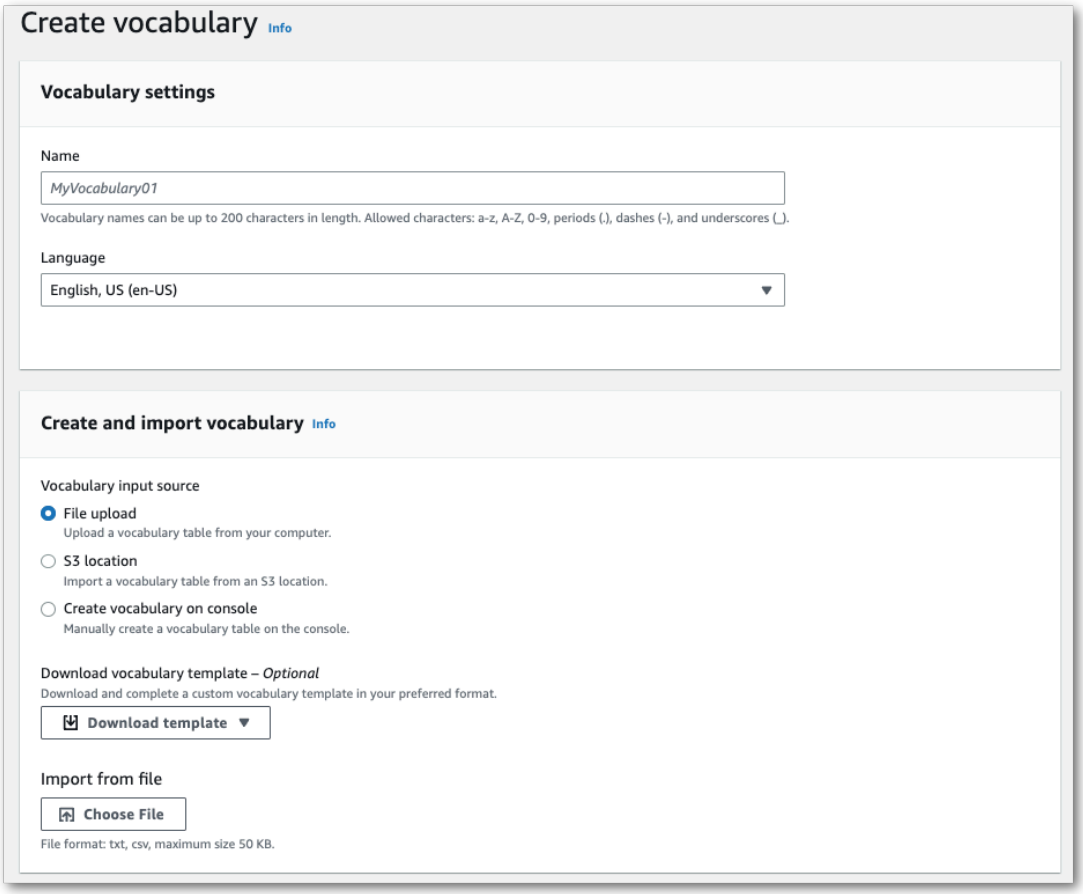


This takes you to the **Create vocabulary** page. Enter a name for your new custom vocabulary.

Here, you have three options:

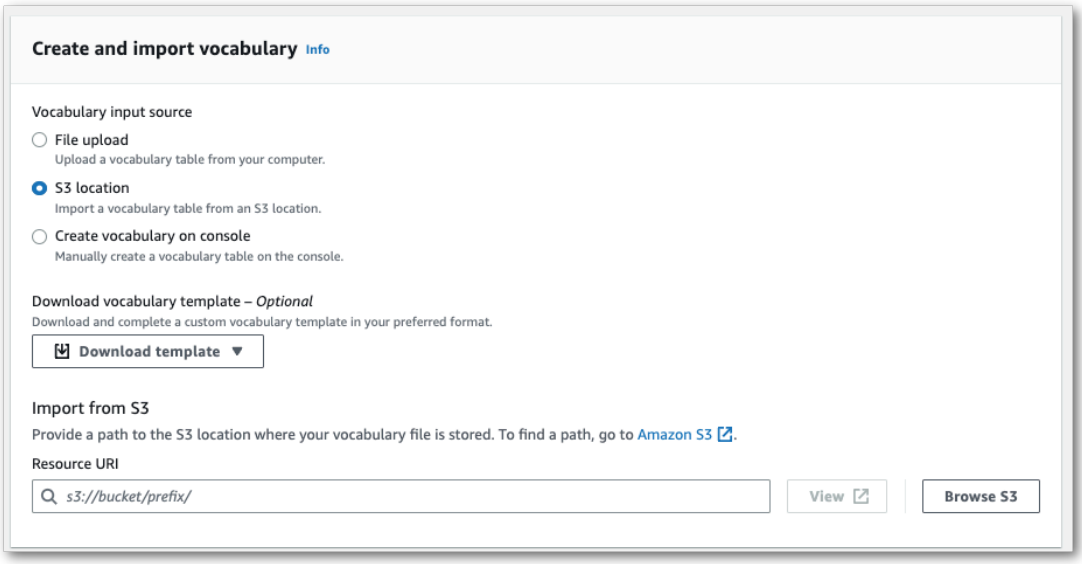
a. Upload a txt or csv file from your computer.

   You can either create your custom vocabulary from scratch or download a template to help you get started. Your vocabulary is then auto-populated in the **View and edit vocabulary** pane.



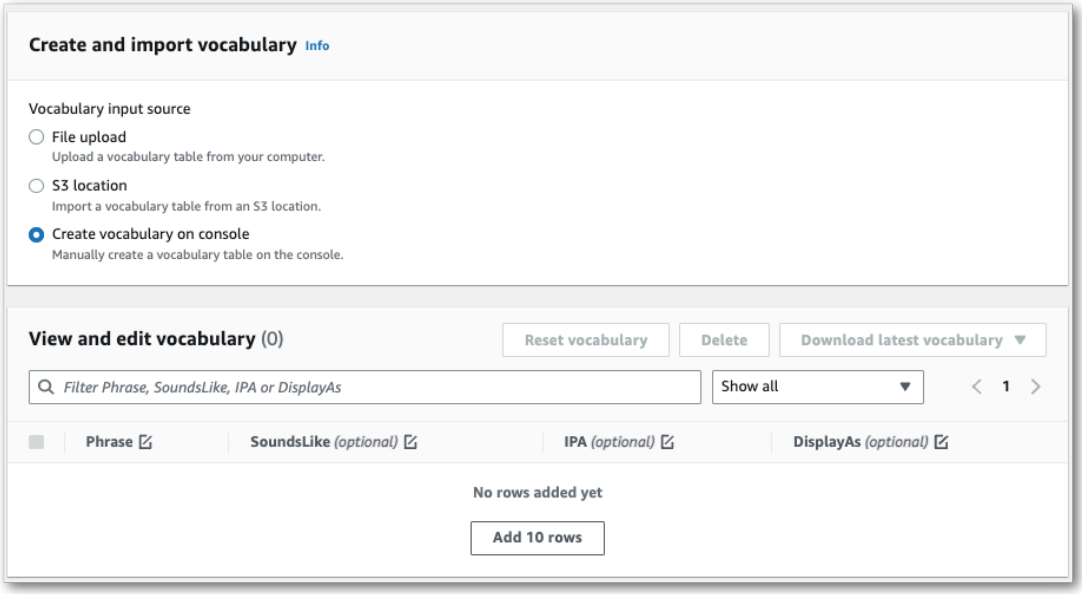
b. Import a txt or csv file from an Amazon S3 location.

   You can either create your custom vocabulary from scratch or download a template to help you get started. Upload your finished vocabulary file to an Amazon S3 bucket and specify its URI in your request. Your vocabulary is then auto-populated in the **View and edit vocabulary** pane.

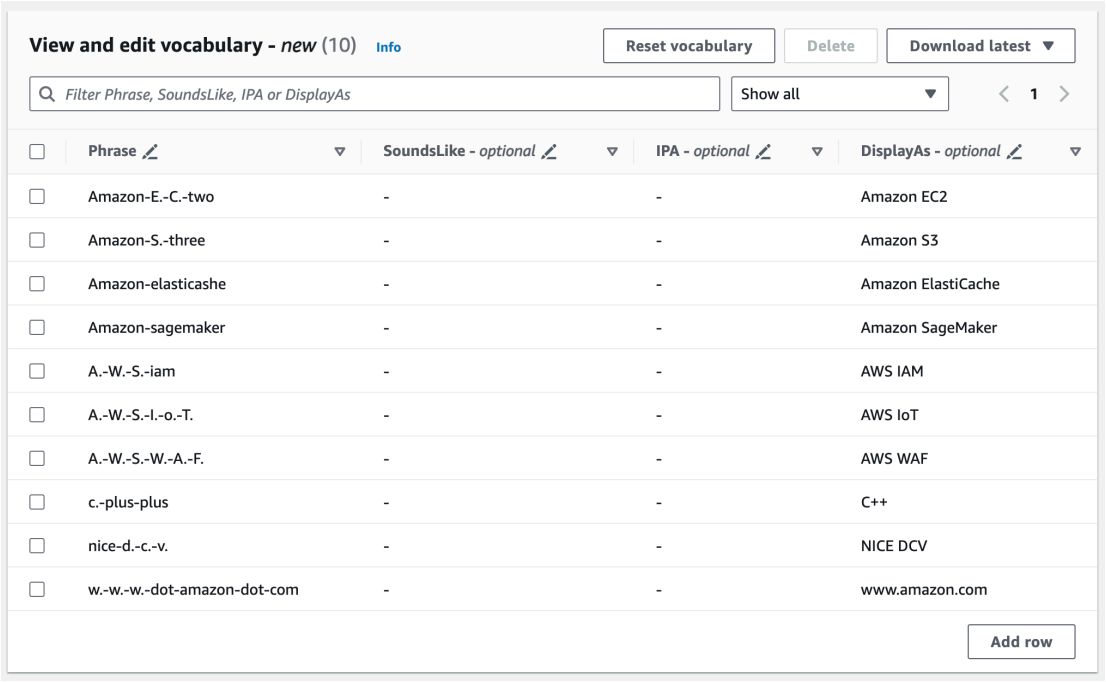


c. Manually create your vocabulary in the console.

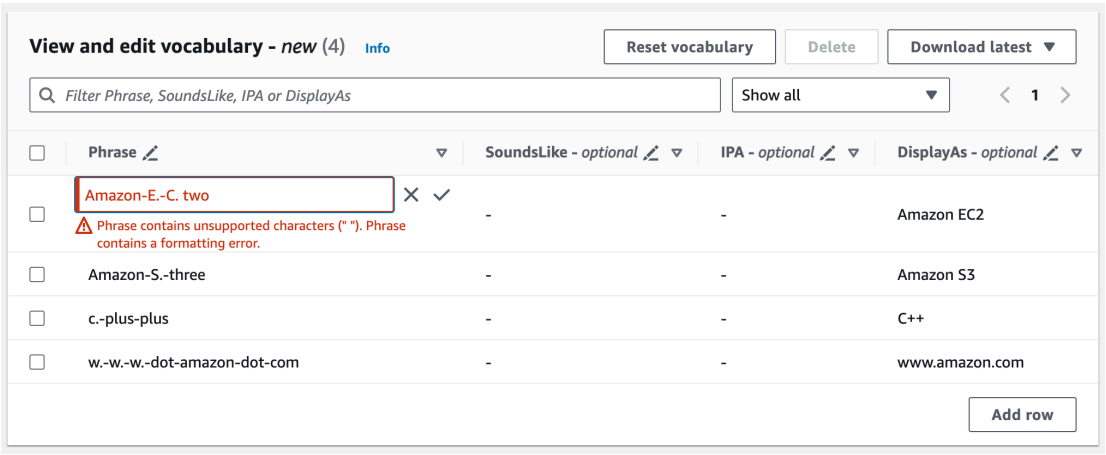
Scroll to the **View and edit vocabulary** pane and select **Add 10 rows**. You can now manually enter terms.



4. You can edit your vocabulary the **View and edit vocabulary** pane. To make changes, click on the entry you want to modify.

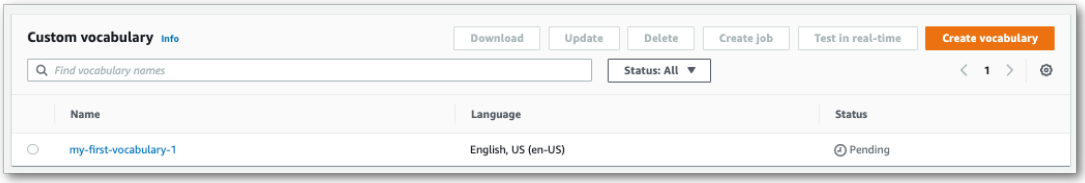


If you make an error, you get a detailed error message so you can correct any issues prior to processing your vocabulary. Note that if you don't correct all errors before selecting **Create vocabulary**, your vocabulary request fails.

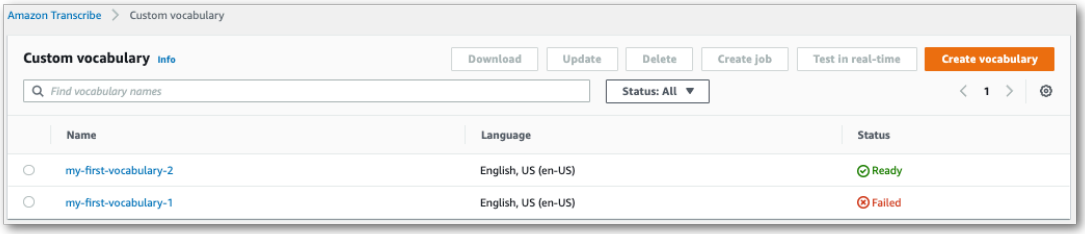


Select the check mark (✓) to save your changes or the 'X' to discard your changes.

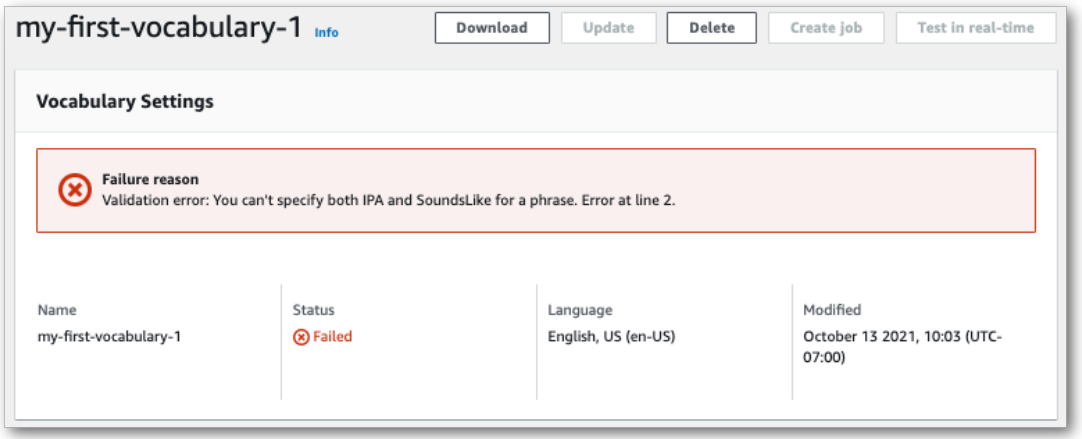
5. Optionally, add tags to your custom vocabulary. Once you have all fields completed and are happy with your vocabulary, select **Create vocabulary** at the bottom of the page. This takes you back to the **Custom vocabulary** page where you can view the status of your custom vocabulary. When the status changes from 'Pending' to 'Ready' your custom vocabulary can be used with a transcription.



6. If the status changes to 'Failed', select the name of your custom vocabulary to go to its information page.



There is a **Failure reason** banner at the top of this page that provides information on why your custom vocabulary failed. Correct the error in your text file and try again.



AWS CLI

This example uses the create-vocabulary command with a table-formatted vocabulary file. For more information, see `CreateVocabulary`.

To use an existing custom vocabulary in a transcription job, set the `VocabularyName` in the `Settings` field when you call the `StartTranscriptionJob` operation or, from the AWS Management Console, choose the custom vocabulary from the dropdown list.

```
aws transcribe create-vocabulary \
--vocabulary-name my-first-vocabulary \
```

```
--vocabulary-file-uri s3://amzn-s3-demo-bucket/my-vocabularies/my-vocabulary-file.txt \
--language-code en-US
```

Here's another example using the create-vocabulary command, and a request body that creates your custom vocabulary.

```
aws transcribe create-vocabulary \
--cli-input-json file://filepath/my-first-vocab-table.json
```

The file *my-first-vocab-table.json* contains the following request body.

```
{
  "VocabularyName": "my-first-vocabulary",
  "VocabularyFileUri": "s3://amzn-s3-demo-bucket/my-vocabularies/my-vocabulary-
table.txt",
  "LanguageCode": "en-US"
}
```

Once `VocabularyState` changes from `PENDING` to `READY`, your custom vocabulary is ready to use with a transcription. To view the current status of your custom vocabulary, run:

```
aws transcribe get-vocabulary \
--vocabulary-name my-first-vocabulary
```

AWS SDK for Python (Boto3)

This example uses the AWS SDK for Python (Boto3) to create a custom vocabulary from a table using the create_vocabulary method. For more information, see `CreateVocabulary`.

To use an existing custom vocabulary in a transcription job, set the `VocabularyName` in the `Settings` field when you call the `StartTranscriptionJob` operation or, from the AWS Management Console, choose the custom vocabulary from the dropdown list.

For additional examples using the AWS SDKs, including feature-specific, scenario, and cross-service examples, refer to the Code examples for Amazon Transcribe using AWS SDKs chapter.

```
from __future__ import print_function
import time
import boto3
transcribe = boto3.client('transcribe', 'us-west-2')
vocab_name = "my-first-vocabulary"
```

```
response = transcribe.create_vocabulary(
    LanguageCode = 'en-US',
    VocabularyName = vocab_name,
    VocabularyFileUri = 's3://amzn-s3-demo-bucket/my-vocabularies/my-vocabulary-
table.txt'
)

while True:
    status = transcribe.get_vocabulary(VocabularyName = vocab_name)
    if status['VocabularyState'] in ['READY', 'FAILED']:
        break
    print("Not ready yet...")
    time.sleep(5)
print(status)
```

Note

If you create a new Amazon S3 bucket for your custom vocabulary files, make sure the IAM role making the `CreateVocabulary` request has permissions to access this bucket. If the role doesn't have the correct permissions, your request fails. You can optionally specify an IAM role within your request by including the `DataAccessRoleArn` parameter. For more information on IAM roles and policies in Amazon Transcribe, see Amazon Transcribe identity-based policy examples.

Note

To encrypt your custom vocabulary artifacts at rest with a customer-managed KMS key, include an `EncryptionConfiguration` in your `CreateVocabulary` request. For more information about encrypting resource artifacts with a customer-managed key, see Encrypting resource artifacts with a customer-managed key.

Creating a custom vocabulary using a list

Important

Custom vocabularies in list format are being deprecated, so if you're creating a new custom vocabulary, we strongly recommend using the table format.

You can create custom vocabularies from lists using the AWS Management Console, AWS CLI, or AWS SDKs.

- **AWS Management Console**: You must create and upload a text file containing your custom vocabulary. You can use line-separated or comma-separated entries. Note that your list must be saved as a text (*.txt) file. Both LF and CRLF line endings are supported.
- **AWS CLI** and **AWS SDKs**: You must include your custom vocabulary as comma-separated entries within your API call using the `Phrases` flag.

If an entry contains multiple words, you must hyphenate each word. For example, you include 'Los Angeles' as **`Los-Angeles`** and 'Andorra la Vella' as **`Andorra-la-Vella`**.

Here are examples of the two valid list formats. Refer to Creating custom vocabulary lists for method-specific examples.

- Comma-separated entries:

```
Los-Angeles,CLI,Eva-Maria,ABCs,Andorra-la-Vella
```

- Line-separated entries:

```
Los-Angeles
CLI
Eva-Maria
ABCs
Andorra-la-Vella
```

Important

You can only use characters that are supported for your language. Refer to your language's character set for details.

Custom vocabulary lists are not supported with the `CreateMedicalVocabulary` operation. If creating a custom medical vocabulary, you must use a table format; refer to Creating a custom vocabulary using a table for instructions.

Creating custom vocabulary lists

To process a custom vocabulary list for use with Amazon Transcribe, see the following examples:

AWS CLI

This example uses the create-vocabulary command with a list-formatted custom vocabulary file. For more information, see `CreateVocabulary`.

```
aws transcribe create-vocabulary \
--vocabulary-name my-first-vocabulary \
--language-code en-US \
--phrases {CLI,Eva-Maria,ABCs}
```

Here's another example using the create-vocabulary command, and a request body that creates your custom vocabulary.

```
aws transcribe create-vocabulary \
--cli-input-json file://filepath/my-first-vocab-list.json
```

The file *my-first-vocab-list.json* contains the following request body.

```
{
  "VocabularyName": "my-first-vocabulary",
  "LanguageCode": "en-US",
  "Phrases": [
        "CLI","Eva-Maria","ABCs"
  ]
}
```

Once `VocabularyState` changes from `PENDING` to `READY`, your custom vocabulary is ready to use with a transcription. To view the current status of your custom vocabulary, run:

```
aws transcribe get-vocabulary \
--vocabulary-name my-first-vocabulary
```

AWS SDK for Python (Boto3)

This example uses the AWS SDK for Python (Boto3) to create a custom vocabulary from a list using the create_vocabulary method. For more information, see `CreateVocabulary`.

For additional examples using the AWS SDKs, including feature-specific, scenario, and cross-service examples, refer to the Code examples for Amazon Transcribe using AWS SDKs chapter.

```
from __future__ import print_function
import time
import boto3
transcribe = boto3.client('transcribe', 'us-west-2')
vocab_name = "my-first-vocabulary"
response = transcribe.create_vocabulary(
    LanguageCode = 'en-US',
    VocabularyName = vocab_name,
    Phrases = [
        'CLI','Eva-Maria','ABCs'
    ]
)

while True:
    status = transcribe.get_vocabulary(VocabularyName = vocab_name)
    if status['VocabularyState'] in ['READY', 'FAILED']:
        break
    print("Not ready yet...")
    time.sleep(5)
print(status)
```

Note

If you create a new Amazon S3 bucket for your custom vocabulary files, make sure the IAM role making the `CreateVocabulary` request has permissions to access this bucket. If the role doesn't have the correct permissions, your request fails. You can optionally specify an IAM role within your request by including the `DataAccessRoleArn` parameter. For more information on IAM roles and policies in Amazon Transcribe, see Amazon Transcribe identity-based policy examples.

Note

To encrypt your custom vocabulary artifacts at rest with a customer-managed KMS key, include an `EncryptionConfiguration` in your `CreateVocabulary` request. For

more information about encrypting resource artifacts with a customer-managed key, see Encrypting resource artifacts with a customer-managed key.

Using a custom vocabulary

Once your custom vocabulary is created, you can include it in your transcription requests; refer to the following sections for examples.

A custom vocabulary must be in the READY state before you can use it in a transcription request.

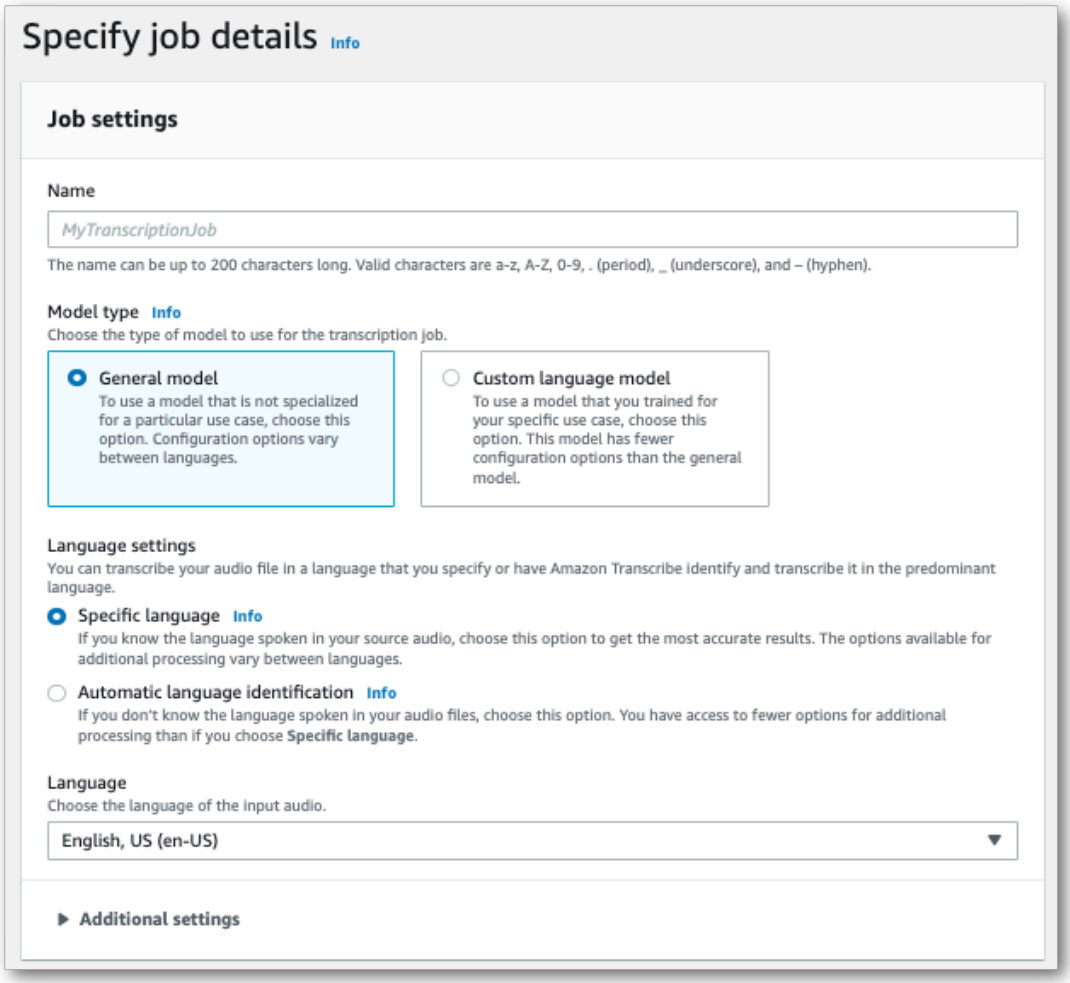
The language of the custom vocabulary you're including in your request must match the language code you specify for your media. If the languages don't match, your custom vocabulary is not applied to your transcription and there are no warnings or errors.

Using a custom vocabulary in a batch transcription

To use a custom vocabulary with a batch transcription, see the following for examples:

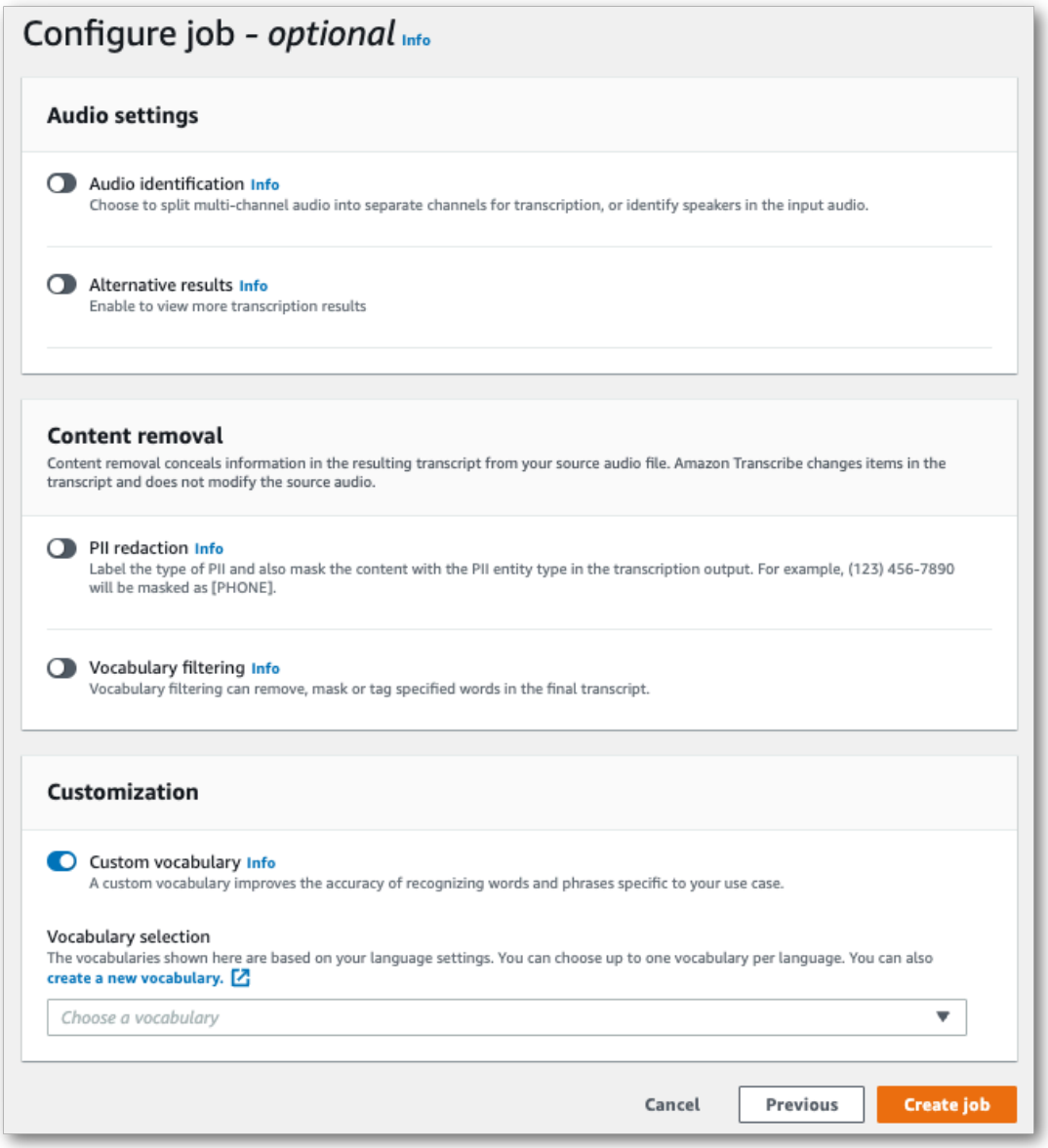
AWS Management Console

1. Sign in to the AWS Management Console.
2. In the navigation pane, choose **Transcription jobs**, then select **Create job** (top right). This opens the **Specify job details** page.



Name your job and specify your input media. Optionally include any other fields, then choose **Next**.

3. At the bottom of the **Configure job** page, in the **Customization** panel, toggle on **Custom vocabulary**.



4. Select your custom vocabulary from the dropdown menu.

   Select **Create job** to run your transcription job.

AWS CLI

This example uses the start-transcription-job command and `Settings` parameter with the `VocabularyName` sub-parameter. For more information, see `StartTranscriptionJob` and `Settings`.

```
aws transcribe start-transcription-job \
--region us-west-2 \
```

```
--transcription-job-name my-first-transcription-job \
--media MediaFileUri=s3://amzn-s3-demo-bucket/my-input-files/my-media-file.flac \
--output-bucket-name amzn-s3-demo-bucket \
--output-key my-output-files/ \
--language-code en-US \
--settings VocabularyName=my-first-vocabulary
```

Here's another example using the start-transcription-job command, and a request body that includes your custom vocabulary with that job.

```
aws transcribe start-transcription-job \
--region us-west-2 \
--cli-input-json file://my-first-vocabulary-job.json
```

The file *my-first-vocabulary-job.json* contains the following request body.

```
{
  "TranscriptionJobName": "my-first-transcription-job",
  "Media": {
        "MediaFileUri": "s3://amzn-s3-demo-bucket/my-input-files/my-media-file.flac"
  },
  "OutputBucketName": "amzn-s3-demo-bucket",
  "OutputKey": "my-output-files/",
  "LanguageCode": "en-US",
  "Settings": {
        "VocabularyName": "my-first-vocabulary"
   }
}
```

AWS SDK for Python (Boto3)

This example uses the AWS SDK for Python (Boto3) to include a custom vocabulary using the `Settings` argument for the start_transcription_job method. For more information, see `StartTranscriptionJob` and `Settings`.

For additional examples using the AWS SDKs, including feature-specific, scenario, and cross-service examples, refer to the Code examples for Amazon Transcribe using AWS SDKs chapter.

```
from __future__ import print_function
import time
import boto3
```

```
transcribe = boto3.client('transcribe', 'us-west-2')
job_name = "my-first-transcription-job"
job_uri = "s3://amzn-s3-demo-bucket/my-input-files/my-media-file.flac"
transcribe.start_transcription_job(
    TranscriptionJobName = job_name,
    Media = {
        'MediaFileUri': job_uri
    },
    OutputBucketName = 'amzn-s3-demo-bucket',
    OutputKey = 'my-output-files/',
    LanguageCode = 'en-US',
    Settings = {
        'VocabularyName': 'my-first-vocabulary'
   }
)

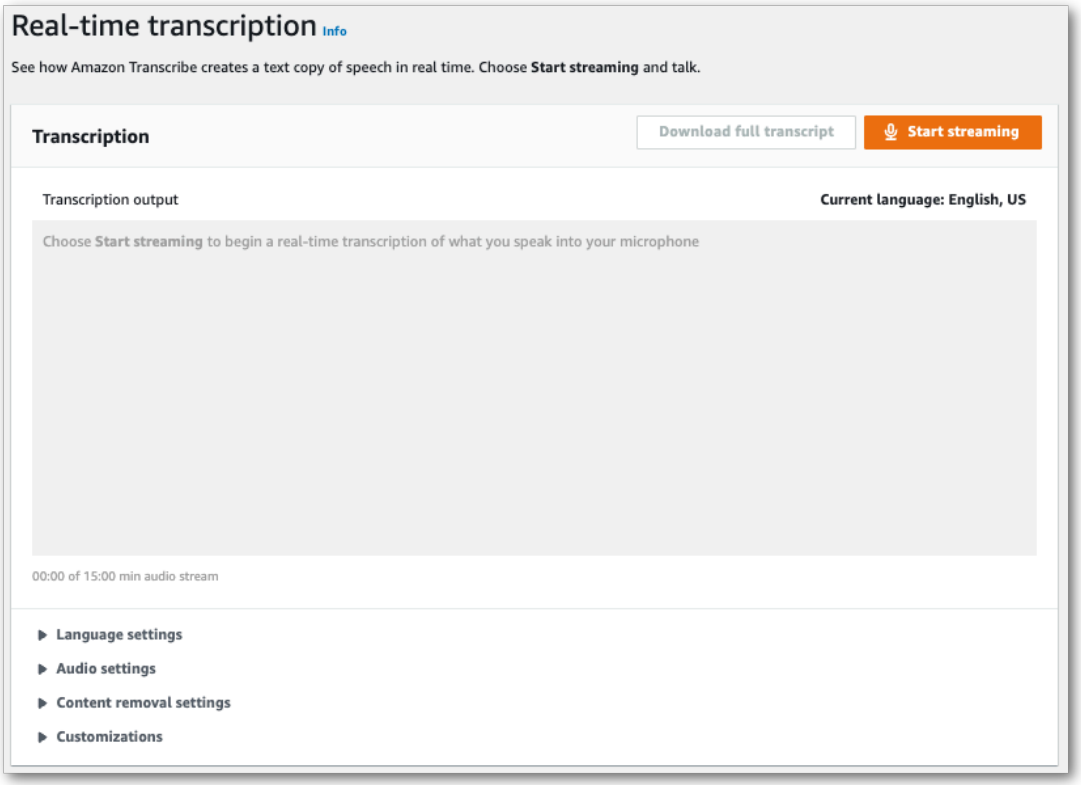
while True:
    status = transcribe.get_transcription_job(TranscriptionJobName = job_name)
    if status['TranscriptionJob']['TranscriptionJobStatus'] in ['COMPLETED', 'FAILED']:
        break
    print("Not ready yet...")
    time.sleep(5)
print(status)
```

Using a custom vocabulary in a streaming transcription

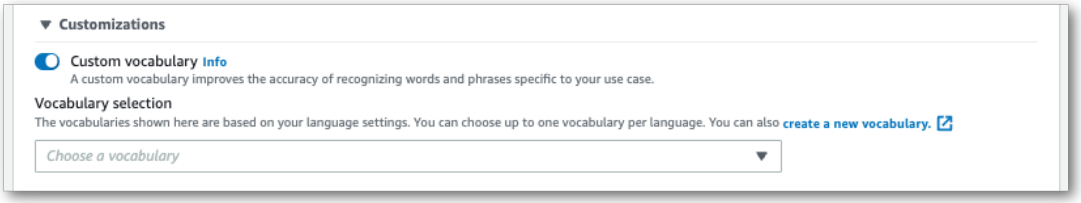
To use a custom vocabulary with a streaming transcription, see the following for examples:

AWS Management Console

1. Sign into the AWS Management Console.
2. In the navigation pane, choose **Real-time transcription**. Scroll down to **Customizations** and expand this field if it is minimized.



3. Toggle on **Custom vocabulary** and select a custom vocabulary from the dropdown menu.



   Include any other settings that you want to apply to your stream.

4. You're now ready to transcribe your stream. Select **Start streaming** and begin speaking. To end your dictation, select **Stop streaming**.

HTTP/2 stream

This example creates an HTTP/2 request that includes your custom vocabulary. For more information on using HTTP/2 streaming with Amazon Transcribe, see Setting up an HTTP/2 stream. For more detail on parameters and headers specific to Amazon Transcribe, see `StartStreamTranscription`.

```
POST /stream-transcription HTTP/2
```

```
host: transcribestreaming.us-west-2.amazonaws.com
X-Amz-Target: com.amazonaws.transcribe.Transcribe.StartStreamTranscription
Content-Type: application/vnd.amazon.eventstream
X-Amz-Content-Sha256: string
X-Amz-Date: 20220208T235959Z
Authorization: AWS4-HMAC-SHA256 Credential=access-key/20220208/us-west-2/transcribe/
aws4_request, SignedHeaders=content-type;host;x-amz-content-sha256;x-amz-date;x-amz-
target;x-amz-security-token, Signature=string
x-amzn-transcribe-language-code: en-US
x-amzn-transcribe-media-encoding: flac
x-amzn-transcribe-sample-rate: 16000
x-amzn-transcribe-vocabulary-name: my-first-vocabulary
transfer-encoding: chunked
```

Parameter definitions can be found in the API Reference; parameters common to all AWS API operations are listed in the Common Parameters section.

WebSocket stream

This example creates a presigned URL that applies your custom vocabulary to a WebSocket stream. Line breaks have been added for readability. For more information on using WebSocket streams with Amazon Transcribe, see Setting up a WebSocket stream. For more detail on parameters, see StartStreamTranscription.

```
GET wss://transcribestreaming.us-west-2.amazonaws.com:8443/stream-transcription-
websocket?
&X-Amz-Algorithm=AWS4-HMAC-SHA256
&X-Amz-Credential=AKIAIOSFODNN7EXAMPLE%2F20220208%2Fus-
west-2%2Ftranscribe%2Faws4_request
&X-Amz-Date=20220208T235959Z
&X-Amz-Expires=300
&X-Amz-Security-Token=security-token
&X-Amz-Signature=string
&X-Amz-SignedHeaders=content-type%3Bhost%3Bx-amz-date
&language-code=en-US
&media-encoding=flac
&sample-rate=16000
&vocabulary-name=my-first-vocabulary
```

Parameter definitions can be found in the API Reference; parameters common to all AWS API operations are listed in the Common Parameters section.

Updating a custom vocabulary

To update the content of an existing custom vocabulary, use the `UpdateVocabulary` operation.

When updating a custom vocabulary, note the following:

- Your vocabulary must be in a terminal state (`READY` or `FAILED`). You cannot update a vocabulary while it is being processed. Use `GetVocabulary` to check the current state.
- The update replaces the existing vocabulary with the new content you provide.
- You must include either `Phrases` or `VocabularyFileUri` in your request.
- (Optional) Include an `EncryptionConfiguration` to encrypt your vocabulary with a customer-managed key. You must also provide the new vocabulary content, because encryption cannot be changed independently. For more information about encrypting resource artifacts with a customer-managed key, see Encrypting resource artifacts with a customer-managed key.

To update a custom vocabulary, see the following for examples:

AWS CLI

This example uses the update-vocabulary command. For more information, see `UpdateVocabulary`.

```
aws transcribe update-vocabulary \
--vocabulary-name my-first-vocabulary \
--vocabulary-file-uri s3://amzn-s3-demo-bucket/my-vocabularies/my-updated-vocabulary-file.txt \
--language-code en-US
```

Here's another example using the update-vocabulary command, and a request body that updates your custom vocabulary.

```
aws transcribe update-vocabulary \
--cli-input-json file://filepath/my-updated-vocab.json
```

The file *my-updated-vocab.json* contains the following request body.

```
{
  "VocabularyName": "my-first-vocabulary",
```

```
  "VocabularyFileUri": "s3://amzn-s3-demo-bucket/my-vocabularies/my-updated-vocabulary-
file.txt",
  "LanguageCode": "en-US"
}
```

Custom language models

Custom language models are designed to improve transcription accuracy for domain-specific speech. This includes any content outside of what you would hear in normal, everyday conversations. For example, if you're transcribing the proceedings from a scientific conference, a standard transcription is unlikely to recognize many of the scientific terms used by presenters. In this case, you can train a custom language model to recognize the specialized terms used in your discipline.

Unlike custom vocabularies, which increase recognition of a word by providing hints (such as pronunciations), custom language models learn the context associated with a given word. This includes how and when a word is used, and the relationship a word has to other words. For example, if you train your model using climate science research papers, your model may learn that 'ice floe' is a more likely word pair than 'ice flow'.

To view the supported languages for custom language models, refer to Supported languages and language-specific features. Note that if you include a custom language model in your request, you can't enable language identification (you must specify a language code).

API operations specific to custom language models

`CreateLanguageModel`, `DeleteLanguageModel`, `DescribeLanguageModel`, `ListLanguageModels`, `UpdateLanguageModel`

Data sources

You can use any type of text data you want to train your model. However, the closer your text content is to your audio content, the more accurate your model. Therefore, it's important to choose text data that use the same terms in the same context as your audio.

The best data for training a model are accurate transcripts. This is considered in-domain data. In-domain text data have the exact same terms, usage, and context as the audio you want to transcribe.

If you don't have accurate transcripts, use journal articles, technical reports, whitepapers, conference proceedings, instruction manuals, news articles, website content, and any other text that contains the desired terms used in a similar context to that of your audio. This is considered domain-related data.

Creating a robust custom language model may require a significant amount of text data, which must contain the terms spoken in your audio. You can supply Amazon Transcribe with up to 2 GB of text data to train your model—this is referred to as **training data**. Optionally, when you have no (or few) in-domain transcripts, you can provide Amazon Transcribe with up to 200 MB of text data to tune your model—this is referred to as **tuning data**.

Training versus tuning data

The purpose of training data is to teach Amazon Transcribe to recognize new terms and learn the context in which these terms are used. In order to create a robust model, Amazon Transcribe may require a large volume of relevant text data. Providing as much training data as possible, up to the 2 GB limit, is strongly recommended.

The purpose of tuning data is to help refine and optimize the contextual relationships learned from your training data. Tuning data are not required to create a custom language model.

It's up to you to decide how best to select training and, optionally, tuning data. Each case is unique and depends on the type and amount of data you have. Tuning data are recommended when you lack in-domain training data.

If you choose to include both data types, do **not** overlap your training and tuning data; training and tuning data should be unique. Overlapping data can bias and skew your custom language model, impacting its accuracy.

As general guidance, we recommend using accurate, in-domain text as training data whenever possible. Here are some general scenarios, listed in order of preference:

- If you have more than 10,000 words of accurate, in-domain transcript text, use it as training data. In this case, there is no need to include tuning data. This is the ideal scenario for training a custom language model.
- If you have accurate, in-domain transcript text that contains less than 10,000 words and are not getting the desired results, consider augmenting your training data with domain-related written texts, such as technical reports. In this case, reserve a small portion (10-25%) of your in-domain transcript data to use as tuning data.

- If you have no in-domain transcript text, upload all your domain-related text as training data. In this case, transcript-style text is preferable to written text. This is the least effective scenario for training a custom language model.

When you're ready to create your model, see Creating a custom language model.

Creating a custom language model

Before you can create your custom language model, you must:

- Prepare your data. Data must be saved in plain text format and can't contain any special characters.
- Upload your data into an Amazon S3 bucket. Creating separate folders for training and tuning data is recommended.
- Make sure Amazon Transcribe has access to your Amazon S3 bucket. You must specify an IAM role that has access permissions to use your data.

Preparing your data

You can compile all your data in one file or save it as multiple files. Note that if you choose to include tuning data, it must be saved in a separate file from your training data.

It doesn't matter how many text files you use for your training or tuning data. Uploading one file with 100,000 words produces the same result as uploading 10 files with 10,000 words. Prepare your text data in a way that's most convenient for you.

Make sure all your data files meet the following criteria:

- They are all in the same language as the model you want to create. For example, if you want to create a custom language model that transcribes audio in US English (en-US), all your text data must be in US English.
- They are in plain text format with UTF-8 encoding.
- They don't contain any special characters or formatting, such as HTML tags.
- They amount to a maximum combined total of 2 GB in size for training data and 200 MB for tuning data.

If any of these criteria are not met, your model fails.

Uploading your data

Before uploading your data, create a new folder for your training data. If using tuning data, create another separate folder.

The URIs for your buckets might look like:

- `s3://amzn-s3-demo-bucket/my-model-training-data/`
- `s3://amzn-s3-demo-bucket/my-model-tuning-data/`

Upload your training and tuning data into the appropriate buckets.

You can add more data to these buckets at a later date. However, if you do, you need to recreate your model with the new data. Existing models can't be updated with new data.

Allowing access to your data

To create a custom language model, you must specify an IAM role that has permissions to access your Amazon S3 bucket. If you don't already have a role with access to the Amazon S3 bucket where you placed your training data, you must create one. After you create a role, you can attach a policy to grant that role permissions. Do not attach a policy to a user.

For example policies, see Amazon Transcribe identity-based policy examples.

To learn how to create a new IAM identity, see IAM Identities (users, user groups, and roles).

To learn more about policies, see:

- Policies and permissions in IAM
- Creating IAM policies
- Access management for AWS resources

Creating your custom language model

When creating your custom language model, you must choose a base model. There are two base model options:

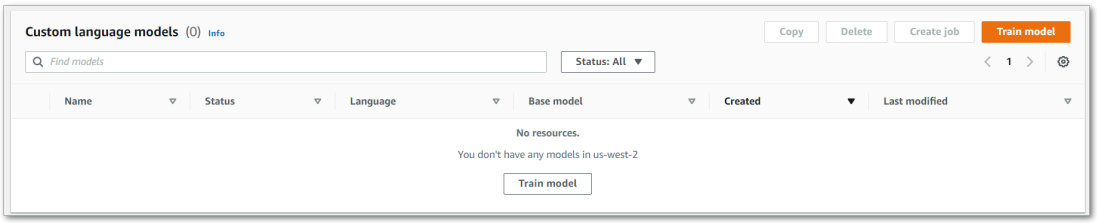
- `NarrowBand`: Use this option for audio with a sample rate of less than 16,000 Hz. This model type is typically used for telephone conversations recorded at 8,000 Hz.

- `WideBand`: Use this option for audio with a sample rate greater than or equal to 16,000 Hz.

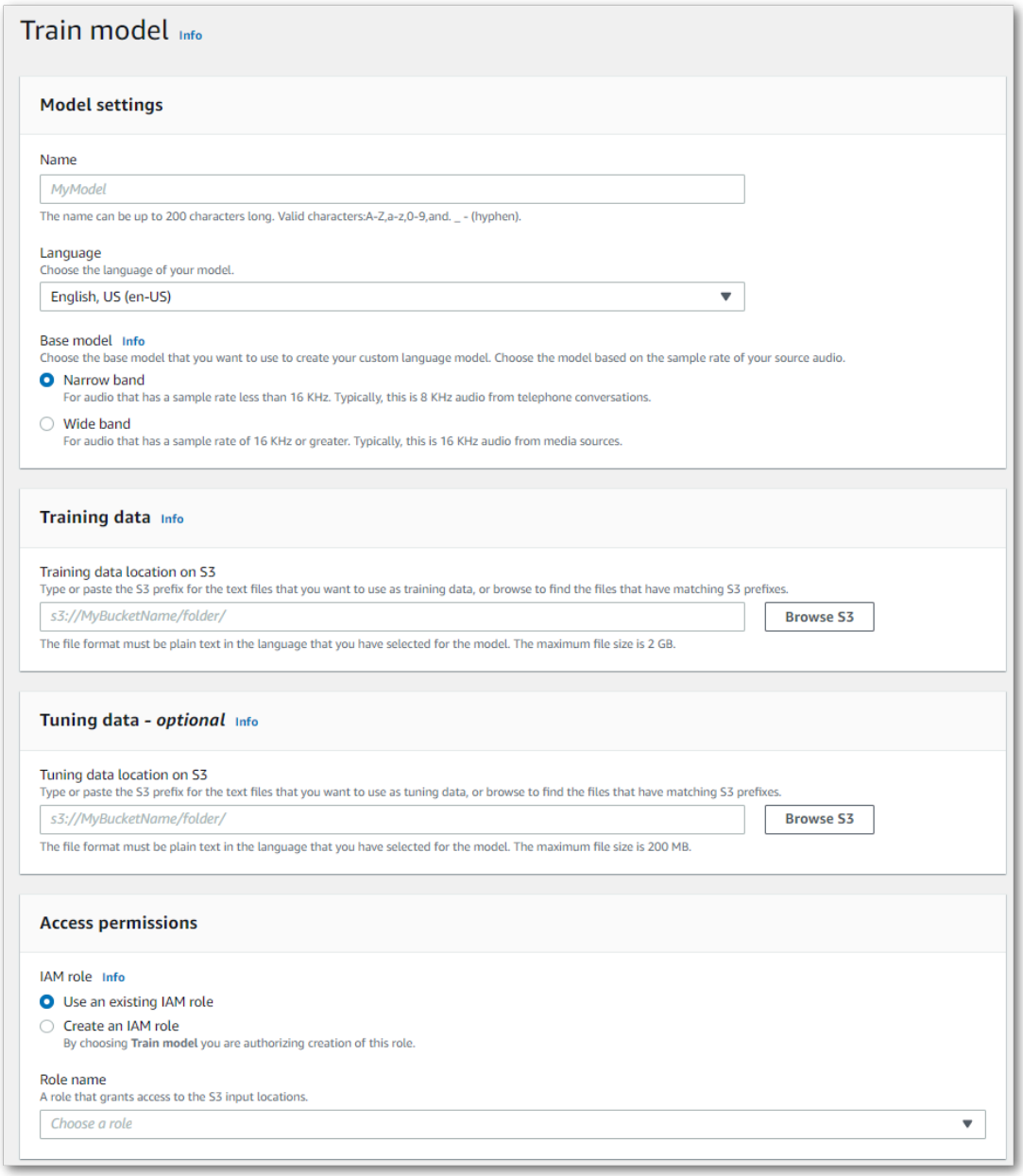
You can create custom language models using the AWS Management Console, AWS CLI, or AWS SDKs.; see the following examples:

AWS Management Console

1. Sign in to the AWS Management Console.
2. In the navigation pane, choose **Custom language model**. This opens the **Custom language models** page where you can view existing custom language models or train a new custom language model.
3. To train a new model, select **Train model**.



   This takes you to the **Train model** page. Add a name, specify the language, and choose the base model you want for your model. Then, add the path to your training and, optionally, your tuning data. You must include an IAM role that has permissions to access your data.



4. Once you have all fields completed, select **Train model** at the bottom of the page.

AWS CLI

This example uses the create-language-model command. For more information, see CreateLanguageModel and LanguageModel.

```
aws transcribe create-language-model \
--base-model-name NarrowBand \
--model-name my-first-language-model \
```

```
--input-data-config S3Uri=s3://amzn-s3-demo-bucket/my-clm-training-
data/,TuningDataS3Uri=s3://amzn-s3-demo-bucket/my-clm-tuning-
data/,DataAccessRoleArn=arn:aws:iam::111122223333:role/ExampleRole \
--language-code en-US
```

Here's another example using the create-language-model command, and a request body that creates your custom language model.

```
aws transcribe create-language-model \
--cli-input-json file://filepath/my-first-language-model.json
```

The file *my-first-language-model.json* contains the following request body.

```
{
  "BaseModelName": "NarrowBand",
  "ModelName": "my-first-language-model",
  "InputDataConfig": {
        "S3Uri": "s3://amzn-s3-demo-bucket/my-clm-training-data/",
        "TuningDataS3Uri"="s3://amzn-s3-demo-bucket/my-clm-tuning-data/",
        "DataAccessRoleArn": "arn:aws:iam::111122223333:role/ExampleRole"
    },
  "LanguageCode": "en-US"
}
```

AWS SDK for Python (Boto3)

This example uses the AWS SDK for Python (Boto3) to create a CLM using the create_language_model method. For more information, see CreateLanguageModel and LanguageModel.

For additional examples using the AWS SDKs, including feature-specific, scenario, and cross-service examples, refer to the Code examples for Amazon Transcribe using AWS SDKs chapter.

```
from __future__ import print_function
import time
import boto3
transcribe = boto3.client('transcribe', 'us-west-2')
model_name = 'my-first-language-model',
transcribe.create_language_model(
    LanguageCode = 'en-US',
    BaseModelName = 'NarrowBand',
    ModelName = model_name,
```

```
    InputDataConfig = {
        'S3Uri':'s3://amzn-s3-demo-bucket/my-clm-training-data/',
        'TuningDataS3Uri':'s3://amzn-s3-demo-bucket/my-clm-tuning-data/',
        'DataAccessRoleArn':'arn:aws:iam::111122223333:role/ExampleRole'
    }
)

while True:
    status = transcribe.get_language_model(ModelName = model_name)
    if status['LanguageModel']['ModelStatus'] in ['COMPLETED', 'FAILED']:
        break
    print("Not ready yet...")
    time.sleep(5)
print(status)
```

Note

To encrypt your custom language model artifacts at rest with a customer-managed KMS key, include an `EncryptionConfiguration` in your `CreateLanguageModel` request. For more information about encrypting resource artifacts with a customer-managed key, see Encrypting resource artifacts with a customer-managed key.

Upgrading your custom language model base

Amazon Transcribe continually updates the base models available for custom language models. To benefit from these updates, we recommend training new custom language models every 6 to 12 months.

To see if your custom language model is using the latest base model, run a `DescribeLanguageModel` request using the AWS CLI or an AWS SDK, then find the `UpgradeAvailability` field in your response.

If `UpgradeAvailability` is `true`, your model is not running the latest version of the base model. The base model of an existing custom language model cannot be upgraded in place. To use the latest base model, create a new custom language model.

Using a custom language model

Once you've created your custom language model, you can include it in your transcription requests; refer to the following sections for examples.

A custom language model must finish building before you can use it in a transcription request. Its `ModelStatus` is `COMPLETED` when the model is ready.

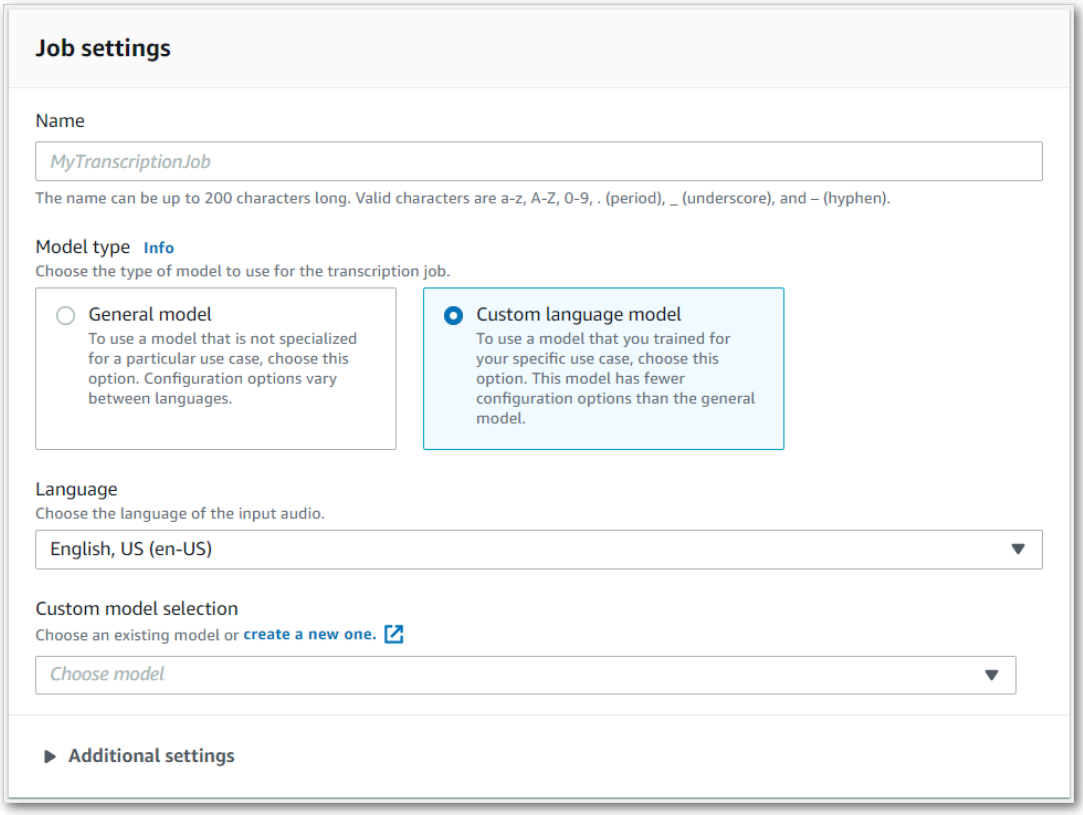
The language of the model you're including in your request must match the language code you specify for your media. If the languages don't match, your custom language model is not applied to your transcription and there are no warnings or errors.

Using a custom language model in a batch transcription

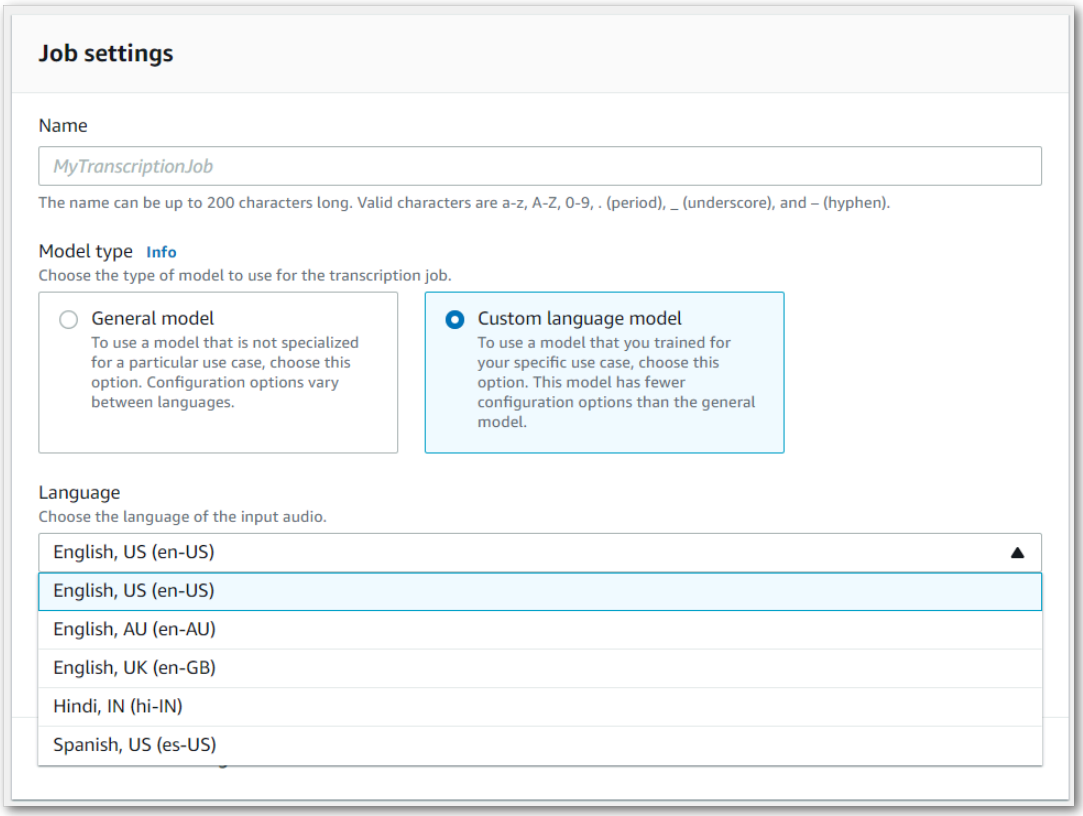
To use a custom language model with a batch transcription, see the following for examples:

AWS Management Console

1. Sign in to the AWS Management Console.
2. In the navigation pane, choose **Transcription jobs**, then select **Create job** (top right). This opens the **Specify job details** page.
3. In the **Job settings** panel under **Model type**, select the **Custom language model** box.



You must also select an input language from the dropdown menu.



4. Under **Custom model selection**, select an existing custom language model from the dropdown menu or **Create a new one**.

   Add the Amazon S3 location of your input file in the **Input data** panel.

5. Select **Next** for additional configuration options.

   Select **Create job** to run your transcription job.

AWS CLI

This example uses the start-transcription-job command and `ModelSettings` parameter with the `VocabularyName` sub-parameter. For more information, see `StartTranscriptionJob` and `ModelSettings`.

```
aws transcribe start-transcription-job \
--region us-west-2 \
--transcription-job-name my-first-transcription-job \
--media MediaFileUri=s3://amzn-s3-demo-bucket/my-input-files/my-media-file.flac \
--output-bucket-name amzn-s3-demo-bucket \
--output-key my-output-files/ \
```

```
--language-code en-US \
--model-settings LanguageModelName=my-first-language-model
```

Here's another example using the start-transcription-job command, and a request body that includes your custom language model with that job.

```
aws transcribe start-transcription-job \
--region us-west-2 \
--cli-input-json file://my-first-model-job.json
```

The file *my-first-model-job.json* contains the following request body.

```
{
  "TranscriptionJobName": "my-first-transcription-job",
  "Media": {
        "MediaFileUri": "s3://amzn-s3-demo-bucket/my-input-files/my-media-file.flac"
  },
  "OutputBucketName": "amzn-s3-demo-bucket",
  "OutputKey": "my-output-files/",
  "LanguageCode": "en-US",
  "ModelSettings": {
        "LanguageModelName": "my-first-language-model"
   }
}
```

AWS SDK for Python (Boto3)

This example uses the AWS SDK for Python (Boto3) to include a custom language model using the `ModelSettings` argument for the start_transcription_job method. For more information, see `StartTranscriptionJob` and `ModelSettings`.

For additional examples using the AWS SDKs, including feature-specific, scenario, and cross-service examples, refer to the Code examples for Amazon Transcribe using AWS SDKs chapter.

```
from __future__ import print_function
import time
import boto3
transcribe = boto3.client('transcribe', 'us-west-2')
job_name = "my-first-transcription-job"
job_uri = "s3://amzn-s3-demo-bucket/my-input-files/my-media-file.flac"
transcribe.start_transcription_job(
```

```
    TranscriptionJobName = job_name,
    Media = {
        'MediaFileUri': job_uri
    },
    OutputBucketName = 'amzn-s3-demo-bucket',
    OutputKey = 'my-output-files/',
    LanguageCode = 'en-US',
    ModelSettings = {
        'LanguageModelName': 'my-first-language-model'
   }
)

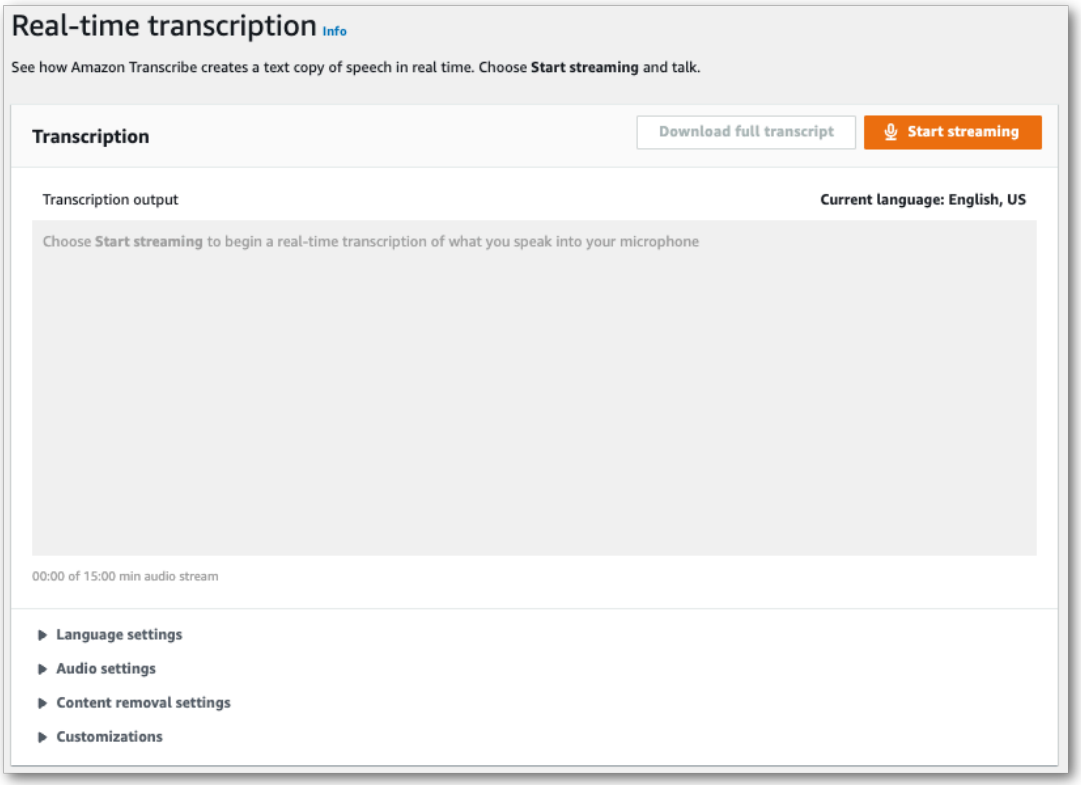
while True:
    status = transcribe.get_transcription_job(TranscriptionJobName = job_name)
    if status['TranscriptionJob']['TranscriptionJobStatus'] in ['COMPLETED', 'FAILED']:
        break
    print("Not ready yet...")
    time.sleep(5)
print(status)
```

Using a custom language model in a streaming transcription

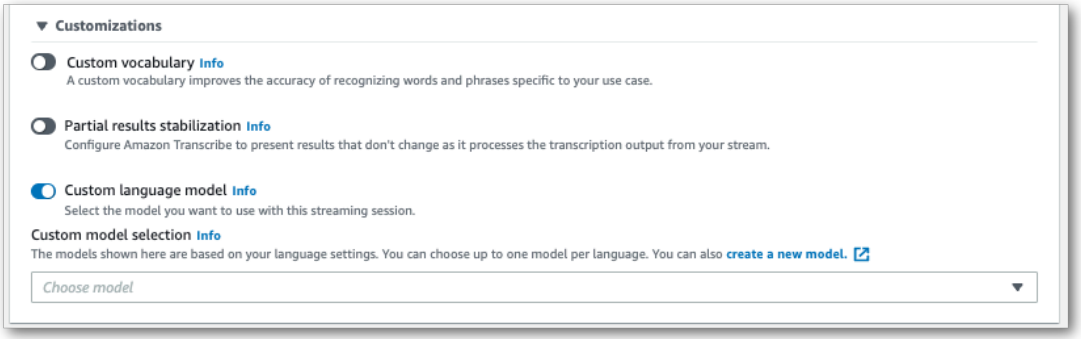
To use a custom language model with a streaming transcription, see the following for examples:

AWS Management Console

1. Sign into the AWS Management Console.
2. In the navigation pane, choose **Real-time transcription**. Scroll down to **Customizations** and expand this field if it is minimized.



3. Toggle on **Custom language model** and select a model from the dropdown menu.



   Include any other settings you want to apply to your stream.

4. You're now ready to transcribe your stream. Select **Start streaming** and begin speaking. To end your dictation, select **Stop streaming**.

HTTP/2 stream

This example creates an HTTP/2 request that includes your custom language model. For more information on using HTTP/2 streaming with Amazon Transcribe, see Setting up an

HTTP/2 stream. For more detail on parameters and headers specific to Amazon Transcribe, see StartStreamTranscription.

```
POST /stream-transcription HTTP/2
host: transcribestreaming.us-west-2.amazonaws.com
X-Amz-Target: com.amazonaws.transcribe.Transcribe.StartStreamTranscription
Content-Type: application/vnd.amazon.eventstream
X-Amz-Content-Sha256: string
X-Amz-Date: 20220208T235959Z
Authorization: AWS4-HMAC-SHA256 Credential=access-key/20220208/us-west-2/transcribe/
aws4_request, SignedHeaders=content-type;host;x-amz-content-sha256;x-amz-date;x-amz-
target;x-amz-security-token, Signature=string
x-amzn-transcribe-language-code: en-US
x-amzn-transcribe-media-encoding: flac
x-amzn-transcribe-sample-rate: 16000
x-amzn-transcribe-language-model-name: my-first-language-model
transfer-encoding: chunked
```

Parameter definitions can be found in the API Reference; parameters common to all AWS API operations are listed in the Common Parameters section.

WebSocket stream

This example creates a presigned URL that applies your custom language model to a WebSocket stream. Line breaks have been added for readability. For more information on using WebSocket streams with Amazon Transcribe, see Setting up a WebSocket stream. For more detail on parameters, see StartStreamTranscription.

```
GET wss://transcribestreaming.us-west-2.amazonaws.com:8443/stream-transcription-
websocket?
&X-Amz-Algorithm=AWS4-HMAC-SHA256
&X-Amz-Credential=AKIAIOSFODNN7EXAMPLE%2F20220208%2Fus-
west-2%2Ftranscribe%2Faws4_request
&X-Amz-Date=20220208T235959Z
&X-Amz-Expires=300
&X-Amz-Security-Token=security-token
&X-Amz-Signature=string
&X-Amz-SignedHeaders=content-type%3Bhost%3Bx-amz-date
&language-code=en-US
&media-encoding=flac
&sample-rate=16000
&language-model-name=my-first-language-model
```

Parameter definitions can be found in the API Reference; parameters common to all AWS API operations are listed in the Common Parameters section.

Changing encryption configuration for a custom language model

To change the encryption configuration of an existing custom language model, use the `UpdateLanguageModel` operation. This operation does not support changes to the model content or training data. To retrain a model, create a new custom language model.

When changing encryption for a custom language model, note the following:

- Your model must not be currently processing an update. You cannot submit another update while a previous update is in progress. Use `DescribeLanguageModel` to check the current state.

> **Note**
>
> In the isolated AWS Regions, avoid using customer-managed key encryption for custom language models that are trained on very large amounts of text if you plan to use them for streaming transcription. Because of compute capacity constraints in these Regions, decrypting the artifacts of a large model when it is loaded for a streaming transcription can be slow or fail to complete. Batch transcription is not affected. To use a large custom language model for streaming in these Regions, do not provide a customer-managed key. The model is then encrypted with the default AWS-owned key.

To update a custom language model, see the following for examples:

AWS CLI

This example uses the update-language-model command. For more information, see `UpdateLanguageModel`.

```
aws transcribe update-language-model \
--model-name my-first-language-model \
--encryption-configuration KMSKey=arn:aws:kms:us-
west-2:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab \
--data-access-role-arn arn:aws:iam::111122223333:role/ExampleRole
```

Here's another example using the update-language-model command, and a request body that updates your custom language model.

```
aws transcribe update-language-model \
--cli-input-json file://filepath/my-updated-language-model.json
```

The file *my-updated-language-model.json* contains the following request body.

```
{
  "ModelName": "my-first-language-model",
  "EncryptionConfiguration": {
      "KMSKey": "arn:aws:kms:us-
west-2:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab"
  },
  "DataAccessRoleArn": "arn:aws:iam::111122223333:role/ExampleRole"
}
```

Using custom vocabulary filters to delete, mask, or flag words

A custom vocabulary filter is a text file that contains a custom list of individual words that you want to modify in your transcription output.

A common use case is the removal of offensive or profane terms; however, custom vocabulary filters are completely custom, so you can select any words you want. For example, if you have a new product about to launch, you can mask the product name in meeting transcripts. In this case, you keep stakeholders up-to-date while keeping the product name secret until launch.

Vocabulary filtering has three display methods: `mask`, `remove`, and `tag`. Refer to the following examples to see how each works.

- **Mask**: Replaces specified words with three asterisks (***).

```
"transcript": "You can specify a list of *** or *** words, and *** *** removes them
 from transcripts automatically."
```

- **Remove**: Deletes specified words, leaving nothing in their place.

```
"transcript": "You can specify a list of or words, and removes them from transcripts
 automatically."
```

- **Tag**: Adds a tag (`"vocabularyFilterMatch": true`) to each specified word, but doesn't alter the word itself. Tagging allows for rapid transcript substitutions and edits.

```
"transcript": "You can specify a list of profane or offensive words, and amazon
 transcribe removes them from transcripts automatically."
...
    "alternatives": [
        {
            "confidence": "1.0",
            "content": "profane"
        }
    ],
    "type": "pronunciation",
    "vocabularyFilterMatch": true
```

When you submit a transcription request, you can specify a custom vocabulary filter and the filtration method you want to apply. Amazon Transcribe then modifies exact word matches when they appear in your transcript, according to the filtration method you specify.

Custom vocabulary filters can be applied to batch and streaming transcription requests. To learn how to create a custom vocabulary filter, see Creating a vocabulary filter. To learn how to apply your custom vocabulary filter, see Using a custom vocabulary filter.

> **Note**
>
> Amazon Transcribe automatically masks racially sensitive terms, though you can opt out of this default filter by contacting AWS Technical Support.

For a video walkthrough of vocabulary filtering, see Using vocabulary filters.

> **API operations specific to vocabulary filtering**
>
> `CreateVocabularyFilter`, `DeleteVocabularyFilter`, `GetVocabularyFilter`, `ListVocabularyFilters`, `UpdateVocabularyFilter`

Creating a vocabulary filter

There are two options for creating a custom vocabulary filter:

1. Save a list of line-separated words as a plain text file with UTF-8 encoding.
   - You can use this approach with the AWS Management Console, AWS CLI, or AWS SDKs.
   - If using the AWS Management Console, you can provide a local path or an Amazon S3 URI for your custom vocabulary file.
   - If using the AWS CLI or AWS SDKs, you must upload your custom vocabulary file to an Amazon S3 bucket and include the Amazon S3 URI in your request.
2. Include a list of comma-separated words directly in your API request.
   - You can use this approach with the AWS CLI or AWS SDKs using the `Words` parameter.

For examples of each method, refer to Creating custom vocabulary filters

Things to note when creating your custom vocabulary filter:

- Words aren't case sensitive. For example, "curse" and "CURSE" are treated the same.
- Only exact word matches are filtered. For example, if your filter includes "swear" but your media contains the word "swears" or "swearing", these are not filtered. Only instances of "swear" are filtered. You must therefore include all variations of the words you want filtered.
- Filters don't apply to words that are contained in other words. For example, if a custom vocabulary filter contains "marine" but not "submarine", "submarine" is not altered in the transcript.
- Each entry can only contain one word (no spaces).
- If you save your custom vocabulary filter as a text file, it must be in plain text format with UTF-8 encoding.
- You can have up to 100 custom vocabulary filters per AWS account and each can be up to 50 Kb in size.
- You can only use characters that are supported for your language. Refer to your language's character set for details.

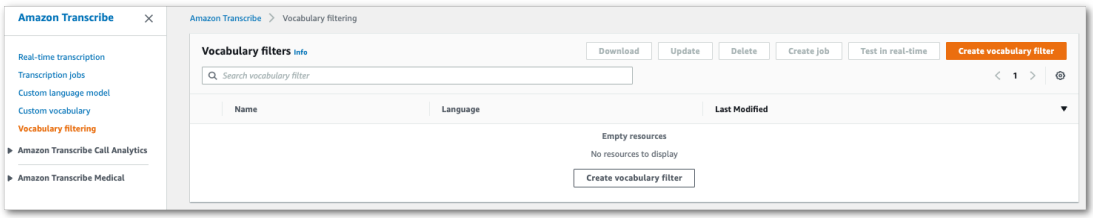
Creating custom vocabulary filters

To process a custom vocabulary filter for use with Amazon Transcribe, see the following examples:

AWS Management Console

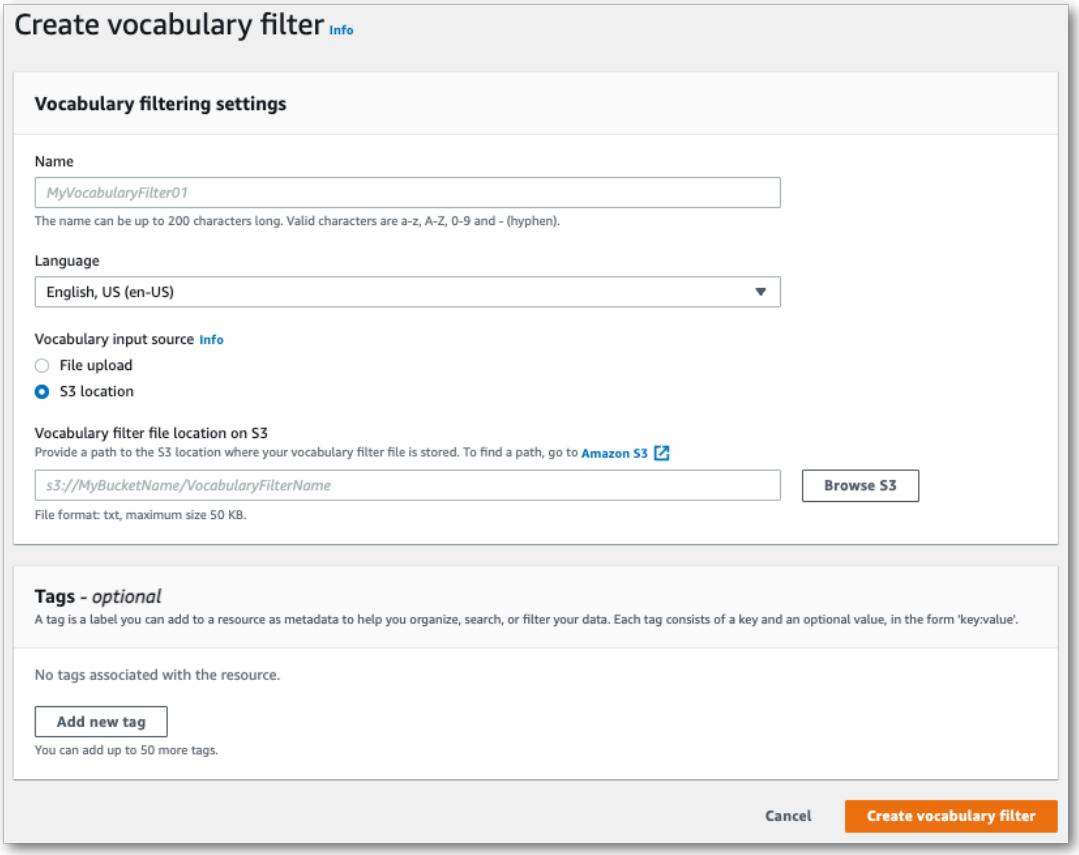
Before continuing, save your custom vocabulary filter as a text (*.txt) file. You can optionally upload your file to an Amazon S3 bucket.

1. Sign in to the AWS Management Console.
2. In the navigation pane, choose **Vocabulary filtering**. This opens the **Vocabulary filters** page where you can view existing custom vocabulary filters or create a new one.
3. Select **Create vocabulary filter**.



This takes you to the **Create vocabulary filter** page. Enter a name for your new custom vocabulary filter.

Select the **File upload** or **S3 location** option under **Vocabulary input source**. Then specify the location of your custom vocabulary file.



4. Optionally, add tags to your custom vocabulary filter. Once you have all fields completed, select **Create vocabulary filter** at the bottom of the page. If there are no errors processing your file, this takes you back to the **Vocabulary filters** page.

   Your custom vocabulary filter is now ready to use.

AWS CLI

This example uses the create-vocabulary-filter command to process a word list into a usable custom vocabulary filter. For more information, see `CreateVocabularyFilter`.

Option 1: You can include your list of words to your request using the `words` parameter.

```
aws transcribe create-vocabulary-filter \
--vocabulary-filter-name my-first-vocabulary-filter \
--language-code en-US \
```

```
--words profane,offensive,Amazon,Transcribe
```

Option 2: You can save your list of words as a text file and upload it to an Amazon S3 bucket, then include the file's URI in your request using the `vocabulary-filter-file-uri` parameter.

```
aws transcribe create-vocabulary-filter \
--vocabulary-filter-name my-first-vocabulary-filter \
--language-code en-US \
--vocabulary-filter-file-uri s3://amzn-s3-demo-bucket/my-vocabulary-filters/my-vocabulary-filter.txt
```

Here's another example using the create-vocabulary-filter command, and a request body that creates your custom vocabulary filter.

```
aws transcribe create-vocabulary-filter \
--cli-input-json file://filepath/my-first-vocab-filter.json
```

The file *my-first-vocab-filter.json* contains the following request body.

Option 1: You can include your list of words to your request using the `Words` parameter.

```
{
  "VocabularyFilterName": "my-first-vocabulary-filter",
  "LanguageCode": "en-US",
  "Words": [
        "profane","offensive","Amazon","Transcribe"
  ]
}
```

Option 2: You can save your list of words as a text file and upload it to an Amazon S3 bucket, then include the file's URI in your request using the `VocabularyFilterFileUri` parameter.

```
{
  "VocabularyFilterName": "my-first-vocabulary-filter",
  "LanguageCode": "en-US",
  "VocabularyFilterFileUri": "s3://amzn-s3-demo-bucket/my-vocabulary-filters/my-vocabulary-filter.txt"
}
```

> **Note**
>
> If you include `VocabularyFilterFileUri` in your request, you cannot use `Words`; you must choose one or the other.

AWS SDK for Python (Boto3)

This example uses the AWS SDK for Python (Boto3) to create a custom vocabulary filter using the create_vocabulary_filter method. For more information, see `CreateVocabularyFilter`.

For additional examples using the AWS SDKs, including feature-specific, scenario, and cross-service examples, refer to the Code examples for Amazon Transcribe using AWS SDKs chapter.

Option 1: You can include your list of words to your request using the `Words` parameter.

```
from __future__ import print_function
import time
import boto3
transcribe = boto3.client('transcribe', 'us-west-2')
vocab_name = "my-first-vocabulary-filter"
response = transcribe.create_vocabulary_filter(
    LanguageCode = 'en-US',
    VocabularyFilterName = vocab_name,
    Words = [
        'profane','offensive','Amazon','Transcribe'
    ]
)
```

Option 2: You can save your list of words as a text file and upload it to an Amazon S3 bucket, then include the file's URI in your request using the `VocabularyFilterFileUri` parameter.

```
from __future__ import print_function
import time
import boto3
transcribe = boto3.client('transcribe', 'us-west-2')
vocab_name = "my-first-vocabulary-filter"
response = transcribe.create_vocabulary_filter(
    LanguageCode = 'en-US',
    VocabularyFilterName = vocab_name,
    VocabularyFilterFileUri = 's3://amzn-s3-demo-bucket/my-vocabulary-filters/my-
vocabulary-filter.txt'
```

```
)
```

> **Note**
>
> If you include `VocabularyFilterFileUri` in your request, you cannot use `Words`; you must choose one or the other.

> **Note**
>
> If you create a new Amazon S3 bucket for your custom vocabulary filter files, make sure the IAM role making the `CreateVocabularyFilter` request has permissions to access this bucket. If the role doesn't have the correct permissions, your request fails. You can optionally specify an IAM role within your request by including the `DataAccessRoleArn` parameter. For more information on IAM roles and policies in Amazon Transcribe, see Amazon Transcribe identity-based policy examples.

> **Note**
>
> To encrypt your custom vocabulary filter artifacts at rest with a customer-managed KMS key, include an `EncryptionConfiguration` in your `CreateVocabularyFilter` request. For more information about encrypting resource artifacts with a customer-managed key, see Encrypting resource artifacts with a customer-managed key.

Using a custom vocabulary filter

Once your custom vocabulary filter is created, you can include it in your transcription requests; refer to the following sections for examples.

A custom vocabulary filter is available for use as soon as you create it.

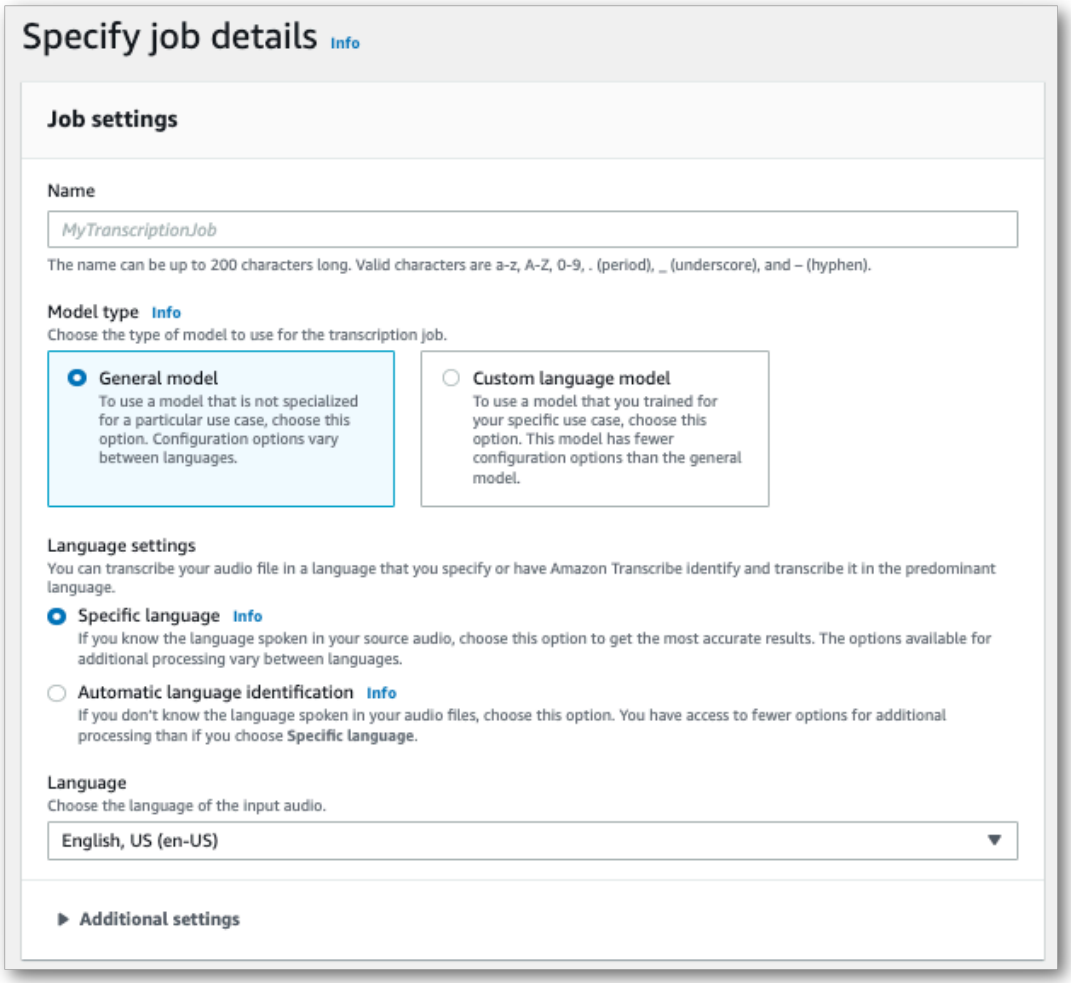
The language of the custom vocabulary filter you're including in your request must match the language code you specify for your media. If you use language identification and specify multiple language options, you can include one custom vocabulary filter per specified language. If the languages of your custom vocabulary filters don't match the language identified in your audio, your filters are not applied to your transcription and there are no warnings or errors.

Using a custom vocabulary filter in a batch transcription

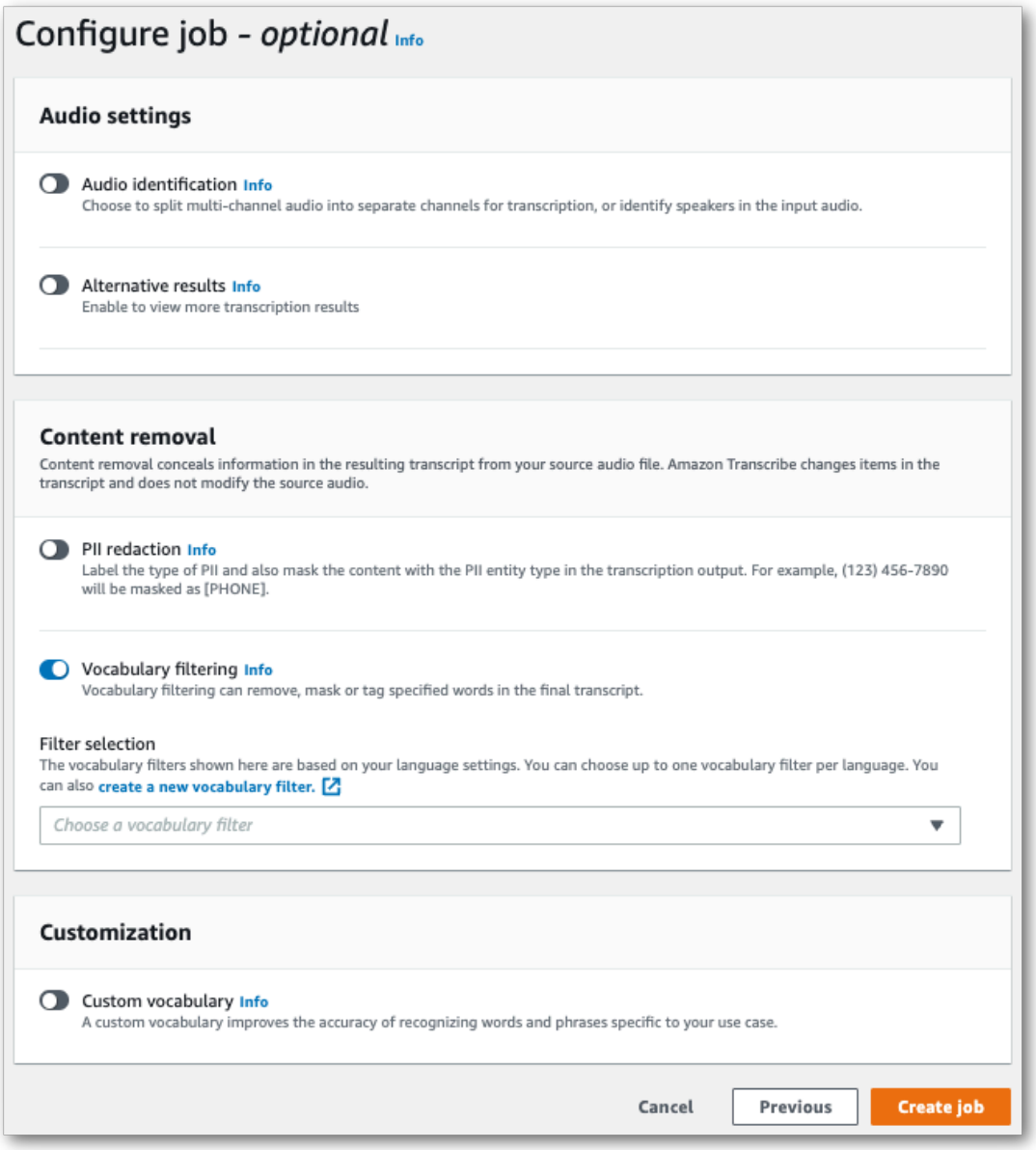
To use a custom vocabulary filter with a batch transcription, see the following for examples:

AWS Management Console

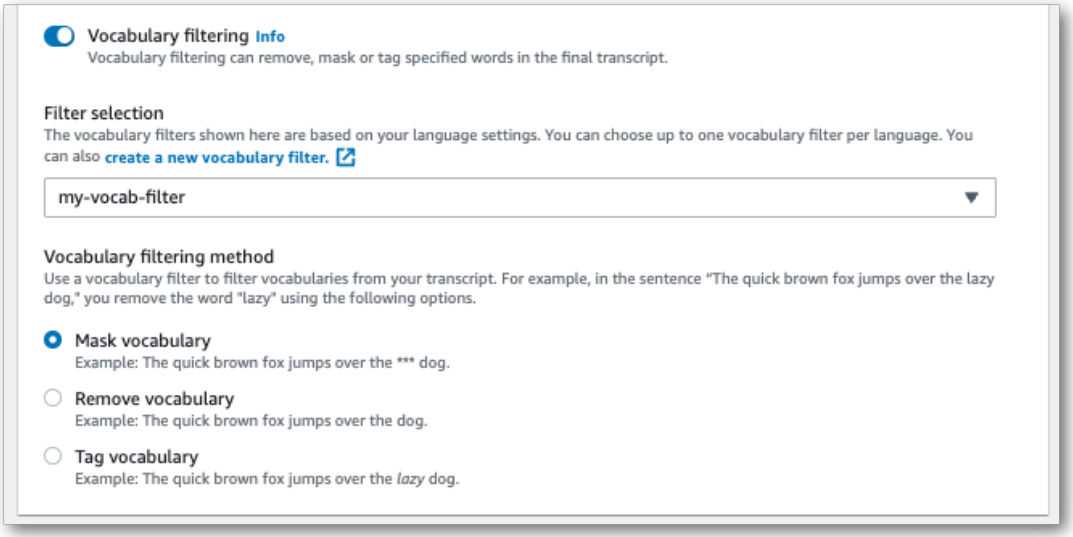
1. Sign in to the AWS Management Console.
2. In the navigation pane, choose **Transcription jobs**, then select **Create job** (top right). This opens the **Specify job details** page.



   Name your job and specify your input media. Optionally include any other fields, then choose **Next**.
3. On the **Configure job** page, in the **Content removal** panel, toggle on **Vocabulary filtering**.



4. Select your custom vocabulary filter from the dropdown menu and specify the filtration method.



5. Select **Create job** to run your transcription job.

AWS CLI

This example uses the start-transcription-job command and `Settings` parameter with the `VocabularyFilterName` and `VocabularyFilterMethod` sub-parameters. For more information, see `StartTranscriptionJob` and `Settings`.

```
aws transcribe start-transcription-job \
--region us-west-2 \
--transcription-job-name my-first-transcription-job \
--media MediaFileUri=s3://amzn-s3-demo-bucket/my-input-files/my-media-file.flac \
--output-bucket-name amzn-s3-demo-bucket \
--output-key my-output-files/ \
--language-code en-US \
--settings VocabularyFilterName=my-first-vocabulary-filter,VocabularyFilterMethod=mask
```

Here's another example using the start-transcription-job command, and a request body that includes your custom vocabulary filter with that job.

```
aws transcribe start-transcription-job \
--region us-west-2 \
--cli-input-json file://my-first-vocabulary-filter-job.json
```

The file *my-first-vocabulary-filter-job.json* contains the following request body.

```
{
  "TranscriptionJobName": "my-first-transcription-job",
  "Media": {
        "MediaFileUri": "s3://amzn-s3-demo-bucket/my-input-files/my-media-file.flac"
  },
  "OutputBucketName": "amzn-s3-demo-bucket",
  "OutputKey": "my-output-files/",
  "LanguageCode": "en-US",
  "Settings": {
        "VocabularyFilterName": "my-first-vocabulary-filter",
        "VocabularyFilterMethod": "mask"
   }
}
```

AWS SDK for Python (Boto3)

This example uses the AWS SDK for Python (Boto3) to include a custom vocabulary filter using the `Settings` argument for the start_transcription_job method. For more information, see `StartTranscriptionJob` and `Settings`.

For additional examples using the AWS SDKs, including feature-specific, scenario, and cross-service examples, refer to the Code examples for Amazon Transcribe using AWS SDKs chapter.

```
from __future__ import print_function
import time
import boto3
transcribe = boto3.client('transcribe', 'us-west-2')
job_name = "my-first-transcription-job"
job_uri = "s3://amzn-s3-demo-bucket/my-input-files/my-media-file.flac"
transcribe.start_transcription_job(
    TranscriptionJobName = job_name,
    Media = {
        'MediaFileUri': job_uri
    },
    OutputBucketName = 'amzn-s3-demo-bucket',
    OutputKey = 'my-output-files/',
    LanguageCode = 'en-US',
    Settings = {
        'VocabularyFilterName': 'my-first-vocabulary-filter',
        'VocabularyFilterMethod': 'mask'
   }
)
```

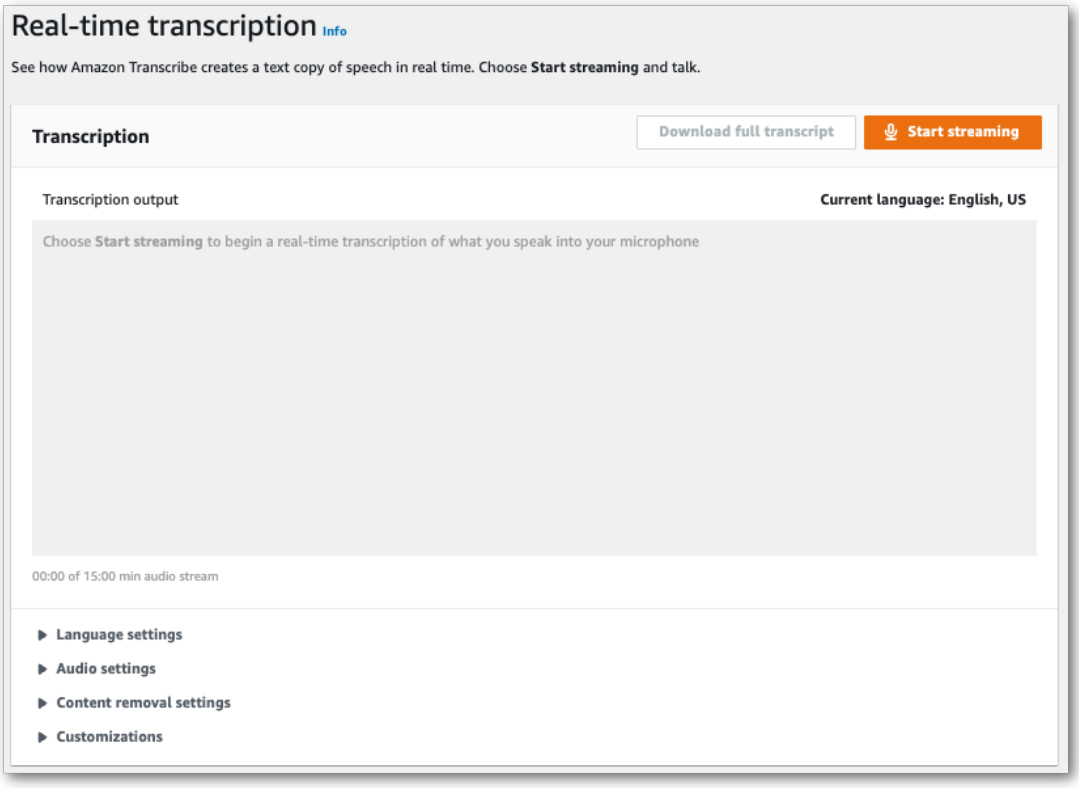
```
while True:
    status = transcribe.get_transcription_job(TranscriptionJobName = job_name)
    if status['TranscriptionJob']['TranscriptionJobStatus'] in ['COMPLETED', 'FAILED']:
        break
    print("Not ready yet...")
    time.sleep(5)
print(status)
```

Using a custom vocabulary filter in a streaming transcription

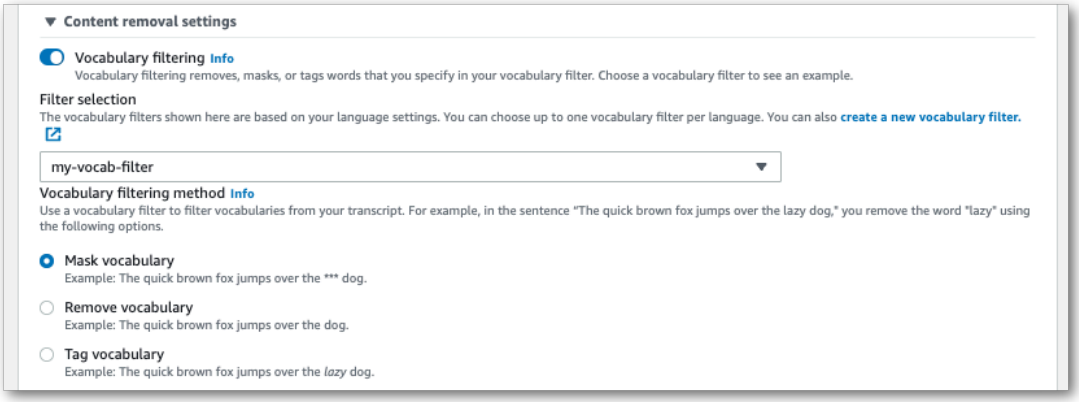
To use a custom vocabulary filter with a streaming transcription, see the following for examples:

AWS Management Console

1. Sign into the AWS Management Console.
2. In the navigation pane, choose **Real-time transcription**. Scroll down to **Content removal settings** and expand this field if it is minimized.



3. Toggle on **Vocabulary filtering**. Select a custom vocabulary filter from the dropdown menu and specify the filtration method.



Include any other settings you want to apply to your stream.

4. You're now ready to transcribe your stream. Select **Start streaming** and begin speaking. To end your dictation, select **Stop streaming**.

HTTP/2 stream

This example creates an HTTP/2 request that includes your custom vocabulary filter and filter method. For more information on using HTTP/2 streaming with Amazon Transcribe, see Setting up an HTTP/2 stream. For more detail on parameters and headers specific to Amazon Transcribe, see `StartStreamTranscription`.

```
POST /stream-transcription HTTP/2
host: transcribestreaming.us-west-2.amazonaws.com
X-Amz-Target: com.amazonaws.transcribe.Transcribe.StartStreamTranscription
Content-Type: application/vnd.amazon.eventstream
X-Amz-Content-Sha256: string
X-Amz-Date: 20220208T235959Z
Authorization: AWS4-HMAC-SHA256 Credential=access-key/20220208/us-west-2/transcribe/
aws4_request, SignedHeaders=content-type;host;x-amz-content-sha256;x-amz-date;x-amz-
target;x-amz-security-token, Signature=string
x-amzn-transcribe-language-code: en-US
x-amzn-transcribe-media-encoding: flac
x-amzn-transcribe-sample-rate: 16000
x-amzn-transcribe-vocabulary-filter-name: my-first-vocabulary-filter
x-amzn-transcribe-vocabulary-filter-method: mask
transfer-encoding: chunked
```

Parameter definitions can be found in the API Reference; parameters common to all AWS API operations are listed in the Common Parameters section.

WebSocket stream

This example creates a presigned URL that applies your custom vocabulary filter to a WebSocket stream. Line breaks have been added for readability. For more information on using WebSocket streams with Amazon Transcribe, see Setting up a WebSocket stream. For more detail on parameters, see StartStreamTranscription.

```
GET wss://transcribestreaming.us-west-2.amazonaws.com:8443/stream-transcription-
websocket?
&X-Amz-Algorithm=AWS4-HMAC-SHA256
&X-Amz-Credential=AKIAIOSFODNN7EXAMPLE%2F20220208%2Fus-
west-2%2Ftranscribe%2Faws4_request
&X-Amz-Date=20220208T235959Z
&X-Amz-Expires=300
&X-Amz-Security-Token=security-token
&X-Amz-Signature=string
&X-Amz-SignedHeaders=content-type%3Bhost%3Bx-amz-date
&language-code=en-US
&media-encoding=flac
&sample-rate=16000
&vocabulary-filter-name=my-first-vocabulary-filter
&vocabulary-filter-method=mask
```

Parameter definitions can be found in the API Reference; parameters common to all AWS API operations are listed in the Common Parameters section.

Updating a custom vocabulary filter

To update the word list in an existing custom vocabulary filter, use the UpdateVocabularyFilter operation.

When updating a custom vocabulary filter, note the following:

- The update replaces the existing filter with the new content you provide.
- You must include either `Words` or `VocabularyFilterFileUri` in your request.
- (Optional) Include an `EncryptionConfiguration` to encrypt your vocabulary filter with a customer-managed key. You must also provide the new filter content, because encryption cannot be changed independently. For more information about encrypting resource artifacts with a customer-managed key, see Encrypting resource artifacts with a customer-managed key.

To update a custom vocabulary filter, see the following for examples:

AWS CLI

This example uses the update-vocabulary-filter command. For more information, see UpdateVocabularyFilter.

```
aws transcribe update-vocabulary-filter \
--vocabulary-filter-name my-first-vocabulary-filter \
--vocabulary-filter-file-uri s3://amzn-s3-demo-bucket/my-vocabulary-filters/my-updated-vocabulary-filter.txt
```

Here's another example using the update-vocabulary-filter command, and a request body that updates your custom vocabulary filter.

```
aws transcribe update-vocabulary-filter \
--cli-input-json file://filepath/my-updated-vocab-filter.json
```

The file *my-updated-vocab-filter.json* contains the following request body.

```
{
  "VocabularyFilterName": "my-first-vocabulary-filter",
  "VocabularyFilterFileUri": "s3://amzn-s3-demo-bucket/my-vocabulary-filters/my-updated-vocabulary-filter.txt"
}
```

Detecting toxic speech

Toxic speech detection is designed to help moderate social media platforms that involve peer-to-peer dialogue, such as online gaming and social chat platforms. The use of toxic speech can be deeply detrimental to individuals, peer groups, and communities. Flagging harmful language helps organizations keep conversations civil and maintain a safe and inclusive online environment for users to create, share and participate freely.

Amazon Transcribe Toxicity Detection leverages both audio and text-based cues to identify and classify voice-based toxic content across seven categories including sexual harassment, hate speech, threat, abuse, profanity, insult, and graphic. In addition to text, Amazon Transcribe Toxicity Detection uses speech cues, such as tones and pitch to hone in on toxic intent in speech. This is an improvement from standard content moderation systems that are designed to focus only on specific terms, without accounting for intention.

Amazon Transcribe flags and categorizes toxic speech, which minimizes the volume of data that must be manually processed. This enables content moderators to quickly and efficiently manage the discourse on their platforms.

Toxic speech categories include:

- **Profanity**: Speech that contains words, phrases, or acronyms that are impolite, vulgar, or offensive.
- **Hate speech**: Speech that criticizes, insults, denounces, or dehumanizes a person or group on the basis of an identity (such as race, ethnicity, gender, religion, sexual orientation, ability, and national origin).
- **Sexual**: Speech that indicates sexual interest, activity, or arousal using direct or indirect references to body parts, physical traits, or sex.
- **Insults**: Speech that includes demeaning, humiliating, mocking, insulting, or belittling language. This type of language is also labeled as bullying.
- **Violence or threat**: Speech that includes threats seeking to inflict pain, injury, or hostility toward a person or group.
- **Graphic**: Speech that uses visually descriptive and unpleasantly vivid imagery. This type of language is often intentionally verbose to amplify a recipient's discomfort.
- **Harassment or abusive**: Speech intended to affect the psychological well-being of the recipient, including demeaning and objectifying terms. This type of language is also labeled as harassment.

Toxicity detection analyzes speech segments (the speech between natural pauses) and assigns confidence scores to these segments. Confidence scores are values between 0 and 1. A larger confidence score indicates a greater likelihood that the content is toxic speech in the associated category. You can use these confidence scores to set the appropriate threshold of toxicity detection for your use case.

> **Note**
>
> Toxicity detection is only available for batch transcriptions in US English (`en-US`).

View example output in JSON format.

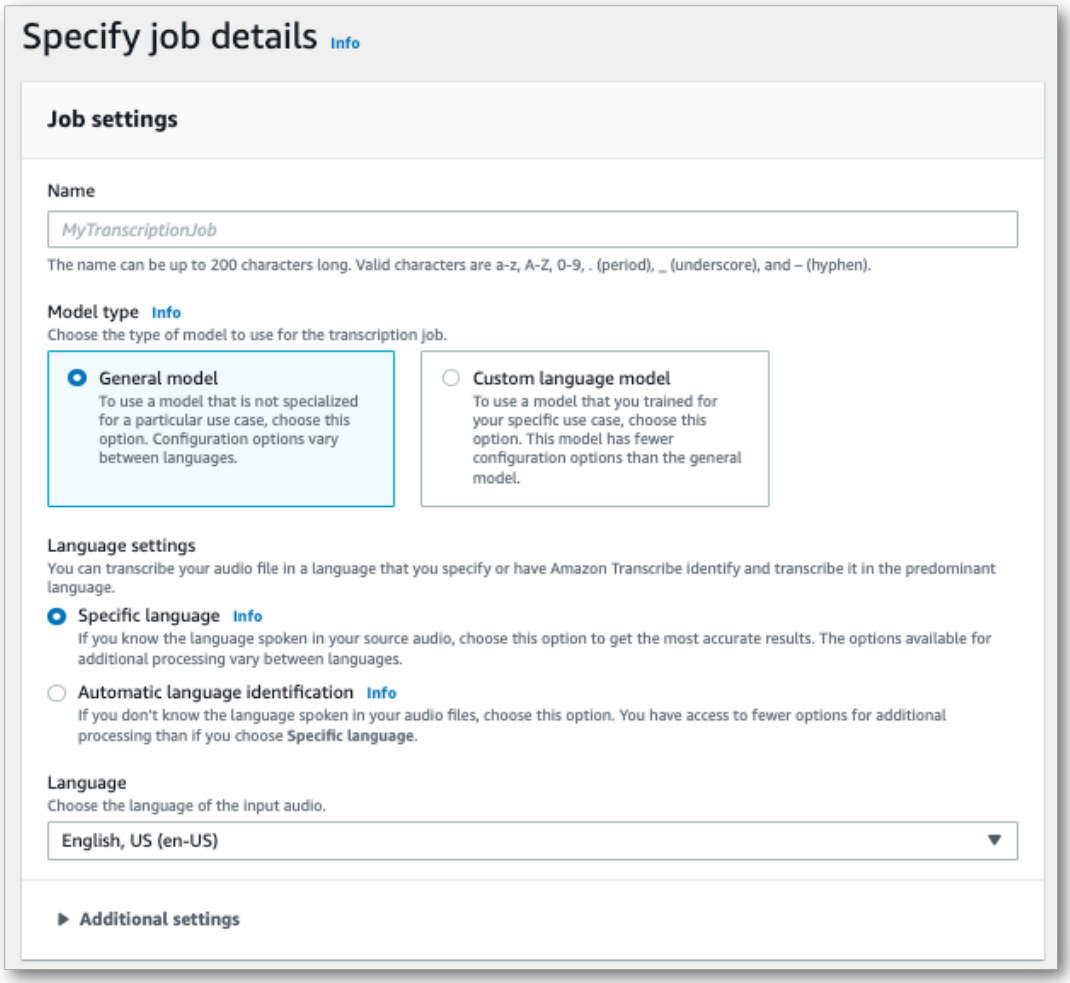
Using toxic speech detection

Using toxic speech detection in a batch transcription

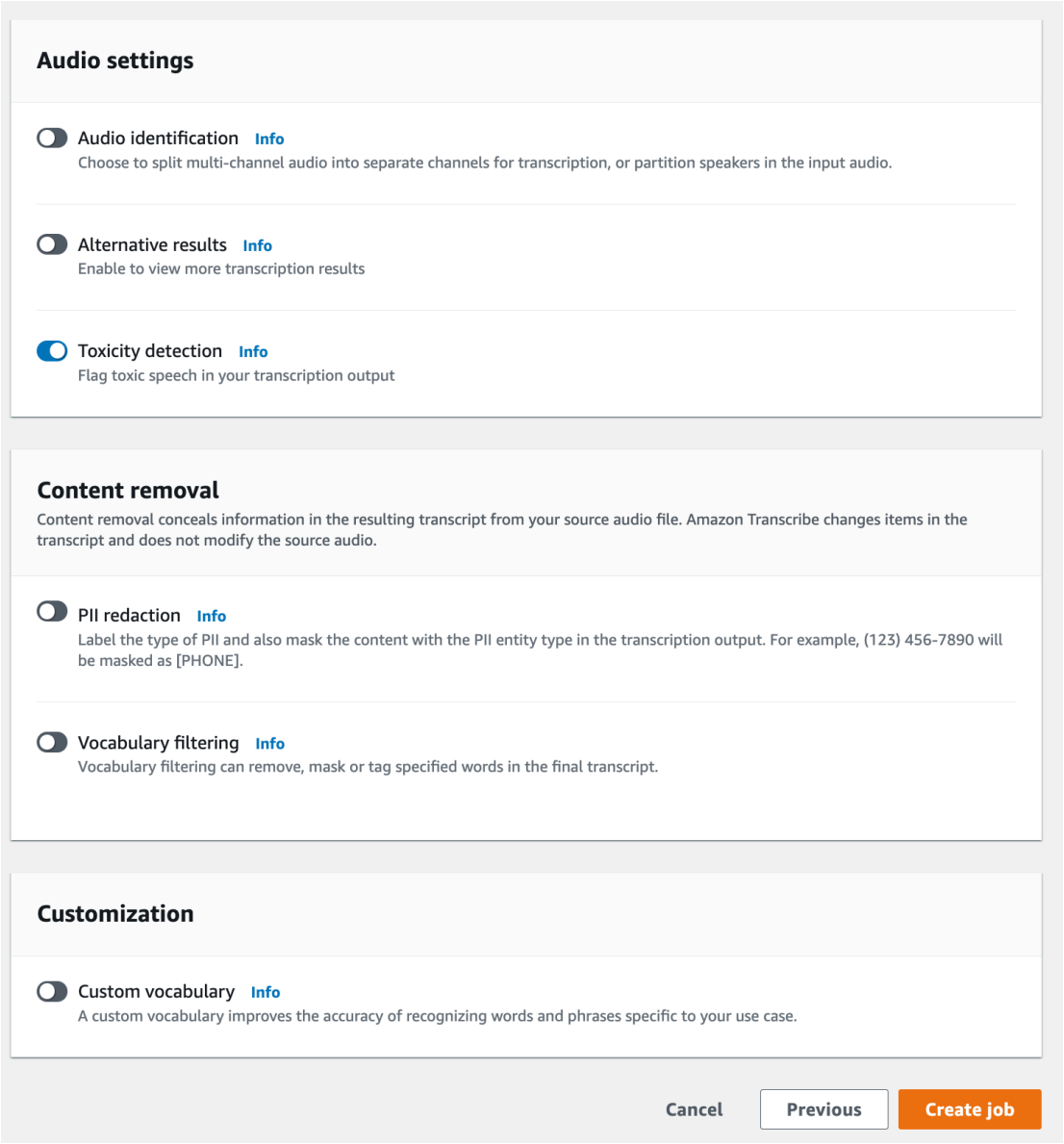
To use toxic speech detection with a batch transcription, see the following for examples:

AWS Management Console

1. Sign in to the AWS Management Console.
2. In the navigation pane, choose **Transcription jobs**, then select **Create job** (top right). This opens the **Specify job details** page.



3. On the **Specify job details** page, you can also enable PII redaction if you want. Note that the other listed options are not supported with Toxicity detection. Select **Next**. This takes you to the **Configure job - optional** page. In the **Audio settings** panel, select **Toxicity detection**.



4. Select **Create job** to run your transcription job.
5. Once your transcription job is complete, you can download your transcript from the **Download** drop-down menu in the transcription job's detail page.

AWS CLI

This example uses the start-transcription-job command and `ToxicityDetection` parameter. For more information, see `StartTranscriptionJob` and `ToxicityDetection`.

```
aws transcribe start-transcription-job \
--region us-west-2 \
```

```
--transcription-job-name my-first-transcription-job \
--media MediaFileUri=s3://amzn-s3-demo-bucket/my-input-files/my-media-file.flac \
--output-bucket-name amzn-s3-demo-bucket \
--output-key my-output-files/ \
--language-code en-US \
--toxicity-detection ToxicityCategories=ALL
```

Here's another example using the start-transcription-job command, and a request body that includes toxicity detection.

```
aws transcribe start-transcription-job \
--region us-west-2 \
--cli-input-json file://filepath/my-first-toxicity-job.json
```

The file *my-first-toxicity-job.json* contains the following request body.

```
{
  "TranscriptionJobName": "my-first-transcription-job",
  "Media": {
        "MediaFileUri": "s3://amzn-s3-demo-bucket/my-input-files/my-media-file.flac"
  },
  "OutputBucketName": "amzn-s3-demo-bucket",
  "OutputKey": "my-output-files/",
  "LanguageCode": "en-US",
  "ToxicityDetection": [
      {
         "ToxicityCategories": [ "ALL" ]
      }
   ]
}
```

AWS SDK for Python (Boto3)

This example uses the AWS SDK for Python (Boto3) to enable `ToxicityDetection` for the start_transcription_job method. For more information, see `StartTranscriptionJob` and `ToxicityDetection`.

For additional examples using the AWS SDKs, including feature-specific, scenario, and cross-service examples, refer to the Code examples for Amazon Transcribe using AWS SDKs chapter.

```
from __future__ import print_function
import time
import boto3
transcribe = boto3.client('transcribe', 'us-west-2')
job_name = "my-first-transcription-job"
job_uri = "s3://amzn-s3-demo-bucket/my-input-files/my-media-file.flac"
transcribe.start_transcription_job(
    TranscriptionJobName = job_name,
    Media = {
        'MediaFileUri': job_uri
    },
    OutputBucketName = 'amzn-s3-demo-bucket',
    OutputKey = 'my-output-files/',
    LanguageCode = 'en-US',
    ToxicityDetection = [
        {
            'ToxicityCategories': ['ALL']
        }
    ]
)

while True:
    status = transcribe.get_transcription_job(TranscriptionJobName = job_name)
    if status['TranscriptionJob']['TranscriptionJobStatus'] in ['COMPLETED', 'FAILED']:
        break
    print("Not ready yet...")
    time.sleep(5)
print(status)
```

Example output

Toxic speech is tagged and categorized in your transcription output. Each instance of toxic speech is categorized and assigned a confidence score (a value between 0 and 1). A larger confidence value indicates a greater likelihood that the content is toxic speech within the specified category.

Example output (JSON)

The following is an example output in JSON format showing categorized toxic speech with associated confidence scores.


```
{
    "jobName": "my-toxicity-job",
    "accountId": "111122223333",
    "results": {
        "transcripts": [...],
        "items":[...],
        "toxicity_detection": [
            {
                "text": "What the * are you doing man? That's why I didn't want to play
 with your * .  man it was a no, no I'm not calming down * man. I well I spent I spent
 too much * money on this game.",
                "toxicity": 0.7638,
                "categories": {
                    "profanity": 0.9913,
                    "hate_speech": 0.0382,
                    "sexual": 0.0016,
                    "insult": 0.6572,
                    "violence_or_threat": 0.0024,
                    "graphic": 0.0013,
                    "harassment_or_abuse": 0.0249
                },
                "start_time": 8.92,
                "end_time": 21.45
            },
            Items removed for brevity
            {
                "text": "What? Who? What the * did you just say to me? What's your
 address? What is your * address? I will pull up right now on your * * man. Take your *
 back to , tired of this **.",
                "toxicity": 0.9816,
                "categories": {
                    "profanity": 0.9865,
                    "hate_speech": 0.9123,
                    "sexual": 0.0037,
                    "insult": 0.5447,
                    "violence_or_threat": 0.5078,
                    "graphic": 0.0037,
                    "harassment_or_abuse": 0.0613
```

```
                },
                "start_time": 43.459,
                "end_time": 54.639
            },
        ]
    },
    ...
    "status": "COMPLETED"
}
```

Redacting or identifying personally identifiable information

Redaction is used to mask or remove sensitive content, in the form of personally identifiable information (PII), from your transcripts. The types of PII Amazon Transcribe can redact varies between batch and streaming transcriptions. To view the PII list for each transcription method, refer to Redacting PII in your batch job and Redacting or identifying PII in a real-time stream. With streaming transcriptions, you also have the option to flag PII without redacting it; refer to Example PII identification output for an output example.

When redaction is enabled, you have the option to generate only a redacted transcript or both a redacted transcript and an unredacted transcript. If you choose to generate only a redacted transcript, note that your media is the only place where the complete conversation is stored. If you delete your original media, there is no record of the unredacted PII. Because of this, it may be prudent to generate an unredacted transcript in addition to a redacted one.

To learn more about PII redaction with batch transcriptions, refer to: Redacting PII in your batch job.

To learn more about PII redaction or identification with streaming transcriptions, refer to: Redacting or identifying PII in a real-time stream.

Important

The redaction feature is designed to identify and remove sensitive data. However, due to the predictive nature of machine learning, Amazon Transcribe may not identify and remove all instances of sensitive data in your transcript. We strongly recommend that you review any redacted output to ensure it meets your needs.

The redaction feature does not meet the requirements for de-identification under medical privacy laws, such as the U.S. Health Insurance Portability and Accountability Act of 1996 (HIPAA).

For a video walkthrough of the Amazon Transcribe's redaction feature, see Use content redaction to identify & redact PII.

Redacting PII in your batch job

When redacting personally identifiable information (PII) from a transcript during a batch transcription job, Amazon Transcribe replaces each identified instance of PII with `[PII]` in the main text body of your transcript. You can also view the type of PII that is redacted in the word-for-word portion of the transcription output. For an output sample, see Example redacted output (batch).

Redaction with batch transcriptions is available with English dialects: US (`en-US`), Australia (`en-AU`), British (`en-GB`), India (`en-IN`), Ireland (`en-IE`), New Zealand (`en-NZ`), Scotland (`en-AB`), South Africa (`en-ZA`), Wales (`en-WL`); Spanish dialect: US (`es-US`); French dialects: France (`fr-FR`), Canada (`fr-CA`); German dialects: Germany (`de-DE`), Swiss (`de-CH`); Italian dialect: Italy (`it-IT`); and Portuguese dialects: Portugal (`pt-PT`), Brazil (`pt-BR`). Redaction is not compatible with language identification.

Both redacted and unredacted transcripts are stored in the same output Amazon S3 bucket. Amazon Transcribe stores them in a bucket you specify or in the default Amazon S3 bucket managed by the service.

Types of PII Amazon Transcribe can recognize for batch transcriptions

PII type	Description
`ADDRESS`	A physical address, such as *100 Main Street, Anytown, USA* or *Suite #12, Building 123*. An address can include a street, building, location, city, state, country, county, zip, precinct, neighborhood, and more.
`AGE`	An individual's age, including the quantity and unit of time. For example, in the phrase "I am 40 years old," Amazon Transcribe recognizes "40 years" as an age.
`ALL`	Redact or identify all PII types listed in this table.
`AWS_ACCESS_KEY`	A unique identifier that's associated with a secret access key; you use the access key ID

PII type	Description
	and secret access key to sign programmatic AWS requests cryptographically.
AWS_SECRET_KEY	A unique identifier that's associated with an access key. You use the access key ID and secret access key to sign programmatic AWS requests cryptographically.
BANK_ACCOUNT_NUMBER	A US bank account number. These are typically between 10 - 12 digits long, but Amazon Transcribe also recognizes bank account numbers when only the last 4 digits are present.
BANK_ROUTING	A US bank account routing number. These are typically 9 digits long, but Amazon Transcrib e also recognizes routing numbers when only the last 4 digits are present.
CA_HEALTH_NUMBER	A Canadian Health Service Number is a 10-digit unique identifier, required for individuals to access healthcare benefits.
CA_SOCIAL_INSURANCE_NUMBER	A Canadian Social Insurance Number (SIN) is a 9-digit unique identifier, required for individuals to access government programs and benefits.
CREDIT_DEBIT_CVV	A 3-digit card verification code (CVV) that is present on VISA, MasterCard, and Discover credit and debit cards. In American Express credit or debit cards, it is a 4-digit numeric code.

PII type	Description
CREDIT_DEBIT_EXPIRY	The expiration date for a credit or debit card. This number is usually 4 digits long and formatted as month/year or MM/YY. For example, Amazon Transcribe can recognize expiration dates such as *01/21*, *01/2021*, and *Jan 2021*.
CREDIT_DEBIT_NUMBER	The number for a credit or debit card. These numbers can vary from 13 to 16 digits in length, but Amazon Transcribe also recognize s credit or debit card numbers when only the last 4 digits are present.
DATE_TIME	A date can include a year, month, day, day of week, or time of day. For example, Amazon Transcribe recognizes "January 19, 2020" or "11 am" as dates. Amazon Transcribe will recognize partial dates, date ranges, and date intervals. It will also recognize decades, such as "the 1990s".
DRIVER_ID	The number assigned to a driver's license, which is an official document permitting an individual to operate one or more motorized vehicles on a public road. A driver's license number consists of alphanumeric characters.
EMAIL	An email address, such as *efua.owusu@email.com*.
INTERNATIONAL_BANK_ACCOUNT_NUMBER	An International Bank Account Number has specific formats in each country. For more information, see *www.iban.com/structure*.
IP_ADDRESS	An IPv4 address, such as *198.51.100.0*.

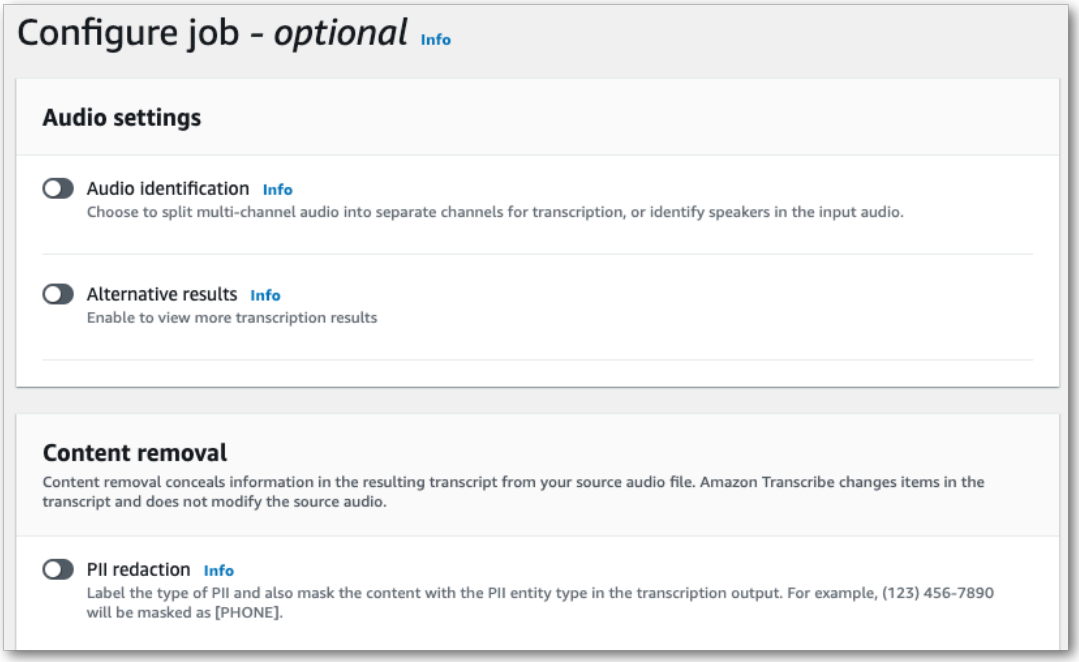
PII type	Description
LICENSE_PLATE	A license plate for a vehicle is issued by the state or country where the vehicle is registered. The format for passenger vehicles is typically five to eight digits, consisting of upper-case letters and numbers. The format varies depending on the location of the issuing state or country.
MAC_ADDRESS	A media access control (MAC) address is a unique identifier assigned to a network interface controller (NIC).
NAME	An individual's name. This entity type does not include titles, such as Mr., Mrs., Miss, or Dr. Amazon Transcribe does not apply this entity type to names that are part of organizations or addresses. For example, Amazon Transcrib e recognizes the *John Doe Organization* as an organization, and *Jane Doe Street* as an address.
PASSPORT_NUMBER	A unique identifier assigned to an individua l's passport. The format typically includes a combination of letters and numbers and varies by country.
PASSWORD	An alphanumeric string that is used as a password, such as "*very20special#pass*".
PHONE	A phone number. This entity type also includes fax and pager numbers.
PIN	A 4-digit personal identification number (PIN) that allows someone to access their bank account information.

PII type	Description
SSN	A Social Security Number (SSN) is a 9-digit number that is issued to US citizens, permanent residents, and temporary working residents. Amazon Transcribe also recognizes Social Security Numbers when only the last 4 digits are present.
SWIFT_CODE	A SWIFT code is a standard format of Bank Identifier Code (BIC) used to specify a particular bank or branch. Banks use these codes for money transfers such as internati onal wire transfers. SWIFT codes consist of 8 or 11 characters. The 11-digit codes refer to specific branches, while 8-digit codes (or 11-digit codes ending in 'XXX') refer to the head or primary office.
URL	A web address, such as *www.example.com*.
US_INDIVIDUAL_TAX_IDENTIFIC ATION_NUMBER	A US Individual Taxpayer Identification Number (ITIN) is a 9-digit number that starts with a "9" and contains a "7" or "8" as the fourth digit. An ITIN can be formatted with a space or a dash after the third and fourth digits.
USERNAME	A user name that identifies an account, such as a login name, screen name, nick name, or handle.
VEHICLE_IDENTIFICATION_NUMBER	A Vehicle Identification Number (VIN) uniquely identifies a vehicle. VIN content and format are defined in the ISO 3779 specification. Each country has specific codes and formats for VINs.

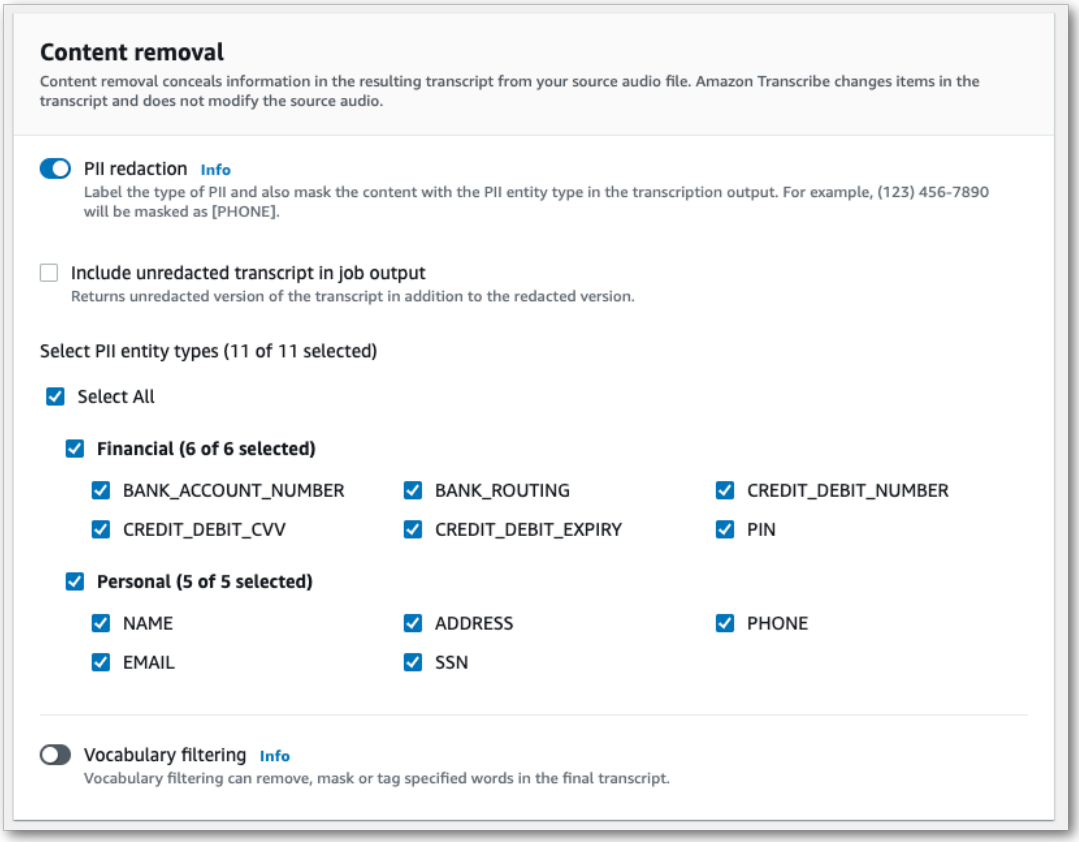
You can start a batch transcription job using the AWS Management Console, AWS CLI, or AWS SDK.

AWS Management Console

1. Sign in to the AWS Management Console.
2. In the navigation pane, choose **Transcription jobs**, then select **Create job** (top right). This will open the **Specify job details** page.
3. After filling in your desired fields on the **Specify job details** page, select **Next** to go to the **Configure job - *optional*** page. Here you'll find the **Content removal** panel with the **PII redaction** toggle.



4. Once you select **PII redaction**, you have the option to select all PII types you want to redact. You can also choose to have an unredacted transcript if you select **Include unredacted transcript in job output** box.



5. Select **Create job** to run your transcription job.

AWS CLI

This example uses the start-transcription-job command and `content-redaction` parameter. For more information, see `StartTranscriptionJob` and `ContentRedaction`.

```
aws transcribe start-transcription-job \
--region us-west-2 \
--transcription-job-name my-first-transcription-job \
--media MediaFileUri=s3://amzn-s3-demo-bucket/my-input-files/my-media-file.flac \
--output-bucket-name amzn-s3-demo-bucket \
--output-key my-output-files/ \
--language-code en-US \
--content-redaction
 RedactionType=PII,RedactionOutput=redacted,PiiEntityTypes=NAME,ADDRESS,BANK_ACCOUNT_NUMBER
```

Here's another example using the start-transcription-job method, and the request body redacts PII for that job.

```
aws transcribe start-transcription-job \
--region us-west-2 \
--cli-input-json file://filepath/my-first-redaction-job.json
```

The file *my-first-redaction-job.json* contains the following request body.

```
{
  "TranscriptionJobName": "my-first-transcription-job",
  "Media": {
      "MediaFileUri":  "s3://amzn-s3-demo-bucket/my-input-files/my-media-file.flac"
  },
  "OutputBucketName": "amzn-s3-demo-bucket",
  "OutputKey": "my-output-files/",
  "LanguageCode": "en-US",
  "ContentRedaction": {
      "RedactionOutput":"redacted",
      "RedactionType":"PII",
      "PiiEntityTypes": [
           "NAME",
           "ADDRESS",
           "BANK_ACCOUNT_NUMBER"
      ]
  }
}
```

AWS SDK for Python (Boto3)

This example uses the AWS SDK for Python (Boto3) to redact content using the `ContentRedaction` argument for the start_transcription_job method. For more information, see `StartTranscriptionJob` and `ContentRedaction`.

For additional examples using the AWS SDKs, including feature-specific, scenario, and cross-service examples, refer to the Code examples for Amazon Transcribe using AWS SDKs chapter.

```
from __future__ import print_function
import time
import boto3
transcribe = boto3.client('transcribe', 'us-west-2')
job_name = "my-first-transcription-job"
job_uri = "s3://amzn-s3-demo-bucket/my-input-files/my-media-file.flac"
transcribe.start_transcription_job(
    TranscriptionJobName = job_name,
```

```
    Media = {
        'MediaFileUri': job_uri
    },
    OutputBucketName = 'amzn-s3-demo-bucket',
    OutputKey = 'my-output-files/',
    LanguageCode = 'en-US',
    ContentRedaction = {
        'RedactionOutput':'redacted',
        'RedactionType':'PII',
        'PiiEntityTypes': [
            'NAME','ADDRESS','BANK_ACCOUNT_NUMBER'
        ]
    }
)

while True:
    status = transcribe.get_transcription_job(TranscriptionJobName = job_name)
    if status['TranscriptionJob']['TranscriptionJobStatus'] in ['COMPLETED', 'FAILED']:
        break
    print("Not ready yet...")
    time.sleep(5)
print(status)
```

Note

PII redaction for batch jobs is only supported in these AWS Regions: Asia Pacific (Hong Kong), Asia Pacific (Mumbai), Asia Pacific (Seoul), Asia Pacific (Singapore), Asia Pacific (Sydney), Asia Pacific (Tokyo), GovCloud (US-West), Canada (Central), EU (Frankfurt), EU (Ireland), EU (London), EU (Paris), Middle East (Bahrain), South America (Sao Paulo), US East (N. Virginia), US East (Ohio), US West (Oregon), and US West (N. California).

Redacting or identifying PII in a real-time stream

When redacting personally identifiable information (PII) from a streaming transcription, Amazon Transcribe replaces each identified instance of PII with `[PII]` in your transcript.

An additional option available for streaming transcriptions is *PII identification*. When you activate PII Identification, Amazon Transcribe labels the PII in your transcription results under an `Entities` object. For an output sample, see Example redacted streaming output and Example PII identification output.

Redaction and identification of PII with streaming transcriptions is available with these English dialects: Scottish (en-AB), Australia (en-AU), Canada (en-CA), British (en-GB), Ireland (en-IE), India (en-IN), New Zealand (en-NZ), United States (en-US), Wales (en-WL), and South Africa (en-ZA); Spanish dialects: US (es-US), Spain (es-ES); French dialects: French (fr-FR), Canada (fr-CA); Portuguese dialects: Portugal (pt-PT), Brazil (pt-BR); Italian dialect: Italy (it-IT); and German dialects: Germany (de-DE), Swiss (de-CH).

PII identification and redaction for streaming jobs is performed only upon complete transcription of the audio segments.

Types of PII Amazon Transcribe can recognize for streaming transcriptions

PII type	Description
ADDRESS	A physical address, such as *100 Main Street, Anytown, USA* or *Suite #12, Building 123*. An address can include a street, building, location, city, state, country, county, zip, precinct, neighborhood, and more.
ALL	Redact or identify all PII types listed in this table.
BANK_ACCOUNT_NUMBER	A US bank account number. These are typically between 10 - 12 digits long, but Amazon Transcribe also recognizes bank account numbers when only the last 4 digits are present.
BANK_ROUTING	A US bank account routing number. These are typically 9 digits long, but Amazon Transcrib e also recognizes routing numbers when only the last 4 digits are present.
CREDIT_DEBIT_CVV	A 3-digit card verification code (CVV) that is present on VISA, MasterCard, and Discover credit and debit cards. In American Express credit or debit cards, it is a 4-digit numeric code.

PII type	Description
CREDIT_DEBIT_EXPIRY	The expiration date for a credit or debit card. This number is usually 4 digits long and formatted as month/year or MM/YY. For example, Amazon Transcribe can recognize expiration dates such as *01/21*, *01/2021*, and *Jan 2021*.
CREDIT_DEBIT_NUMBER	The number for a credit or debit card. These numbers can vary from 13 to 16 digits in length, but Amazon Transcribe also recognize s credit or debit card numbers when only the last 4 digits are present.
EMAIL	An email address, such as *efua.owusu@email.com*.
NAME	An individual's name. This entity type does not include titles, such as Mr., Mrs., Miss, or Dr. Amazon Transcribe does not apply this entity type to names that are part of organizations or addresses. For example, Amazon Transcrib e recognizes the *John Doe Organization* as an organization, and *Jane Doe Street* as an address.
PHONE	A phone number. This entity type also includes fax and pager numbers.
PIN	A 4-digit personal identification number (PIN) that allows someone to access their bank account information.

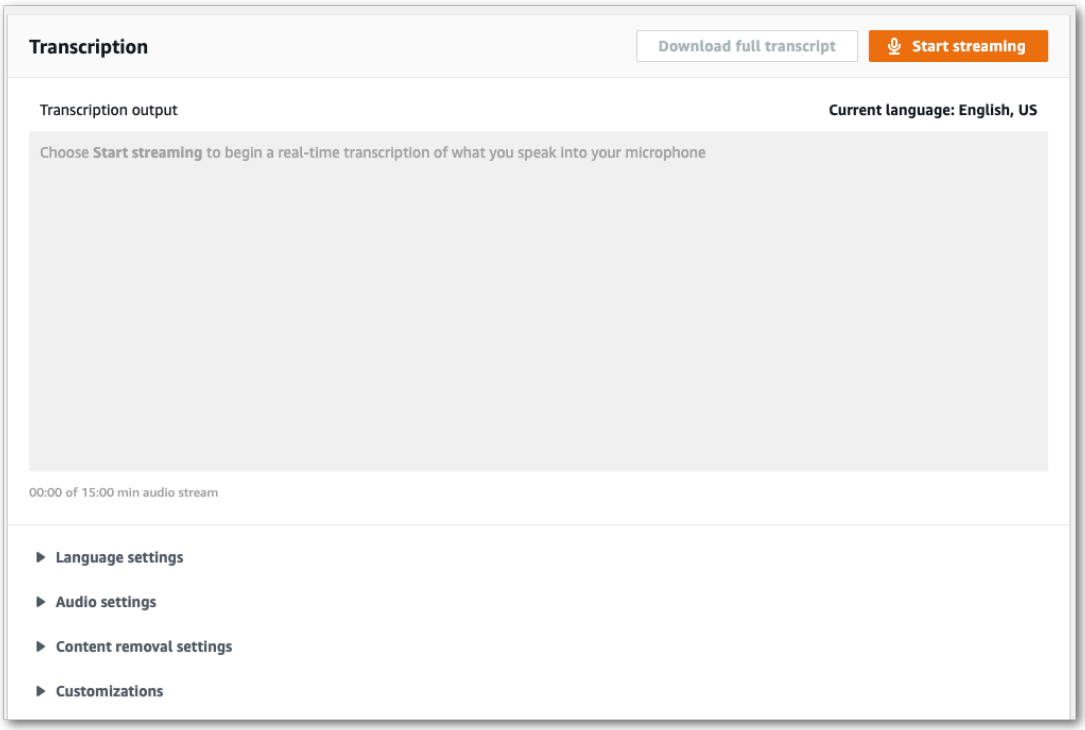
PII type	Description
SSN	A Social Security Number (SSN) is a 9-digit number that is issued to US citizens, permanent residents, and temporary working residents. Amazon Transcribe also recognizes Social Security Numbers when only the last 4 digits are present.
AGE	An individual's age, including the quantity and unit of time. For example, in the phrase "I am 40 years old," Amazon Transcribe recognizes "40 years" as an age.
DATE_TIME	A date can include a year, month, day, day of week, or time of day. For example, Amazon Transcribe recognizes "January 19, 2020" or "11 am" as dates. Amazon Transcribe will recognize partial dates, date ranges, and date intervals. It will also recognize decades, such as "the 1990s".
LICENSE_PLATE	A license plate for a vehicle is issued by the state or country where the vehicle is registered. The format for passenger vehicles is typically five to eight digits, consisting of upper-case letters and numbers. The format varies depending on the location of the issuing state or country.
PASSPORT_NUMBER	A unique identifier assigned to an individua l's passport. The format typically includes a combination of letters and numbers and varies by country.
PASSWORD	An alphanumeric string that is used as a password, such as "*very20special#pass*".

PII type	Description
USERNAME	A user name that identifies an account, such as a login name, screen name, nick name, or handle.
VEHICLE_IDENTIFICATION_NUMBER	A Vehicle Identification Number (VIN) uniquely identifies a vehicle. VIN content and format are defined in the ISO 3779 specification. Each country has specific codes and formats for VINs.

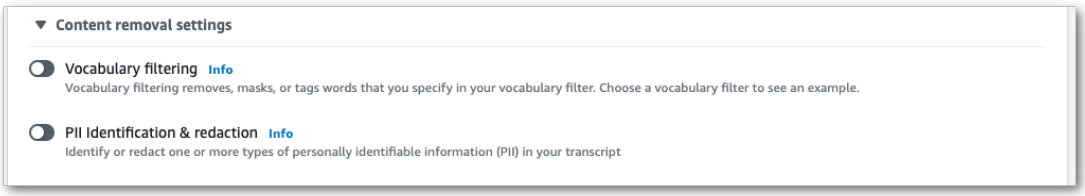
You can start a streaming transcription using the AWS Management Console, WebSocket, or HTTP/2.

AWS Management Console

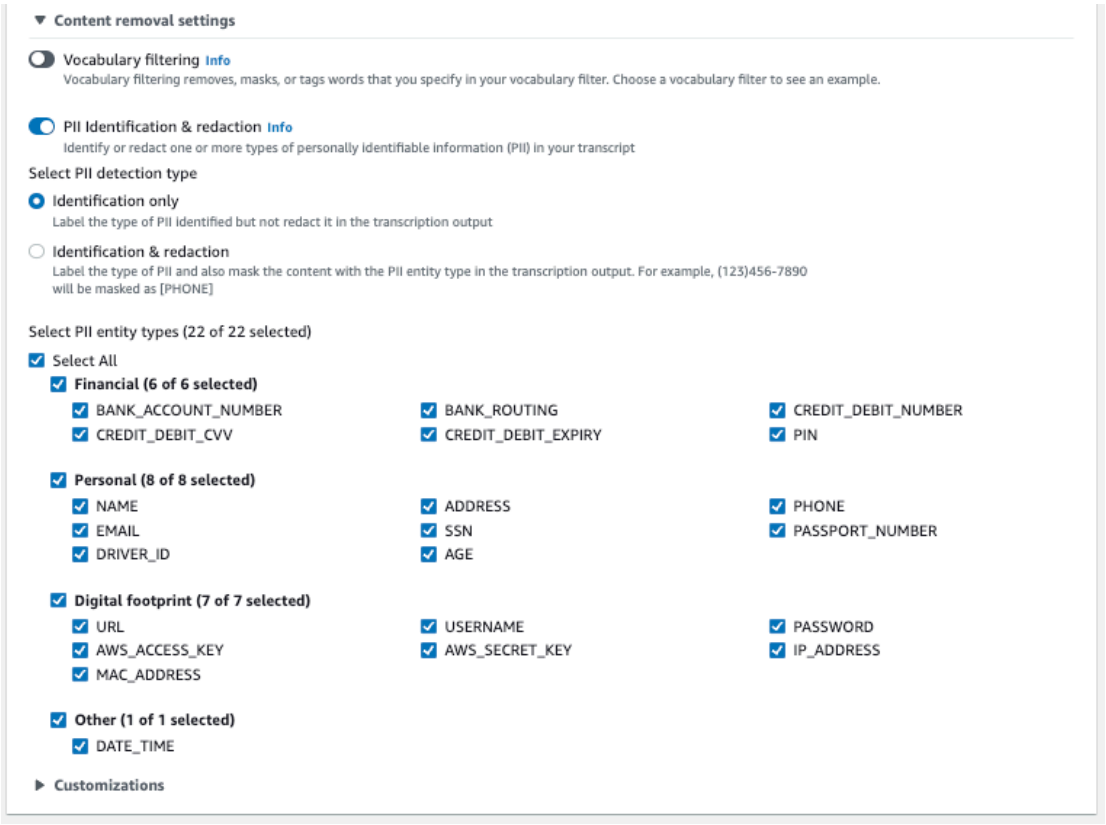
1. Sign into the AWS Management Console.
2. In the navigation pane, choose **Real-time transcription**. Scroll down to **Content removal settings** and expand this field if it is minimized.



3. Toggle on **PII Identification & redaction**.



4. Select **Identification only** or **Identification & redaction**, then select the PII entity types you want to identify or redact in your transcript.



5. You're now ready to transcribe your stream. Select **Start streaming** and begin speaking. To end your dictation, select **Stop streaming**.

WebSocket stream

This example creates a presigned URL that uses PII redaction (or PII identification) in a WebSocket stream. Line breaks have been added for readability. For more information on using WebSocket streams with Amazon Transcribe, see Setting up a WebSocket stream. For more detail on parameters, see `StartStreamTranscription`.

```
GET wss://transcribestreaming.us-west-2.amazonaws.com:8443/stream-transcription-
websocket?
&X-Amz-Algorithm=AWS4-HMAC-SHA256
&X-Amz-Credential=AKIAIOSFODNN7EXAMPLE%2F20220208%2Fus-
west-2%2Ftranscribe%2Faws4_request
&X-Amz-Date=20220208T235959Z
&X-Amz-Expires=300
&X-Amz-Security-Token=security-token
&X-Amz-Signature=string
&X-Amz-SignedHeaders=content-type%3Bhost%3Bx-amz-date
&language-code=en-US
&media-encoding=flac
&sample-rate=16000
&pii-entity-types=NAME,ADDRESS
&content-redaction-type=PII (or &content-identification-type=PII)
```

You cannot use both `content-identification-type` and `content-redaction-type` in the same request.

Parameter definitions can be found in the API Reference; parameters common to all AWS API operations are listed in the Common Parameters section.

HTTP/2 stream

This example creates an HTTP/2 request with PII identification or PII redaction enabled. For more information on using HTTP/2 streaming with Amazon Transcribe, see Setting up an HTTP/2 stream. For more detail on parameters and headers specific to Amazon Transcribe, see `StartStreamTranscription`.

```
POST /stream-transcription HTTP/2
host: transcribestreaming.us-west-2.amazonaws.com
X-Amz-Target: com.amazonaws.transcribe.Transcribe.StartStreamTranscription
Content-Type: application/vnd.amazon.eventstream
X-Amz-Content-Sha256: string
X-Amz-Date: 20220208T235959Z
Authorization: AWS4-HMAC-SHA256 Credential=access-key/20220208/us-west-2/transcribe/
aws4_request, SignedHeaders=content-type;host;x-amz-content-sha256;x-amz-date;x-amz-
target;x-amz-security-token, Signature=string
x-amzn-transcribe-language-code: en-US
x-amzn-transcribe-media-encoding: flac
x-amzn-transcribe-sample-rate: 16000
```

```
x-amzn-transcribe-content-identification-type: PII (or x-amzn-transcribe-content-
redaction-type: PII)
x-amzn-transcribe-pii-entity-types: NAME,ADDRESS
transfer-encoding: chunked
```

You cannot use both `content-identification-type` and `content-redaction-type` in the same request.

Parameter definitions can be found in the API Reference; parameters common to all AWS API operations are listed in the Common Parameters section.

> **Note**
>
> PII redaction for streaming is only supported in these AWS Regions: Asia Pacific (Seoul), Asia Pacific (Sydney), Asia Pacific (Tokyo), Canada (Central), EU (Frankfurt), EU (Ireland), EU (London), US East (N. Virginia), US East (Ohio), and US West (Oregon).

Example PII redaction and identification output

The following examples show redacted output from batch and streaming jobs, and PII identification from a streaming job.

Transcription jobs using content redaction generate two types of `confidence` values. The Automatic Speech Recognition (ASR) confidence indicates the items that have the `type` of `pronunciation` or `punctuation` is a specific utterance. In the following transcript output, the word Good has a `confidence` of `1.0`. This confidence value indicates that Amazon Transcribe is 100 percent confident that the word uttered in this transcript is 'Good'. The `confidence` value for a `[PII]` tag is the confidence that the speech it flagged for redaction is truly PII. In the following transcript output, the `confidence` of `0.9999` indicates that Amazon Transcribe is 99.99 percent confident that the entity it redacted in the transcript is PII.

Example redacted output (batch)

```
{
    "jobName": "my-first-transcription-job",
    "accountId": "111122223333",
    "isRedacted": true,
    "results": {
        "transcripts": [
```

```
            {
                "transcript": "Good morning, everybody. My name is [PII], and today I
feel like
                sharing a whole lot of personal information with you. Let's start with
my Social
                Security number [PII]. My credit card number is [PII] and my C V V code
is [PII].
                I hope that Amazon Transcribe is doing a good job at redacting that
personal
                information away. Let's check."
            }
        ],
        "items": [
            {
                "id": 0,
                "start_time": "2.86",
                "end_time": "3.35",
                "alternatives": [
                    {
                        "confidence": "1.0",
                        "content": "Good"
                    }
                ],
                "type": "pronunciation"
            },
            Items removed for brevity
            {
                "id": 8,
                "start_time": "5.56",
                "end_time": "6.25",
                "alternatives": [
                    {
                        "content": "[PII]",
                        "redactions": [
                            {
                                "confidence": "0.9999",
                                "type": "NAME",
                                "category": "PII"
                            }
                        ]
                    }
                ],
                "type": "pronunciation"
            },
```

```
            Items removed for brevity
        ],
    },
    "status": "COMPLETED"
}
```

Here's the unredacted transcript for comparison:

```
{
    "jobName": "job id",
    "accountId": "111122223333",
    "isRedacted": false,
    "results": {
        "transcripts": [
            {
                "transcript": "Good morning, everybody. My name is Mike, and today I
 feel like
                sharing a whole lot of personal information with you. Let's start with
 my Social
                Security number 000000000. My credit card number is 5555555555555555
                and my C V V code is 000. I hope that Amazon Transcribe is doing a good
 job
                at redacting that personal information away. Let's check."
            }
        ],
        "items": [
            {
                "id": 0,
                "start_time": "2.86",
                "end_time": "3.35",
                "alternatives": [
                    {
                        "confidence": "1.0",
                        "content": "Good"
                    }
                ],
                "type": "pronunciation"
            },
            Items removed for brevity
            {
                "id": 8,
                "start_time": "5.56",
                "end_time": "6.25",
```

```
                "alternatives": [
                    {
                        "confidence": "0.9999",
                        "content": "Mike",
                     {
                ],
                "type": "pronunciation"
            },
            Items removed for brevity
        ],
    },
    "status": "COMPLETED"
}
```

Example redacted streaming output

```
{
    "TranscriptResultStream": {
        "TranscriptEvent": {
            "Transcript": {
                "Results": [
                    {
                        "Alternatives": [
                            {
                                "Transcript": "my name is [NAME]",
                                "Items": [
                                    {
                                        "Content": "my",
                                        "EndTime": 0.3799375,
                                        "StartTime": 0.0299375,
                                        "Type": "pronunciation"
                                    },
                                    {
                                        "Content": "name",
                                        "EndTime": 0.5899375,
                                        "StartTime": 0.3899375,
                                        "Type": "pronunciation"
                                    },
                                    {
                                        "Content": "is",
                                        "EndTime": 0.7899375,
                                        "StartTime": 0.5999375,
                                        "Type": "pronunciation"
```

```
                                    },
                                    {
                                        "Content": "[NAME]",
                                        "EndTime": 1.0199375,
                                        "StartTime": 0.7999375,
                                        "Type": "pronunciation"
                                    }
                                ],
                                "Entities": [
                                    {
                                        "Content": "[NAME]",
                                        "Category": "PII",
                                        "Type": "NAME",
                                        "StartTime" : 0.7999375,
                                        "EndTime" : 1.0199375,
                                        "Confidence": 0.9989
                                    }
                                ]
                            }
                        ],
                        "EndTime": 1.02,
                        "IsPartial": false,
                        "ResultId": "12345a67-8bc9-0de1-2f34-a5b678c90d12",
                        "StartTime": 0.0199375
                    }
                ]
            }
        }
    }
}
```

Example PII identification output

PII identification is an additional feature that you can use with your streaming transcription job. The identified PII is listed in each segment's `Entities` section.

```
{
    "TranscriptResultStream": {
        "TranscriptEvent": {
            "Transcript": {
                "Results": [
                    {
                        "Alternatives": [
```

```
                    {
                        "Transcript": "my name is mike",
                        "Items": [
                            {
                                "Content": "my",
                                "EndTime": 0.3799375,
                                "StartTime": 0.0299375,
                                "Type": "pronunciation"
                            },
                            {
                                "Content": "name",
                                "EndTime": 0.5899375,
                                "StartTime": 0.3899375,
                                "Type": "pronunciation"
                            },
                            {
                                "Content": "is",
                                "EndTime": 0.7899375,
                                "StartTime": 0.5999375,
                                "Type": "pronunciation"
                            },
                            {
                                "Content": "mike",
                                "EndTime": 0.9199375,
                                "StartTime": 0.7999375,
                                "Type": "pronunciation"

                            }
                        ],
                        "Entities": [
                            {
                                "Content": "mike",
                                "Category": "PII",
                                "Type": "NAME",
                                "StartTime" : 0.7999375,
                                "EndTime" : 1.0199375,
                                "Confidence": 0.9989
                            }
                        ]
                    }
                ],
                "EndTime": 1.02,
                "IsPartial": false,
                "ResultId": "12345a67-8bc9-0de1-2f34-a5b678c90d12",
```

```
                        "StartTime": 0.0199375
                    }
                ]
            }
        }
    }
}
```

Creating video subtitles

Amazon Transcribe supports WebVTT (*.vtt) and SubRip (*.srt) output for use as video subtitles. You can select one or both file types when setting up your batch video transcription job. When using the subtitle feature, your selected subtitle file(s) and a regular transcript file (containing additional information) are produced. Subtitle and transcription files are output to the same destination.

Subtitles are displayed at the same time text is spoken, and remain visible until there is a natural pause or the speaker finishes talking. Note that if you enable subtitles in your transcription request and your audio contains no speech, a subtitle file is not created.

> **Important**
>
> Amazon Transcribe uses a default start index of `0` for subtitle output, which differs from the more widely used value of 1. If you require a start index of 1, you can specify this in the AWS Management Console or in your API request using the `OutputStartIndex` parameter.

Using the incorrect start index can result in compatibility errors with other services, so be sure to verify which start index you require before creating your subtitles. If you're uncertain which value to use, we recommend choosing 1. Refer to `Subtitles` for more information.

Features supported with subtitles:

- **Content redaction** — Any redacted content is reflected as 'PII' in both your subtitle and regular transcript output files. The audio is not altered.
- **Vocabulary filters** — Subtitle files are generated from the transcription file, so any words you filter in your standard transcription output are also filtered in your subtitles. Filtered content is displayed as whitespace or *** in your transcript and subtitle files. The audio is not altered.
- **Speaker diarization** — If there are multiple speakers in a given subtitle segment, dashes are used to distinguish each speaker. This applies to both WebVTT and SubRip formats; for example:
  - -- Text spoken by Person 1
  - -- Text spoken by Person 2

Subtitle files are stored in the same Amazon S3 location as your transcription output.

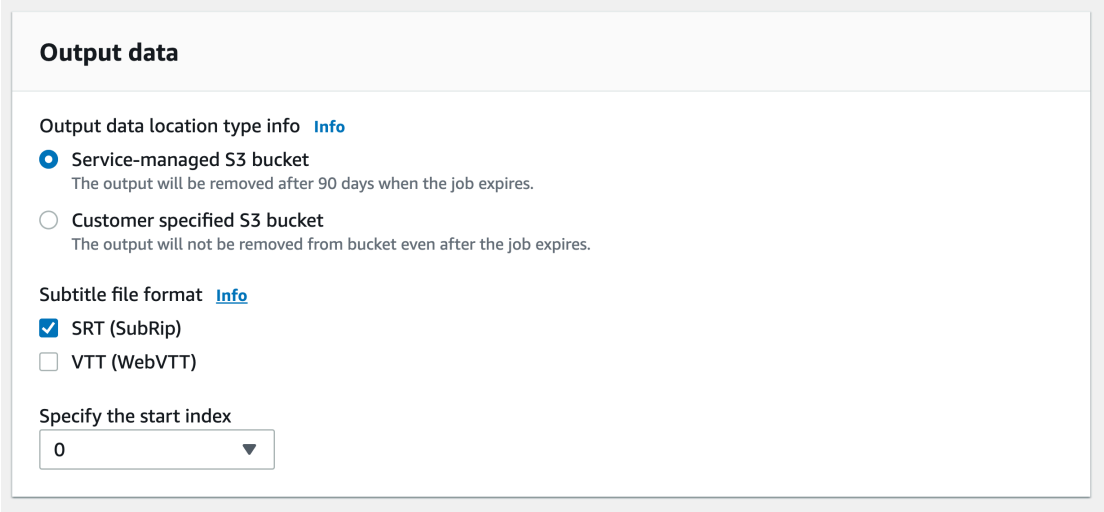
See Amazon Transcribe Video Snacks: Creating Video Subtitles Without Writing Any Code for a video walkthrough of creating subtitles.

Generating subtitle files

You can create subtitle files using the **AWS Management Console**, **AWS CLI**, or **AWS SDKs**; see the following examples:

AWS Management Console

1. Sign in to the AWS Management Console.
2. In the navigation pane, choose **Transcription jobs**, then select **Create job** (top right). This opens the **Specify job details** page. Subtitle options are located in the **Output data** panel.
3. Select the formats you want for your subtitles files, then choose a value for your start index. Note that the Amazon Transcribe default is `0`, but 1 is more widely used. If you're uncertain which value to use, we recommend choosing 1, as this may improve compatibility with other services.



4. Fill in any other fields you want to include on the **Specify job details** page, then select **Next**. This takes you to the **Configure job - *optional* page**.
5. Select **Create job** to run your transcription job.

AWS CLI

This example uses the start-transcription-job command and `Subtitles` parameter. For more information, see `StartTranscriptionJob` and `Subtitles`.

```
aws transcribe start-transcription-job \
--region us-west-2 \
--transcription-job-name my-first-transcription-job \
--media MediaFileUri=s3://amzn-s3-demo-bucket/my-input-files/my-media-file.flac \
--output-bucket-name amzn-s3-demo-bucket \
--output-key my-output-files/ \
--language-code en-US \
--subtitles Formats=vtt,srt,OutputStartIndex=1
```

Here's another example using the start-transcription-job command, and a request body that adds subtitles to that job.

```
aws transcribe start-transcription-job \
--region us-west-2 \
--cli-input-json file://my-first-subtitle-job.json
```

The file *my-first-subtitle-job.json* contains the following request body.

```
{
  "TranscriptionJobName": "my-first-transcription-job",
  "Media": {
        "MediaFileUri": "s3://amzn-s3-demo-bucket/my-input-files/my-media-file.flac"
  },
  "OutputBucketName": "amzn-s3-demo-bucket",
  "OutputKey": "my-output-files/",
  "LanguageCode": "en-US",
  "Subtitles": {
        "Formats": [
            "vtt","srt"
        ],
        "OutputStartIndex": 1
   }
}
```

AWS SDK for Python (Boto3)

This example uses the AWS SDK for Python (Boto3) to add subtitles using the `Subtitles` argument for the start_transcription_job method. For more information, see `StartTranscriptionJob` and `Subtitles`.

For additional examples using the AWS SDKs, including feature-specific, scenario, and cross-service examples, refer to the Code examples for Amazon Transcribe using AWS SDKs chapter.

```
from __future__ import print_function
import time
import boto3
transcribe = boto3.client('transcribe', 'us-west-2')
job_name = "my-first-transcription-job"
job_uri = "s3://amzn-s3-demo-bucket/my-input-files/my-media-file.flac"
transcribe.start_transcription_job(
    TranscriptionJobName = job_name,
    Media = {
        'MediaFileUri': job_uri
    },
    OutputBucketName = 'amzn-s3-demo-bucket',
    OutputKey = 'my-output-files/',
    LanguageCode = 'en-US',
    Subtitles = {
        'Formats': [
            'vtt','srt'
        ],
        'OutputStartIndex': 1
   }
)

while True:
    status = transcribe.get_transcription_job(TranscriptionJobName = job_name)
    if status['TranscriptionJob']['TranscriptionJobStatus'] in ['COMPLETED', 'FAILED']:
        break
    print("Not ready yet...")
    time.sleep(5)
print(status)
```

Analyzing call center audio with Call Analytics

Use Amazon Transcribe Call Analytics to gain insight into customer-agent interactions. Call Analytics is designed specifically for call center audio and automatically provides you with valuable data relating to each call and each participant. You can also hone in on data at specific points throughout call. For example, you can compare customer sentiment in a call's first few seconds to the last quarter of the call to see if your agent provided a positive experience. Other use case examples are listed in the following section.

Call Analytics is available for post-call and real-time transcriptions. If you're transcribing a file located in an Amazon S3 bucket, you're performing a post-call transcription. If you're transcribing an audio stream, you're performing a real-time transcription. These two transcription methods offer different Call Analytics insights and features. For more detail on each method, see Post-call analytics and Real-time Call Analytics.

With real-time Call Analytics transcriptions, you can also include post-call analytics in your request. Your post-call analytics transcript is stored in the Amazon S3 bucket that you specify in your request. Refer to Post-call analytics with real-time transcriptions for more information.

API operations specific to Call Analytics

Post-call: `CreateCallAnalyticsCategory`, `DeleteCallAnalyticsCategory`, `DeleteCallAnalyticsJob`, `GetCallAnalyticsCategory`, `GetCallAnalyticsJob`, `ListCallAnalyticsCategories`, `ListCallAnalyticsJobs`, `StartCallAnalyticsJob`, `UpdateCallAnalyticsCategory`
Real-time: `StartCallAnalyticsStreamTranscription`, StartCallAnalyticsStreamTranscriptionWebSocket

Common use cases

Post-call transcriptions:

- **Monitor issue frequency over time**: Use call categorization to identify recurring keywords within your transcripts.

- **Gain insight into your customer service experience**: Use call characteristics (non-talk time, talk time, interruptions, voice loudness, talk speed) and sentiment analysis to determine if customer issues are being appropriately resolved during the call.
- **Ensure regulatory compliance or adherence to company policy**: Set keywords and phrases for company-specific greetings or disclaimers to verify that your agents are meeting regulatory requirements.
- **Improve handling of customers' personal data**: Use PII redaction in your transcription output or audio file to help protect customer privacy.
- **Improve staff training**: Use criteria (sentiment, non-talk time, interruptions, talk speed) to flag transcripts that can be used as examples of positive or negative customer interactions.
- **Measure staff efficacy in creating a positive customer experience**: Use sentiment analysis to measure if your agents are able to turn a negative customer sentiment into a positive one as calls progress.
- **Improve data organization**: Label and sort calls based on custom categories (including keywords and phrases, sentiment, talk time, and interruptions).
- **Summarize the important aspects of a call using Generative AI**: Use generative call summarization to get a concise summary of the transcript, which includes key components such as issues, action items and outcomes discussed in the call.

Real-time transcriptions:

- **Mitigate escalations in real time**: Set up real-time alerts for key phrases—such as a customer saying "speak to a manager"—to flag calls as they begin to escalate. You can create real-time alerts using real-time category matches.
- **Improve handling of customer data**: Use PII identification or PII redaction in your transcription output to help protect customer privacy.
- **Identify custom keywords and phrases**: Use custom categories to flag specific keywords in a call.
- **Automatically identify issues**: Use automatic issue detection to get a succinct summary of all issues identified in a call.
- **Measure staff efficacy in creating a positive customer experience**: Use sentiment analysis to measure if your agents are able to turn a negative customer sentiment into a positive one as calls progress.

- **Set up agent-assist**: Use the insights of your choice to provide your agents with proactive assistance in resolving customer calls. See Live Call Analytics and agent assist for your contact center with Amazon language AI services for more information.

To get started, see Starting a post-call analytics transcription and Starting a real-time Call Analytics transcription. Call Analytics output is similar to that of a standard transcription job, but contains additional analytics data. To view sample output, see Post-call analytics output and Real-time Call Analytics output.

Considerations and additional information

Before using Call Analytics, note that:

- Call Analytics only supports two-channel audio, where an agent is present on one channel and a customer is present on a second channel.
- Job queueing is always enabled for post-call analytics jobs, so you're limited to 100 concurrent Call Analytics jobs. If you want to request a quota increase, see AWS service quotas.
- Input files for post-call analytics jobs cannot be greater than 500 MB and must be less than 4 hours. For audio files with frequency greater than 8kHz, the audio duration must be less than or equal to 2 hours. Note that the file size limit may be smaller for certain compressed, non-WAV audio file formats.
- If using categories, you must create all desired categories before starting a Call Analytics transcription. Any new categories cannot be applied to existing transcriptions. To learn how to create a new category, see Creating categories for post-call transcriptions and Creating categories for real-time transcriptions.
- Some Call Analytics quotas differ from Amazon Transcribe and Amazon Transcribe Medical; refer to the AWS General Reference for details.

Dive deeper with the AWS Machine Learning Blog

To learn more about Call Analytics options, see:

- Post Call Analytics for your contact center with Amazon language AI services
- Live Call Analytics and agent assist for your contact center with Amazon language AI services

To view sample Call Analytics output and features, see our GitHub demo. We also offer a JSON to Word document application to convert your transcript into an easy-to-read format.

Region availability and quotas

Call Analytics is supported in the following AWS Regions:

Region	Transcription type
ap-northeast-1 (Tokyo)	post-call, real-time
ap-northeast-2 (Seoul)	post-call, real-time
ap-south-1 (Mumbai)	post-call
ap-southeast-1 (Singapore)	post-call
ap-southeast-2 (Sydney)	post-call, real-time
ca-central-1 (Canada, Central)	post-call, real-time
eu-central-1 (Frankfurt)	post-call, real-time
eu-west-2 (London)	post-call, real-time
us-east-1 (N. Virginia)	post-call, real-time
us-west-2 (Oregon)	post-call, real-time

Note that Region support differs for Amazon Transcribe, Amazon Transcribe Medical, and Call Analytics.

To get the endpoints for each supported Region, see Service endpoints in the *AWS General Reference*.

For a list of quotas that pertain to your transcriptions, refer to the Service quotas in the *AWS General Reference*. Some quotas can be changed upon request. If the **Adjustable** column contains '**Yes**', you can request an increase. To do so, select the provided link.

Post-call analytics

Call Analytics provides post-call analyses, which are useful for monitoring customer service trends.

Post-call transcriptions offer the following insights:

- Call characteristics, including talk time, non-talk time, speaker loudness, interruptions, talk speed, issues, outcomes, and action items
- Generative call summarization, which creates a concise summary of the entire call
- Custom categorization with rules that you can use to hone in on specific keywords and criteria
- PII redaction of your text transcript and your audio file
- Speaker sentiment for each caller at various points in a call

Post-call insights

This section details the insights available for post-call analytics transcriptions.

Call characteristics

The call characteristics feature measures the quality of agent-customer interactions using these criteria:

- **Interruption**: Measures if and when one participant cuts off the other participant mid-sentence. Frequent interruptions may be associated with rudeness or anger, and could correlate to negative sentiment for one or both participants.
- **Loudness**: Measures the volume at which each participant is speaking. Use this metric to see if the caller or the agent is speaking loudly or yelling, which is often indicative of being upset. This metric is represented as a normalized value (speech level per second of speech in a given segment) on a scale from 0 to 100, where a higher value indicates a louder voice.
- **Non-talk time**: Measures periods of time that do not contain speech. Use this metric to see if there are long periods of silence, such as an agent keeping a customer on hold for an excessive amount of time.
- **Talk speed**: Measures the speed at which both participants are speaking. Comprehension can be affected if one participant speaks too quickly. This metric is measured in words per minute.
- **Talk time**: Measures the amount of time (in milliseconds) each participant spoke during the call. Use this metric to help identify if one participant is dominating the call or if the dialogue is balanced.

- **Issues, Outcomes, and Action Items**: Identifies issues, outcomes and action items from the call transcript.

Here's an output example.

Generative call summarization

Generative call summarization creates a concise summary of the entire call, capturing key components such as reason for the call, steps taken to resolve issue, and next steps.

Using generative call summarization, you can:

- Reduce the need for manual note-taking during and after calls.
- Improve agent efficiency as they can spend more time talking to callers waiting in queue rather than engaging in after-call work.
- Speed up supervisor reviews as call summaries are much quicker to review than entire transcripts.

To use generative call summarization with a post-call analytics job, see Enabling generative call summarization. For example output, see Generative call summarization output example. Generative call summarization is priced separately (please refer to pricing page).

> **Note**
>
> Generative call summarization is currently available in `us-east-1` and `us-west-2`. This capability is supported with these English language dialects: Australian (`en-AU`), British (`en-GB`), Indian (`en-IN`), Irish (`en-IE`), Scottish (`en-AB`), US (`en-US`), and Welsh (`en-WL`).

Custom categorization

Use call categorization to flag keywords, phrases, sentiment, or actions within a call. Our categorization options can help you triage escalations, such as negative-sentiment calls with many interruptions, or organize calls into specific categories, such as company departments.

The criteria you can add to a category include:

- **Non-talk time**: Periods of time when neither the customer nor the agent is talking.

- **Interruptions**: When the customer or the agent is interrupting the other person.
- **Customer or agent sentiment**: How the customer or the agent is feeling during a specified time period. If at least 50 percent of the conversation turns (the back-and-forth between two speakers) in a specified time period match the specified sentiment, Amazon Transcribe considers the sentiment a match.
- **Keywords or phrases**: Matches part of the transcription based on an exact phrase. For example, if you set a filter for the phrase "I want to speak to the manager", Amazon Transcribe filters for that *exact* phrase.

You can also flag the inverse of the previous criteria (talk time, lack of interruptions, a sentiment not being present, and the lack of a specific phrase).

Here's an output example.

For more information on categories or to learn how to create a new category, see Creating categories for post-call transcriptions.

Sensitive data redaction

Sensitive data redaction replaces personally identifiable information (PII) in the text transcript and the audio file. A redacted transcript replaces the original text with `[PII]`; a redacted audio file replaces spoken personal information with silence. This parameter is useful for protecting customer information.

> **Note**
>
> Post-call PII redaction is supported with US English (`en-US`) and US Spanish (`es-US`).

To view the list of PII that is redacted using this feature, or to learn more about redaction with Amazon Transcribe, see Redacting or identifying personally identifiable information.

Here is an output example.

Sentiment analysis

Sentiment analysis estimates how the customer and agent are feeling throughout the call. This metric is represented as both a quantitative value (with a range from `5` to `-5`) and a qualitative

value (`positive`, `neutral`, `mixed`, or `negative`). Quantitative values are provided per quarter and per call; qualitative values are provided per turn.

This metric can help identify if your agent is able to delight an upset customer by the time the call ends.

Sentiment analysis works out-of-the-box and thus doesn't support customization, such as model training or custom categories.

Here's an output example.

Creating categories for post-call transcriptions

Post-call analytics supports the creation of custom categories, enabling you to tailor your transcript analyses to best suit your specific business needs.

You can create as many categories as you like to cover a range of different scenarios. For each category you create, you must create between 1 and 20 rules. Each rule is based on one of four criteria: interruptions, keywords, non-talk time, or sentiment. For more details on using these criteria with the `CreateCallAnalyticsCategory` operation, refer to the Rule criteria for post-call analytics categories section.

If the content in your media matches all the rules you've specified in a given category, Amazon Transcribe labels your output with that category. See call categorization output for an example of a category match in JSON output.

Here are a few examples of what you can do with custom categories:

- Isolate calls with specific characteristics, such as calls that end with a negative customer sentiment
- Identify trends in customer issues by flagging and tracking specific sets of keywords
- Monitor compliance, such as an agent speaking (or omitting) a specific phrase during the first few seconds of a call
- Gain insight into customer experience by flagging calls with many agent interruptions and negative customer sentiment
- Compare multiple categories to measure correlations, such as analyzing whether an agent using a welcome phrase correlates with positive customer sentiment

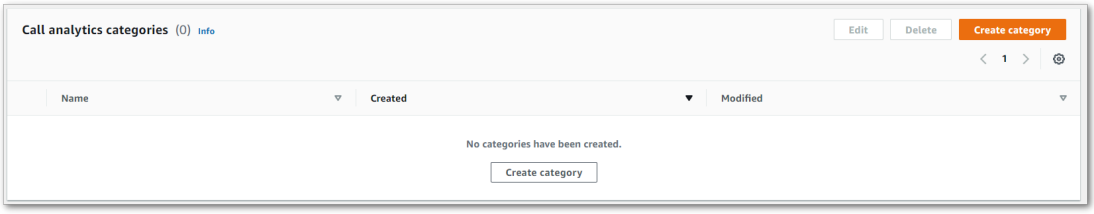
Post-call versus real-time categories

When creating a new category, you can specify whether you want it created as a post-call analytics category (`POST_CALL`) or as a real-time Call Analytics category (`REAL_TIME`). If you don't specify an option, your category is created as a post-call category by default. Post-call analytics category matches are available in your output upon completion of your post-call analytics transcription.

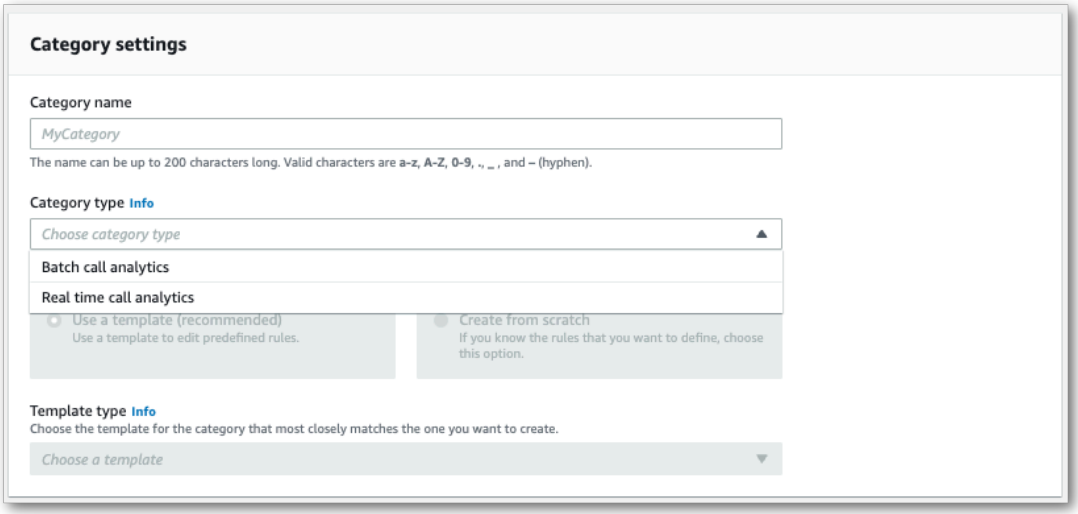
To create a new category for post-call analytics, you can use the **AWS Management Console**, **AWS CLI**, or **AWS SDKs**; see the following for examples:

AWS Management Console

1. In the navigation pane, under Amazon Transcribe, choose **Amazon Transcribe Call Analytics**.
2. Choose **Call analytics categories**, which takes you to the **Call analytics categories** page. Select **Create category**.

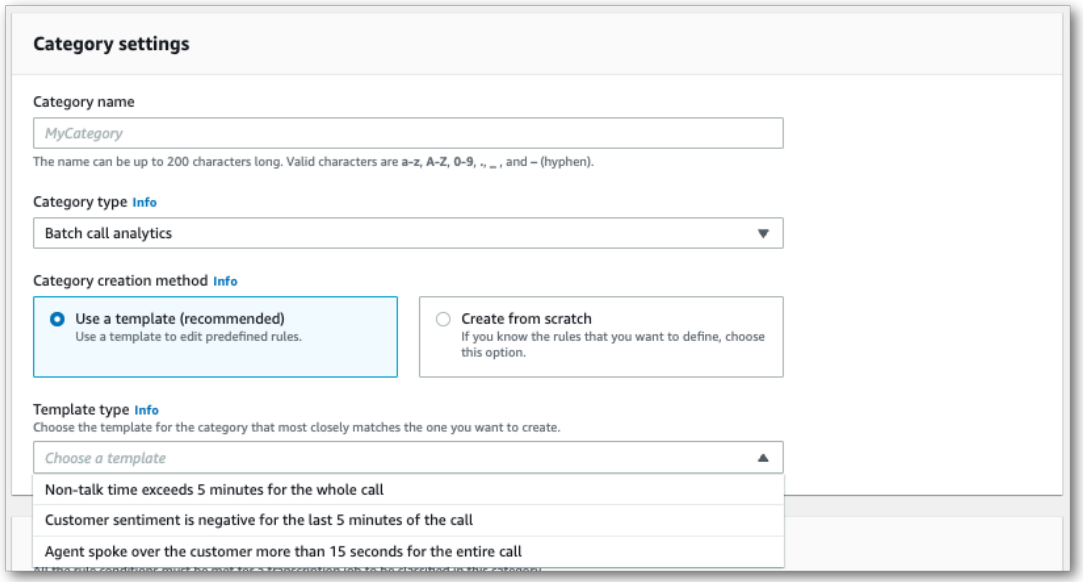


3. You're now on the **Create category page**. Enter a name for your category, then choose 'Batch call analytics' in the **Category type** dropdown menu.

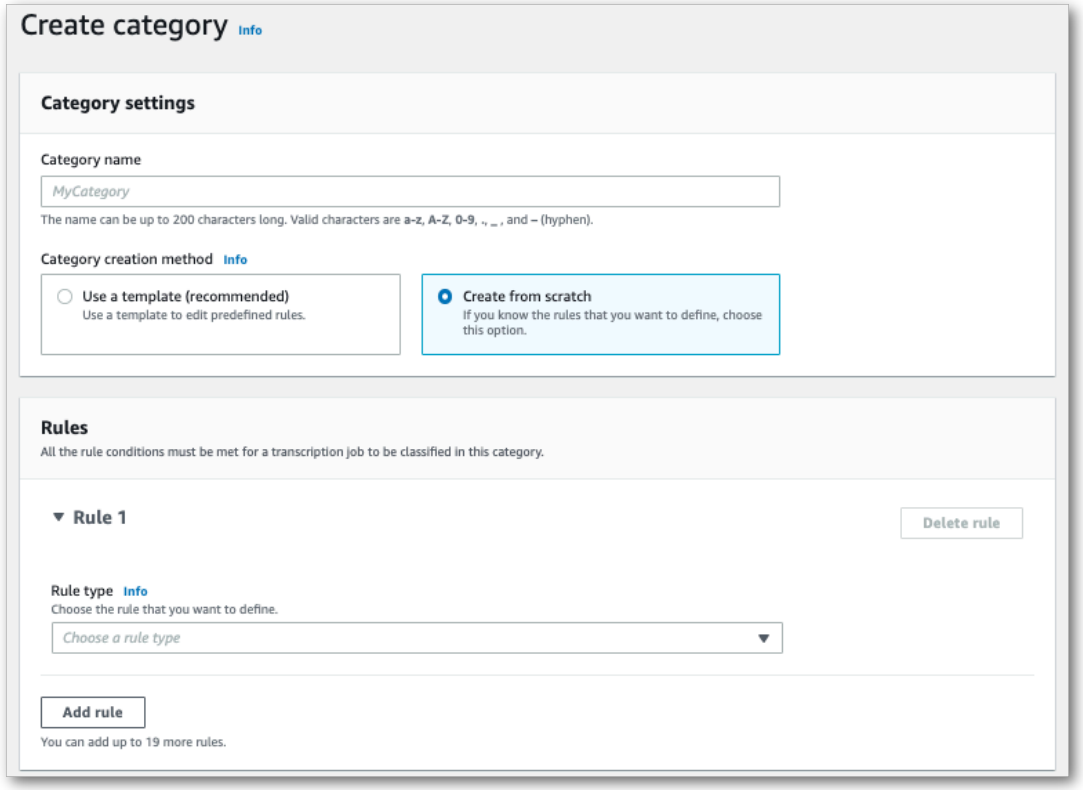


4. You can choose a template to create your category or you can make one from scratch.

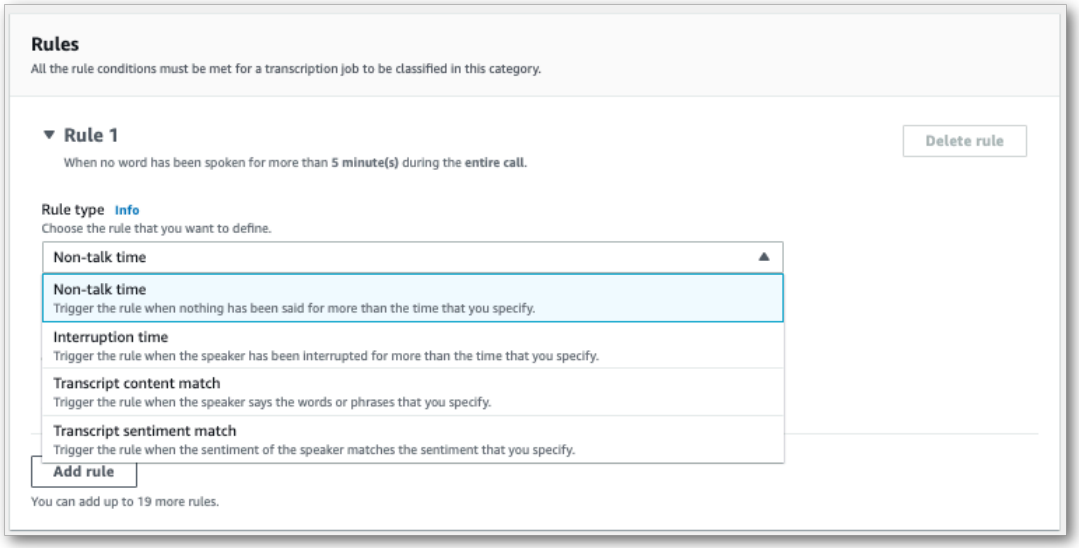
   If using a template: select **Use a template (recommended)**, choose the template you want, then select **Create category**.



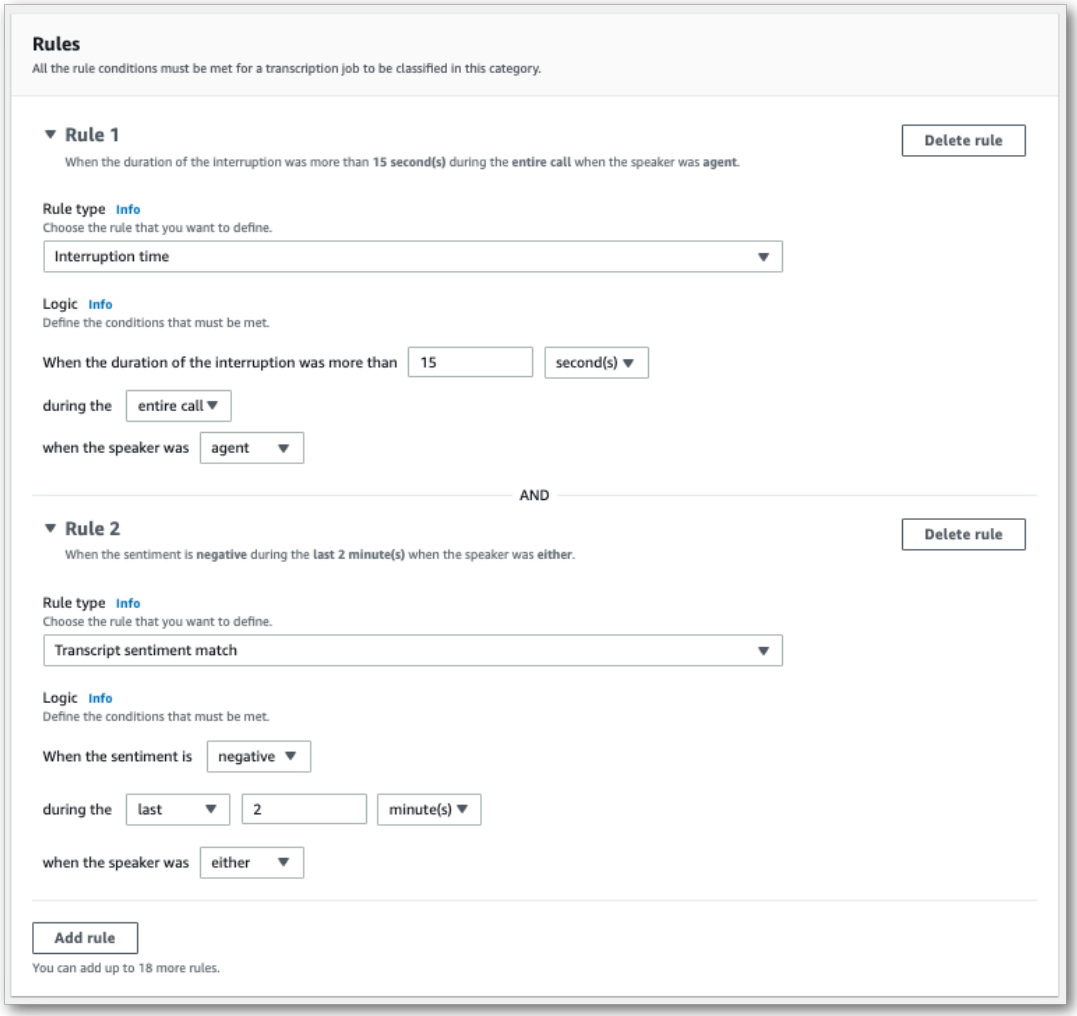
5. If creating a custom category: select **Create from scratch**.



6. Add rules to your category using the dropdown menu. You can add up to 20 rules per category.



7. Here's an example of a category with two rules: an agent who interrupts a customer for more than 15 seconds during the call and a negative sentiment felt by the customer or the agent in the last two minutes of the call.



8. When you're finished adding rules to your category, choose **Create category**.

AWS CLI

This example uses the create-call-analytics-category command. For more information, see `CreateCallAnalyticsCategory`, `CategoryProperties`, and `Rule`.

The following example creates a category with the rules:

- The customer was interrupted in the first 60,000 milliseconds. The duration of these interruptions lasted at least 10,000 milliseconds.
- There was a period of silence that lasted at least 20,000 milliseconds between 10% into the call and 80% into the call.
- The agent had a negative sentiment at some point in the call.

- The words "welcome" or "hello" were not used in the first 10,000 milliseconds of the call.

This example uses the create-call-analytics-category command, and a request body that adds several rules to your category.

```
aws transcribe create-call-analytics-category \
--cli-input-json file://filepath/my-first-analytics-category.json
```

The file *my-first-analytics-category.json* contains the following request body.

```
{
  "CategoryName": "my-new-category",
  "InputType": "POST_CALL",
  "Rules": [
        {
            "InterruptionFilter": {
                "AbsoluteTimeRange": {
                    "First": 60000
                },
                "Negate": false,
                "ParticipantRole": "CUSTOMER",
                "Threshold": 10000
            }
        },
        {
            "NonTalkTimeFilter": {
                "Negate": false,
                "RelativeTimeRange": {
                    "EndPercentage": 80,
                    "StartPercentage": 10
                },
                "Threshold": 20000
            }
        },
        {
            "SentimentFilter": {
                "ParticipantRole": "AGENT",
                "Sentiments": [
                    "NEGATIVE"
                ]
            }
        },
```

```
        {
            "TranscriptFilter": {
                "Negate": true,
                "AbsoluteTimeRange": {
                    "First": 10000
                },
                "Targets": [
                    "welcome",
                    "hello"
                ],
                "TranscriptFilterType": "EXACT"
            }
        }
    ]
}
```

AWS SDK for Python (Boto3)

This example uses the AWS SDK for Python (Boto3) to create a category using the `CategoryName` and `Rules` arguments for the create_call_analytics_category method. For more information, see `CreateCallAnalyticsCategory`, `CategoryProperties`, and `Rule`.

For additional examples using the AWS SDKs, including feature-specific, scenario, and cross-service examples, refer to the Code examples for Amazon Transcribe using AWS SDKs chapter.

The following example creates a category with the rules:

- The customer was interrupted in the first 60,000 milliseconds. The duration of these interruptions lasted at least 10,000 milliseconds.
- There was a period of silence that lasted at least 20,000 milliseconds between 10% into the call and 80% into the call.
- The agent had a negative sentiment at some point in the call.
- The words "welcome" or "hello" were not used in the first 10,000 milliseconds of the call.

```
from __future__ import print_function
import time
import boto3
transcribe = boto3.client('transcribe', 'us-west-2')
category_name = "my-new-category"
transcribe.create_call_analytics_category(
```

```
    CategoryName = category_name,
    InputType = POST_CALL,
    Rules = [
        {
            'InterruptionFilter': {
                'AbsoluteTimeRange': {
                    'First': 60000
                },
                'Negate': False,
                'ParticipantRole': 'CUSTOMER',
                'Threshold': 10000
            }
        },
        {
            'NonTalkTimeFilter': {
                'Negate': False,
                'RelativeTimeRange': {
                    'EndPercentage': 80,
                    'StartPercentage': 10
                },
                'Threshold': 20000
            }
        },
        {
            'SentimentFilter': {
                'ParticipantRole': 'AGENT',
                'Sentiments': [
                    'NEGATIVE'
                ]
            }
        },
        {
            'TranscriptFilter': {
                'Negate': True,
                'AbsoluteTimeRange': {
                    'First': 10000
                },
                'Targets': [
                    'welcome',
                    'hello'
                ],
                'TranscriptFilterType': 'EXACT'
            }
        }
```

```
    ]

)

result = transcribe.get_call_analytics_category(CategoryName = category_name)
print(result)
```

Rule criteria for post-call analytics categories

This section outlines the types of custom POST_CALL rules that you can create using the CreateCallAnalyticsCategory API operation.

Interruption match

Rules using interruptions (InterruptionFilter data type) are designed to match:

- Instances where an agent interrupts a customer
- Instances where a customer interrupts an agent
- Any participant interrupting the other
- A lack of interruptions

Here's an example of the parameters available with InterruptionFilter:

```
"InterruptionFilter": {
    "AbsoluteTimeRange": {
       Specify the time frame, in milliseconds, when the match should occur
    },
    "RelativeTimeRange": {
       Specify the time frame, in percentage, when the match should occur
    },
    "Negate": Specify if you want to match the presence or absence of interruptions,
    "ParticipantRole": Specify if you want to match speech from the agent, the
 customer, or both,
    "Threshold": Specify a threshold for the amount of time, in seconds, interruptions
 occurred during the call
},
```

Refer to CreateCallAnalyticsCategory and InterruptionFilter for more information on these parameters and the valid values associated with each.

Keyword match

Rules using keywords (TranscriptFilter data type) are designed to match:

- Custom words or phrases spoken by the agent, the customer, or both
- Custom words or phrases **not** spoken by the agent, the customer, or both
- Custom words or phrases that occur in a specific time frame

Here's an example of the parameters available with TranscriptFilter:

```
"TranscriptFilter": {
    "AbsoluteTimeRange": {
       Specify the time frame, in milliseconds, when the match should occur
    },
    "RelativeTimeRange": {
       Specify the time frame, in percentage, when the match should occur
    },
    "Negate": Specify if you want to match the presence or absence of your custom
 keywords,
    "ParticipantRole": Specify if you want to match speech from the agent, the
 customer, or both,
    "Targets": [ The custom words and phrases you want to match ],
    "TranscriptFilterType": Use this parameter to specify an exact match for the
 specified targets
}
```

Refer to CreateCallAnalyticsCategory and TranscriptFilter for more information on these parameters and the valid values associated with each.

Non-talk time match

Rules using non-talk time (NonTalkTimeFilter data type) are designed to match:

- The presence of silence at specified periods throughout the call
- The presence of speech at specified periods throughout the call

Here's an example of the parameters available with NonTalkTimeFilter:

```
"NonTalkTimeFilter": {
    "AbsoluteTimeRange": {
```

```
 Specify the time frame, in milliseconds, when the match should occur
 },
    "RelativeTimeRange": {
 Specify the time frame, in percentage, when the match should occur
 },
    "Negate": Specify if you want to match the presence or absence of speech,
    "Threshold": Specify a threshold for the amount of time, in seconds, silence (or
 speech) occurred during the call
},
```

Refer to CreateCallAnalyticsCategory and NonTalkTimeFilter for more information on these parameters and the valid values associated with each.

Sentiment match

Rules using sentiment (SentimentFilter data type) are designed to match:

- The presence or absence of a positive sentiment expressed by the customer, agent, or both at specified points in the call
- The presence or absence of a negative sentiment expressed by the customer, agent, or both at specified points in the call
- The presence or absence of a neutral sentiment expressed by the customer, agent, or both at specified points in the call
- The presence or absence of a mixed sentiment expressed by the customer, agent, or both at specified points in the call

Here's an example of the parameters available with SentimentFilter:

```
"SentimentFilter": {
    "AbsoluteTimeRange": {
    Specify the time frame, in milliseconds, when the match should occur
    },
    "RelativeTimeRange": {
    Specify the time frame, in percentage, when the match should occur
    },
    "Negate": Specify if you want to match the presence or absence of your chosen
 sentiment,
    "ParticipantRole": Specify if you want to match speech from the agent, the
 customer, or both,
    "Sentiments": [ The sentiments you want to match ]
```

```
},
```

Refer to `CreateCallAnalyticsCategory` and `SentimentFilter` for more information on these parameters and the valid values associated with each.

Starting a post-call analytics transcription

Before starting a post-call analytics transcription, you must create all the categories you want Amazon Transcribe to match in your audio.

> **Note**
>
> Call Analytics transcripts can't be retroactively matched to new categories. Only the categories you create *before* starting a Call Analytics transcription can be applied to that transcription output.

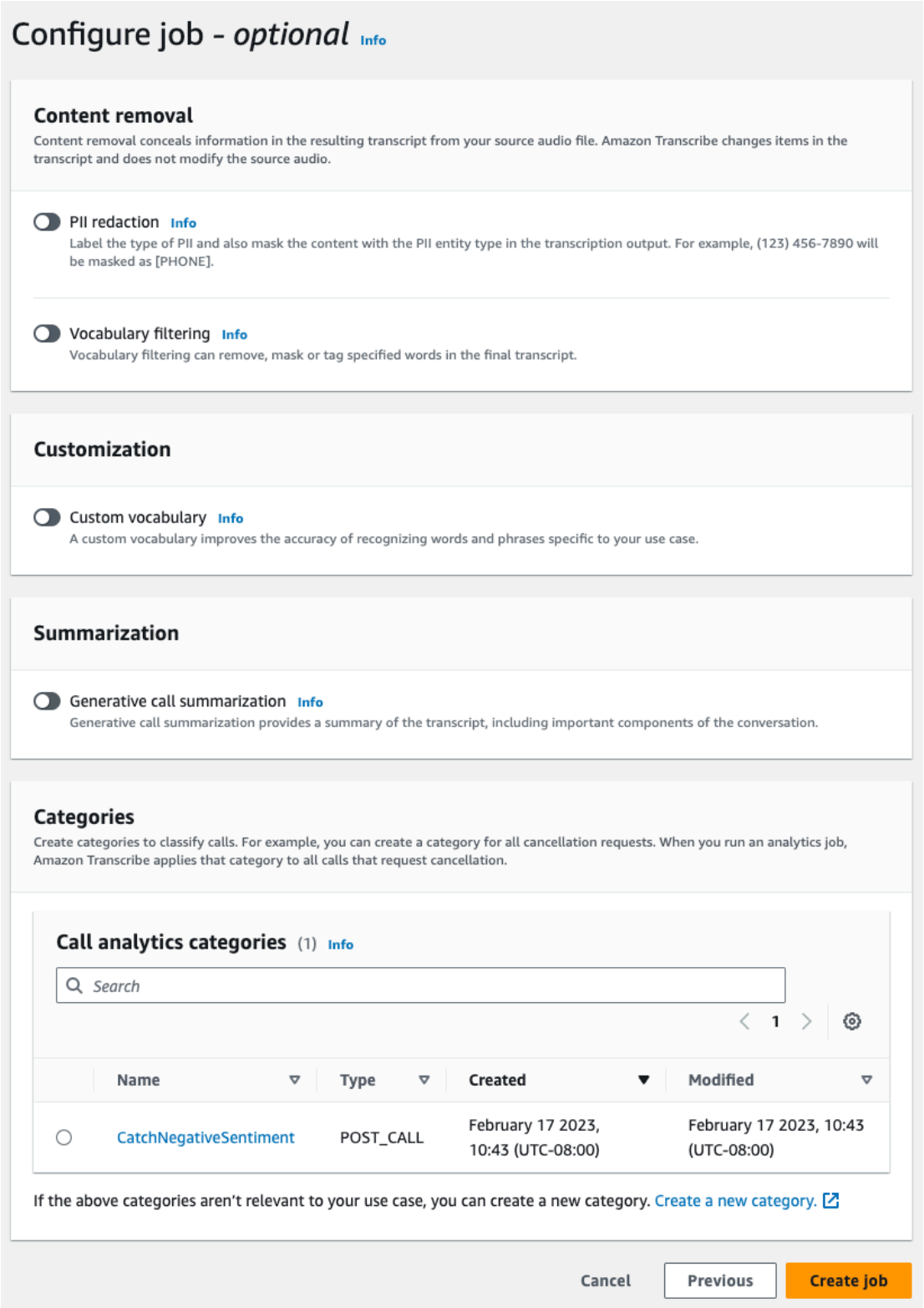
If you've created one or more categories, and your audio matches all the rules within at least one of your categories, Amazon Transcribe flags your output with the matching category. If you choose not to use categories, or if your audio doesn't match the rules specified in your categories, your transcript isn't flagged.

To start a post-call analytics transcription, you can use the **AWS Management Console**, **AWS CLI**, or **AWS SDKs**; see the following for examples:

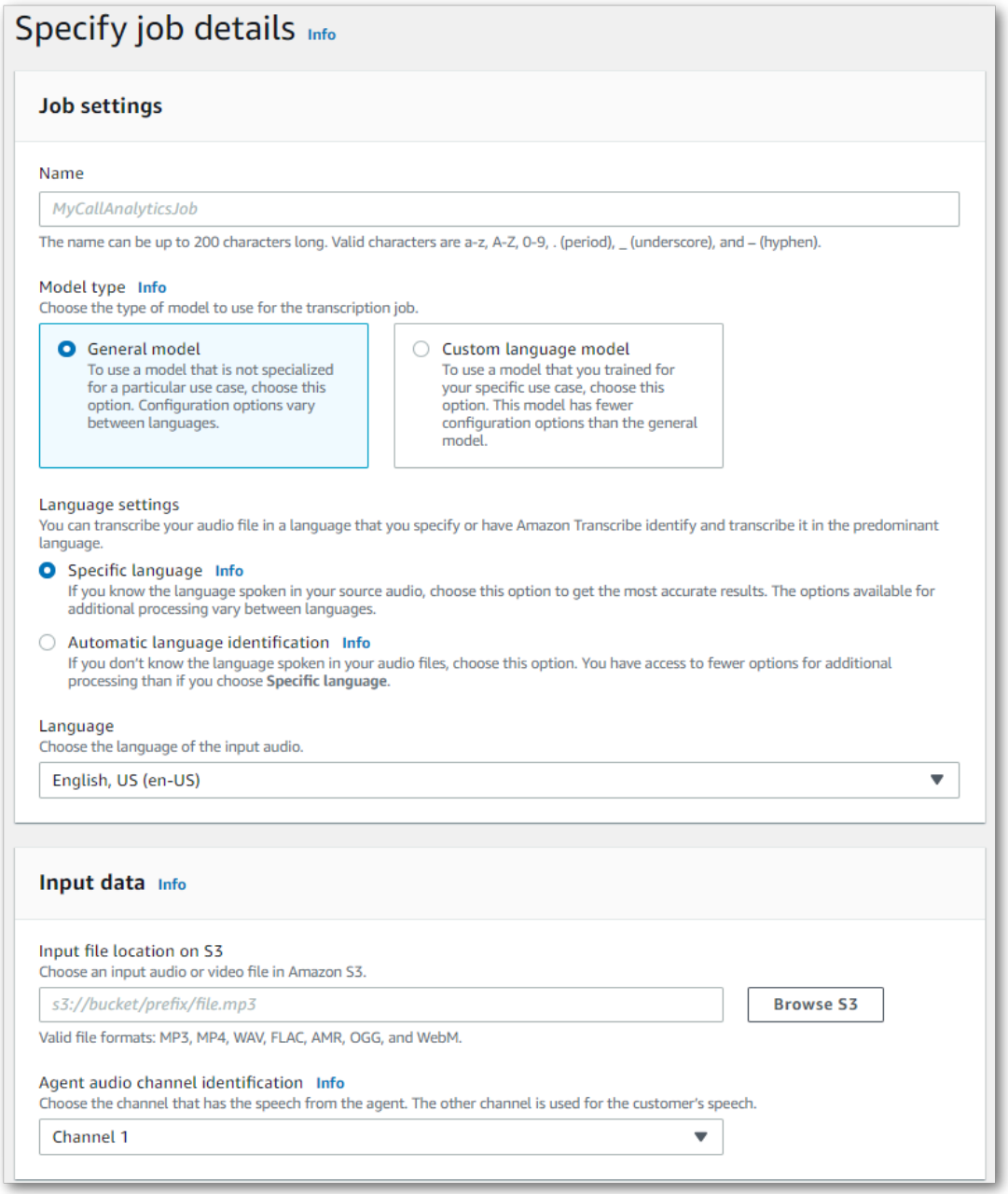
AWS Management Console

Use the following procedure to start a post-call analytics job. The calls that match all characteristics defined by a category are labeled with that category.

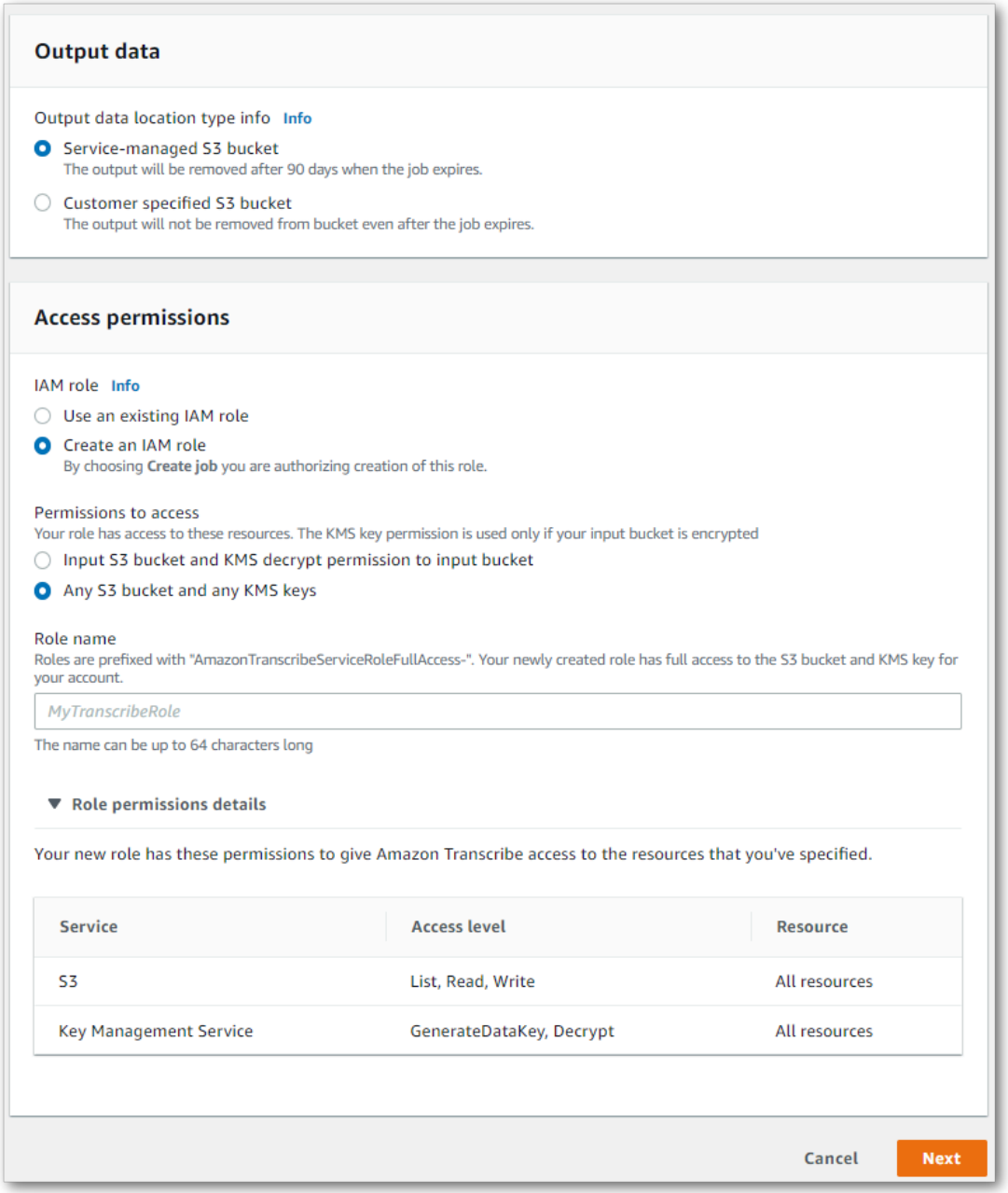
1. In the navigation pane, under Amazon Transcribe Call Analytics, choose **Call analytics jobs**.
2. Choose **Create job**.



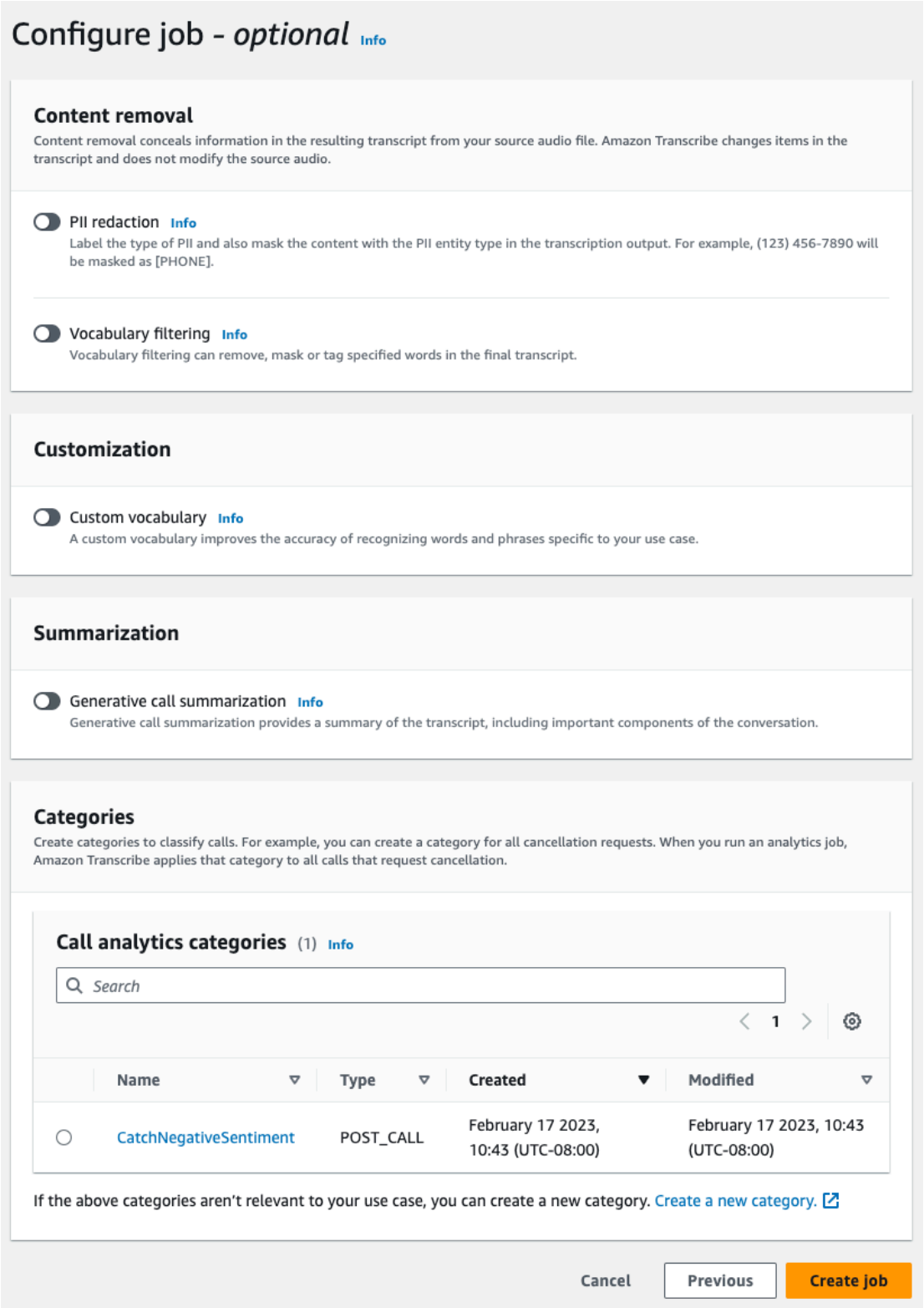
3. On the **Specify job details** page, provide information about your Call Analytics job, including the location of your input data.



Specify the desired Amazon S3 location of your output data and which IAM role to use.



4. Choose **Next**.
5. For **Configure job**, turn on any optional features you want to include with your Call Analytics job. If you previously created categories, they appear in the **Categories** panel and are automatically applied to your Call Analytics job.



6. Choose **Create job**.

AWS CLI

This example uses the start-call-analytics-job command and `channel-definitions` parameter. For more information, see StartCallAnalyticsJob and ChannelDefinition.

```
aws transcribe start-call-analytics-job \
--region us-west-2 \
--call-analytics-job-name my-first-call-analytics-job \
--media MediaFileUri=s3://amzn-s3-demo-bucket/my-input-files/my-media-file.flac \
--output-location s3://amzn-s3-demo-bucket/my-output-files/ \
--data-access-role-arn arn:aws:iam::111122223333:role/ExampleRole \
--channel-definitions ChannelId=0,ParticipantRole=AGENT
 ChannelId=1,ParticipantRole=CUSTOMER
```

Here's another example using the start-call-analytics-job command, and a request body that enables Call Analytics for that job.

```
aws transcribe start-call-analytics-job \
--region us-west-2 \
--cli-input-json file://filepath/my-call-analytics-job.json
```

The file *my-call-analytics-job.json* contains the following request body.

```
{
      "CallAnalyticsJobName": "my-first-call-analytics-job",
      "DataAccessRoleArn": "arn:aws:iam::111122223333:role/ExampleRole",
      "Media": {
          "MediaFileUri": "s3://amzn-s3-demo-bucket/my-input-files/my-media-file.flac"
      },
      "OutputLocation": "s3://amzn-s3-demo-bucket/my-output-files/",
      "ChannelDefinitions": [
          {
              "ChannelId": 0,
              "ParticipantRole": "AGENT"
          },
          {
              "ChannelId": 1,
              "ParticipantRole": "CUSTOMER"
          }
      ]
}
```

AWS SDK for Python (Boto3)

This example uses the AWS SDK for Python (Boto3) to start a Call Analytics job using the start_call_analytics_job method. For more information, see StartCallAnalyticsJob and ChannelDefinition.

For additional examples using the AWS SDKs, including feature-specific, scenario, and cross-service examples, refer to the Code examples for Amazon Transcribe using AWS SDKs chapter.

```
from __future__ import print_function
import time
import boto3
transcribe = boto3.client('transcribe', 'us-west-2')
job_name = "my-first-call-analytics-job"
job_uri = "s3://amzn-s3-demo-bucket/my-input-files/my-media-file.flac"
output_location = "s3://amzn-s3-demo-bucket/my-output-files/"
data_access_role = "arn:aws:iam::111122223333:role/ExampleRole"
transcribe.start_call_analytics_job(
     CallAnalyticsJobName = job_name,
     Media = {
        'MediaFileUri': job_uri
     },
     DataAccessRoleArn = data_access_role,
     OutputLocation = output_location,
     ChannelDefinitions = [
        {
            'ChannelId': 0,
            'ParticipantRole': 'AGENT'
        },
        {
            'ChannelId': 1,
            'ParticipantRole': 'CUSTOMER'
        }
     ]
)

 while True:
   status = transcribe.get_call_analytics_job(CallAnalyticsJobName = job_name)
   if status['CallAnalyticsJob']['CallAnalyticsJobStatus'] in ['COMPLETED', 'FAILED']:
     break
   print("Not ready yet...")
   time.sleep(5)
 print(status)
```

Post-call analytics output

Post-call analytics transcripts are displayed in a turn-by-turn format by segment. They include call categorization, call characteristics (loudness scores, interruptions, non-talk time, talk speed), call summarization (issues, outcomes, and action items), redaction, and sentiment. Additionally, a summary of conversation characteristics is provided at the end of the transcript.

To increase accuracy and further customize your transcripts to your use case, such as including industry-specific terms, add custom vocabularies or custom language models to your Call Analytics request. To mask, remove, or tag words that you don't want in your transcription results, such as profanity, add vocabulary filtering. If you are unsure of the language code to be passed to the media file, you can enable batch language identification to automatically identify the language in your media file.

The following sections show examples of JSON output at an insight level. For compiled output, see Compiled post-call analytics output.

Call categorization

Here's what a category match looks like in your transcription output. This example shows that the audio from the 40040 millisecond timestamp to the 42460 millisecond timestamp is a match to the 'positive-resolution' category. In this case, the custom 'positive-resolution' category required a positive sentiment in last few seconds of speech.

```
"Categories": {
    "MatchedDetails": {
        "positive-resolution": {
            "PointsOfInterest": [
                {
                    "BeginOffsetMillis":  40040,
                    "EndOffsetMillis":  42460
                }
            ]
        }
    },
    "MatchedCategories": [
        " positive-resolution"
    ]
},
```

Call characteristics

Here's what call characteristics look like in your transcription output. Note that loudness scores are provided for each conversation turn, while all other characteristics are provided at the end of the transcript.

```
"LoudnessScores": [
    87.54,
    88.74,
    90.16,
    86.36,
    85.56,
    85.52,
    81.79,
    87.74,
    89.82
],

...

"ConversationCharacteristics": {
    "NonTalkTime": {
        "Instances": [],
        "TotalTimeMillis": 0
    },
    "Interruptions": {
        "TotalCount": 2,
        "TotalTimeMillis": 10700,
        "InterruptionsByInterrupter": {
            "AGENT": [
                {
                    "BeginOffsetMillis": 26040,
                    "DurationMillis": 5510,
                    "EndOffsetMillis": 31550
                }
            ],
            "CUSTOMER": [
                {
                    "BeginOffsetMillis": 770,
                    "DurationMillis": 5190,
                    "EndOffsetMillis": 5960
                }
            ]
```

```
        }
    },
    "TotalConversationDurationMillis": 42460,

    ...

    "TalkSpeed": {
        "DetailsByParticipant": {
            "AGENT": {
                "AverageWordsPerMinute": 150
            },
            "CUSTOMER": {
                "AverageWordsPerMinute": 167
            }
        }
    },
    "TalkTime": {
        "DetailsByParticipant": {
            "AGENT": {
                "TotalTimeMillis": 32750
            },
            "CUSTOMER": {
                "TotalTimeMillis": 18010
            }
        },
        "TotalTimeMillis": 50760
    }
},
```

Issues, Action Items and Next Steps

- In the following example, **issues** are identified as starting at character 7 and ending at character 51, which refers to this section of the text: "*I would like to cancel my recipe subscription*".

```
"Content": "Well, I would like to cancel my recipe subscription.",

"IssuesDetected": [
    {
        "CharacterOffsets": {
            "Begin": 7,
            "End": 51
        }
    }
```

```
],
```

- In the following example, **outcomes** are identified as starting at character 12 and ending at character 78, which refers to this section of the text: "*I made all changes to your account and now this discount is applied*".

```
"Content": "Wonderful. I made all changes to your account and now this discount is
 applied, please check.",

"OutcomesDetected": [
    {
        "CharacterOffsets": {
            "Begin": 12,
            "End": 78
        }
    }
],
```

- In the following example, **action items** are identified as starting at character 0 and ending at character 103, which refers to this section of the text: "*I will send an email with all the details to you today, and I will call you back next week to follow up*".

```
"Content": "I will send an email with all the details to you today, and I will call
 you back next week to follow up. Have a wonderful evening.",

"ActionItemsDetected": [
    {
        "CharacterOffsets": {
            "Begin": 0,
            "End": 103
        }
    }
],
```

Generative call summarization

Here's what generative call summarization looks like in your transcription output:

```
"ContactSummary": {
    "AutoGenerated": {
        "OverallSummary": {
```

```
            "Content": "A customer wanted to check to see if we had a bag allowance. We
 told them that we didn't have it, but we could add the bag from Canada to Calgary and
 then do the one coming back as well."
        }
    }
}
```

The analytics job will complete without summary generation in the following cases:

- Insufficient conversation content: The conversation must include at least one turn from both the agent and the customer. When there is insufficient conversation content, the service will return the error code INSUFFICIENT_CONVERSATION_CONTENT.
- Safety guardrails: The conversation must meet safety guardrails in place to ensure appropriate summary is generated. When these guardrails are not met, the service will return the error code FAILED_SAFETY_GUIDELINES.

The error code can be found in `Skipped` section within `AnalyticsJobDetails` in the output. You may also find the error reason in `CallAnalyticsJobDetails` in the `GetCallAnalyticsJob` API Response.

Sample Error Output

```
{
    "JobStatus": "COMPLETED",
    "AnalyticsJobDetails": {
        "Skipped": [
            {
                "Feature": "GENERATIVE_SUMMARIZATION",
                "ReasonCode": "INSUFFICIENT_CONVERSATION_CONTENT",
                "Message": "The conversation needs to have at least one turn from both
 the participants to generate summary"
            }
        ]
    },
    "LanguageCode": "en-US",
    "AccountId": "***************",
    "JobName": "Test2-copy",
    ...
}
```

Sentiment analysis

Here is what sentiment analysis looks like in your transcription output.

- Qualitative turn-by-turn sentiment values:

```
"Content": "That's very sad to hear. Can I offer you a 50% discount to have you stay
 with us?",

...

"BeginOffsetMillis": 12180,
"EndOffsetMillis": 16960,
"Sentiment": "NEGATIVE",
"ParticipantRole": "AGENT"

...

"Content": "That is a very generous offer. And I accept.",

...

"BeginOffsetMillis": 17140,
"EndOffsetMillis": 19860,
"Sentiment": "POSITIVE",
"ParticipantRole": "CUSTOMER"
```

- Quantitative sentiment values for the entire call:

```
"Sentiment": {
    "OverallSentiment": {
        "AGENT": 2.5,
        "CUSTOMER": 2.1
    },
```

- Quantitative sentiment values per participant and per call quarter:

```
"SentimentByPeriod": {
    "QUARTER": {
        "AGENT": [
            {
                "Score": 0.0,
                "BeginOffsetMillis": 0,
```

```
                "EndOffsetMillis": 9862
            },
            {
                "Score": -5.0,
                "BeginOffsetMillis": 9862,
                "EndOffsetMillis": 19725
            },
            {
                "Score": 5.0,
                "BeginOffsetMillis": 19725,
                "EndOffsetMillis": 29587
            },
            {
                "Score": 5.0,
                "BeginOffsetMillis": 29587,
                "EndOffsetMillis": 39450
            }
        ],
        "CUSTOMER": [
            {
                "Score": -2.5,
                "BeginOffsetMillis": 0,
                "EndOffsetMillis": 10615
            },
            {
                "Score": 5.0,
                "BeginOffsetMillis": 10615,
                "EndOffsetMillis": 21230
            },
            {
                "Score": 2.5,
                "BeginOffsetMillis": 21230,
                "EndOffsetMillis": 31845
            },
            {
                "Score": 5.0,
                "BeginOffsetMillis": 31845,
                "EndOffsetMillis": 42460
            }
        ]
    }
}
```

PII redaction

Here is what PII redaction looks like in your transcription output.

```
"Content": "[PII], my name is [PII], how can I help?",
"Redaction": [{
    "Confidence": "0.9998",
    "Type": "NAME",
    "Category": "PII"
}]
```

For more information, refer to Redacting PII in your batch job.

Language identification

Here is what Language Identification looks like in your transcription output if the feature is enabled.

```
"LanguageIdentification": [{
  "Code": "en-US",
  "Score": "0.8299"
}, {
  "Code": "en-NZ",
  "Score": "0.0728"
}, {
  "Code": "zh-TW",
  "Score": "0.0695"
}, {
  "Code": "th-TH",
  "Score": "0.0156"
}, {
  "Code": "en-ZA",
  "Score": "0.0121"
}]
```

In the above output example, Language Identification will populate the language codes with confidence scores. The result with the highest score will be selected as the language code for transcription. For mode details refer to Identifying the dominant languages in your media.

Compiled post-call analytics output

For brevity, some content is replaced with ellipses in the following transcription output.

This sample includes optional feature - Generative call summarization.

```
{
    "JobStatus": "COMPLETED",
    "LanguageCode": "en-US",
    "Transcript": [
        {
            "LoudnessScores": [
                78.63,
                78.37,
                77.98,
                74.18
            ],
            "Content": "[PII], my name is [PII], how can I help?",

            ...

             "Content": "Well, I would like to cancel my recipe subscription.",
             "IssuesDetected": [
                 {
                     "CharacterOffsets": {
                         "Begin": 7,
                         "End": 51
                     }
                 }
             ],

            ...

            "Content": "That's very sad to hear. Can I offer you a 50% discount to have
 you stay with us?",
            "Items": [
            ...
             ],
            "Id": "649afe93-1e59-4ae9-a3ba-a0a613868f5d",
            "BeginOffsetMillis": 12180,
            "EndOffsetMillis": 16960,
            "Sentiment": "NEGATIVE",
            "ParticipantRole": "AGENT"
        },
        {
            "LoudnessScores": [
                    80.22,
                    79.48,
```

```
                    82.81
            ],
            "Content": "That is a very generous offer. And I accept.",
            "Items": [
            ...
            ],
            "Id": "f9266cba-34df-4ca8-9cea-4f62a52a7981",
            "BeginOffsetMillis": 17140,
            "EndOffsetMillis": 19860,
            "Sentiment": "POSITIVE",
            "ParticipantRole": "CUSTOMER"
        },
        {

    ...

            "Content": "Wonderful. I made all changes to your account and now this
discount is applied, please check.",
            "OutcomesDetected": [
                {
                    "CharacterOffsets": {
                        "Begin": 12,
                        "End": 78
                    }
                }
            ],

            ...

            "Content": "I will send an email with all the details to you today, and I
will call you back next week to follow up. Have a wonderful evening.",
            "Items": [
            ...
            ],
            "Id": "78cd0923-cafd-44a5-a66e-09515796572f",
            "BeginOffsetMillis": 31800,
            "EndOffsetMillis": 39450,
            "Sentiment": "POSITIVE",
            "ParticipantRole": "AGENT"
        },
        {
           "LoudnessScores": [
              78.54,
              68.76,
```

```
                67.76
            ],
            "Content": "Thank you very much, sir. Goodbye.",
            "Items": [
            ...
            ],
            "Id": "5c5e6be0-8349-4767-8447-986f995af7c3",
            "BeginOffsetMillis": 40040,
            "EndOffsetMillis": 42460,
            "Sentiment": "POSITIVE",
            "ParticipantRole": "CUSTOMER"
        }
    ],

    ...

    "Categories": {
         "MatchedDetails": {
             "positive-resolution": {
                 "PointsOfInterest": [
                     {
                         "BeginOffsetMillis": 40040,
                         "EndOffsetMillis": 42460
                     }
                 ]
             }
         },
         "MatchedCategories": [
             "positive-resolution"
         ]
     },

     ...

     "ConversationCharacteristics": {
         "NonTalkTime": {
             "Instances": [],
             "TotalTimeMillis": 0
         },
         "Interruptions": {
             "TotalCount": 2,
             "TotalTimeMillis": 10700,
             "InterruptionsByInterrupter": {
                 "AGENT": [
```

```
                {
                    "BeginOffsetMillis": 26040,
                    "DurationMillis": 5510,
                    "EndOffsetMillis": 31550
                }
            ],
            "CUSTOMER": [
                {
                    "BeginOffsetMillis": 770,
                    "DurationMillis": 5190,
                    "EndOffsetMillis": 5960
                }
            ]
        }
    },
    "TotalConversationDurationMillis": 42460,
    "Sentiment": {
        "OverallSentiment": {
            "AGENT": 2.5,
            "CUSTOMER": 2.1
        },
        "SentimentByPeriod": {
            "QUARTER": {
                "AGENT": [
                    {
                        "Score": 0.0,
                        "BeginOffsetMillis": 0,
                        "EndOffsetMillis": 9862
                    },
                    {
                        "Score": -5.0,
                        "BeginOffsetMillis": 9862,
                        "EndOffsetMillis": 19725
                    },
                    {
                        "Score": 5.0,
                        "BeginOffsetMillis": 19725,
                        "EndOffsetMillis": 29587
                    },
                    {
                        "Score": 5.0,
                        "BeginOffsetMillis": 29587,
                        "EndOffsetMillis": 39450
                    }
```

```
                ],
                "CUSTOMER": [
                    {
                        "Score": -2.5,
                        "BeginOffsetMillis": 0,
                        "EndOffsetMillis": 10615
                    },
                    {
                        "Score": 5.0,
                        "BeginOffsetMillis": 10615,
                        "EndOffsetMillis": 21230
                    },
                    {
                        "Score": 2.5,
                        "BeginOffsetMillis": 21230,
                        "EndOffsetMillis": 31845
                    },
                    {
                        "Score": 5.0,
                        "BeginOffsetMillis": 31845,
                        "EndOffsetMillis": 42460
                    }
                ]
            }
        }
    },
    "TalkSpeed": {
        "DetailsByParticipant": {
            "AGENT": {
                "AverageWordsPerMinute": 150
            },
            "CUSTOMER": {
                "AverageWordsPerMinute": 167
            }
        }
    },
    "TalkTime": {
        "DetailsByParticipant": {
            "AGENT": {
                "TotalTimeMillis": 32750
            },
            "CUSTOMER": {
                "TotalTimeMillis": 18010
            }
```

```
            },
            "TotalTimeMillis": 50760
        },
        "ContactSummary": { // Optional feature - Generative call summarization
            "AutoGenerated": {
                "OverallSummary": {
                    "Content": "The customer initially wanted to cancel but the agent
 convinced them to stay by offering a 50% discount, which the customer accepted after
 reconsidering cancelling given the significant savings. The agent ensured the discount
 was applied and said they would follow up to ensure the customer remained happy with
 the revised subscription."
                }
            }
        }
    },
    "AnalyticsJobDetails": {
        "Skipped": []
    },
    ...
}
```

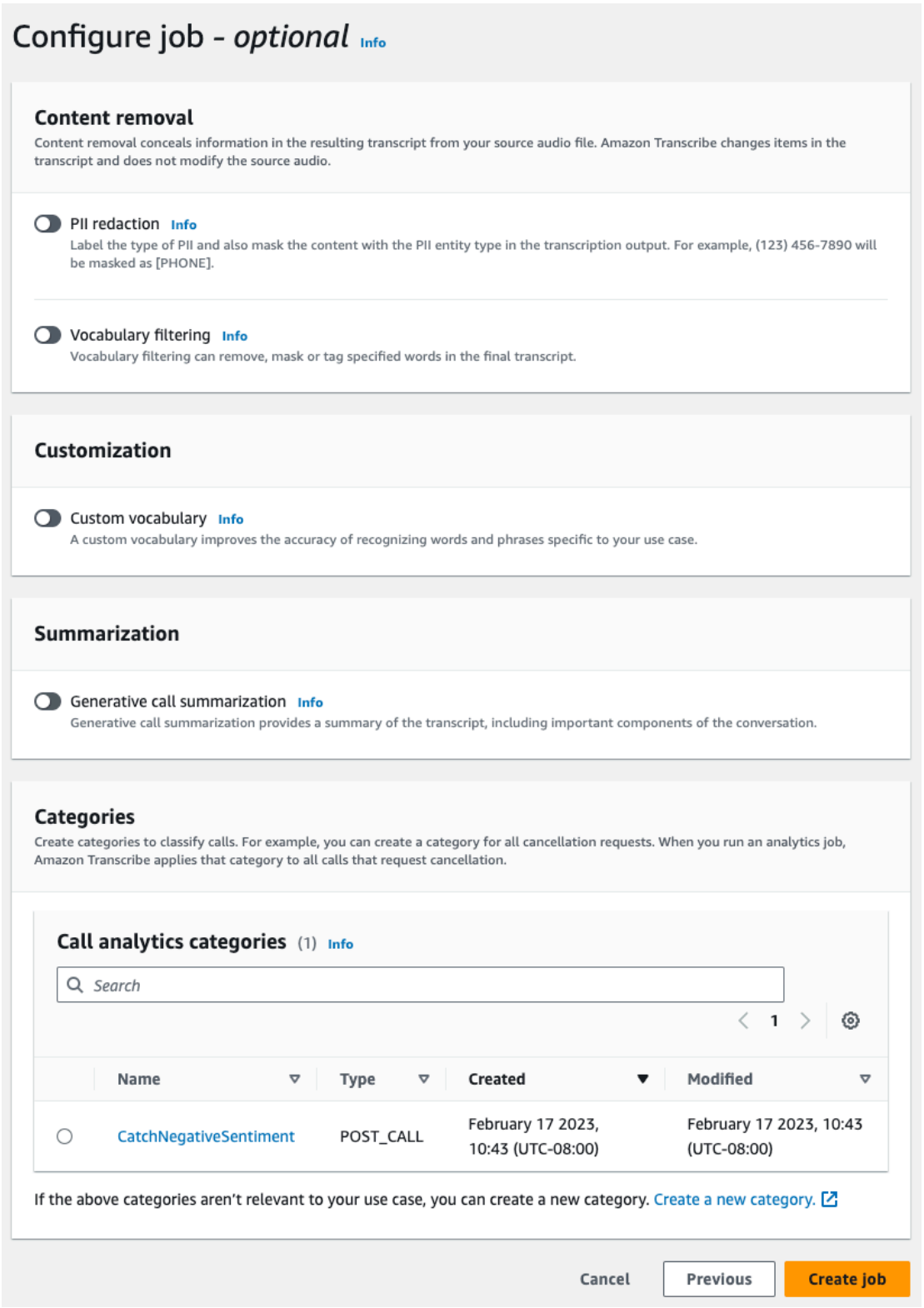
Enabling generative call summarization

> **Note**
>
> **Powered by Amazon Bedrock:** AWS implements automated abuse detection. Because post-contact summarization powered by generative AI is built on Amazon Bedrock, users can take full advantage of the controls implemented in Amazon Bedrock to enforce safety, security, and the responsible use of artificial intelligence (AI).

To use generative call summarization with a post call analytics job, see the following for examples:

AWS Management Console

In the Summarization panel, enable Generative call summarization to receive summary in the output.


Configure job - optional Info
Content removal
Content removal conceals information in the resulting transcript from your source audio file. Amazon Transcribe changes items in the transcript and does not modify the source audio.
PII redaction Info
Label the type of PII and also mask the content with the PII entity type in the transcription output. For example, (123) 456-7890 will be masked as [PHONE].
Vocabulary filtering Info
Vocabulary filtering can remove, mask or tag specified words in the final transcript.
Customization
Custom vocabulary Info
A custom vocabulary improves the accuracy of recognizing words and phrases specific to your use case.
Summarization
Generative call summarization Info
Generative call summarization provides a summary of the transcript, including important components of the conversation.
Categories
Create categories to classify calls. For example, you can create a category for all cancellation requests. When you run an analytics job, Amazon Transcribe applies that category to all calls that request cancellation.
Call analytics categories (1) Info
Search
1
Name
Type
Created
Modified
CatchNegativeSentiment
POST_CALL
February 17 2023, 10:43 (UTC-08:00)
February 17 2023, 10:43 (UTC-08:00)
If the above categories aren't relevant to your use case, you can create a new category. Create a new category.
Cancel
Previous
Create job

AWS CLI

This example uses the start-call-analytics-job command and `Settings` parameter with the `Summarization` sub-parameters. For more information, see `StartCallAnalyticsJob`.

```
aws transcribe start-call-analytics-job \
--region us-west-2 \
--call-analytics-job-name my-first-call-analytics-job \
--media MediaFileUri=s3://amzn-s3-demo-bucket/my-input-files/my-media-file.flac \
--output-location s3://amzn-s3-demo-bucket/my-output-files/ \
--data-access-role-arn arn:aws:iam::111122223333:role/ExampleRole \
--channel-definitions ChannelId=0,ParticipantRole=AGENT
 ChannelId=1,ParticipantRole=CUSTOMER
--settings '{"Summarization":{"GenerateAbstractiveSummary":true}}'
```

Here's another example using the start-call-analytics-job command, and a request body that enables summarization for that job.

```
aws transcribe start-call-analytics-job \
--region us-west-2 \
--cli-input-json file://filepath/my-call-analytics-job.json
```

The file *my-call-analytics-job.json* contains the following request body.

```
{
  "CallAnalyticsJobName": "my-first-call-analytics-job",
  "DataAccessRoleArn": "arn:aws:iam::111122223333:role/ExampleRole",
  "Media": {
    "MediaFileUri": "s3://amzn-s3-demo-bucket/my-input-files/my-media-file.flac"
  },
  "OutputLocation": "s3://amzn-s3-demo-bucket/my-output-files/",
  "ChannelDefinitions": [
    {
      "ChannelId": 0,
      "ParticipantRole": "AGENT"
    },
    {
```

```
      "ChannelId": 1,
      "ParticipantRole": "CUSTOMER"
    }
  ],
  "Settings": {
    "Summarization":{
      "GenerateAbstractiveSummary": true
    }
  }
}
```

AWS SDK for Python (Boto3)

This example uses the AWS SDK for Python (Boto3) to start a Call Analytics with summarization enabled using the start_call_analytics_job method. For more information, see StartCallAnalyticsJob.

For additional examples using the AWS SDKs, including feature-specific, scenario, and cross-service examples, refer to the Code examples for Amazon Transcribe using AWS SDKs chapter.

```
from __future__ import print_function
from __future__ import print_function
import time
import boto3
transcribe = boto3.client('transcribe', 'us-west-2')
job_name = "my-first-call-analytics-job"
job_uri = "s3://amzn-s3-demo-bucket/my-input-files/my-media-file.flac"
output_location = "s3://amzn-s3-demo-bucket/my-output-files/"
data_access_role = "arn:aws:iam::111122223333:role/ExampleRole"
transcribe.start_call_analytics_job(
  CallAnalyticsJobName = job_name,
  Media = {
    'MediaFileUri': job_uri
  },
  DataAccessRoleArn = data_access_role,
  OutputLocation = output_location,
  ChannelDefinitions = [
    {
      'ChannelId': 0,
      'ParticipantRole': 'AGENT'
    },
    {
```

```
      'ChannelId': 1,
      'ParticipantRole': 'CUSTOMER'
    }
  ],
  Settings = {
    "Summarization":
      {
        "GenerateAbstractiveSummary": true
      }
  }
)

while True:
  status = transcribe.get_call_analytics_job(CallAnalyticsJobName = job_name)
  if status['CallAnalyticsJob']['CallAnalyticsJobStatus'] in ['COMPLETED', 'FAILED']:
    break
  print("Not ready yet...")
  time.sleep(5)
print(status)
```

Real-time Call Analytics

Real-time Call Analytics provides real-time insights that can be used for addressing issues and mitigating escalations as they happen.

The following insights are available with real-time Call Analytics:

- Category events that use rules to flag specific keywords and phrases; category events can be used to create real-time alerts
- Issue detection identifies the issues spoken within each audio segment
- PII (sensitive data) identification in your text transcript
- PII (sensitive data) redaction of your text transcript
- Sentiment analysis for each speech segment
- Language identification detects the primary language spoken in each audio channel

In addition to real-time Call Analytics, Amazon Transcribe can also perform post-call analytics on your media stream. You can include post-call analytics in your real-time Call Analytics request using the `PostCallAnalyticsSettings` parameter.

Real-time insights

This section details the insights available for real-time Call Analytics transcriptions.

Category events

Using category events, you can match your transcription based on an exact keyword or phrase. For example, if you set a filter for the phrase "I want to speak to the manager", Amazon Transcribe filters for that *exact* phrase.

Here's an output example.

For more information on creating real-time Call Analytics categories, see Creating categories for real-time transcriptions.

> **Tip**
>
> Category events allow you to set real-time alerts; see Creating real-time alerts for category matches for more information.

Issue detection

Issue detection provides succinct summaries of detected issues within each audio segment. Using the issue detection feature, you can:

- Reduce the need for manual note-taking during and after calls
- Improve agent efficiency, allowing them to respond faster to customers

> **Note**
>
> Issue detection is supported with these English language dialects: Australian (`en-AU`), British (`en-GB`), and US (`en-US`).

The issue detection feature works across all industries and business sectors, and is context-based. It works out-of-the-box and thus doesn't support customization, such as model training or custom categories.

Issue detection with real-time Call Analytics is performed on each complete audio segment.

Here's an output example.

PII (sensitive data) identification

Sensitive data identification labels personally identifiable information (PII) in the text transcript. This parameter is useful for protecting customer information.

> **Note**
>
> Real-time PII identification is supported with these English language dialects: Australian (en-AU), British (en-GB), US (en-US) and with Spanish language dialect (es-US).

PII identification with real-time Call Analytics is performed on each complete audio segment.

To view the list of PII that is identified using this feature, or to learn more about PII identification with Amazon Transcribe, see Redacting or identifying personally identifiable information.

Here is an output example.

PII (sensitive data) redaction

Sensitive data redaction replaces personally identifiable information (PII) in your text transcript with the type of PII (for example, [NAME]). This parameter is useful for protecting customer information.

> **Note**
>
> Real-time PII redaction is supported with these English language dialects: Australian (en-AU), British (en-GB), US (en-US) and with Spanish language dialect (es-US).

PII redaction with real-time Call Analytics is performed on each complete audio segment.

To view the list of PII that is redacted using this feature, or to learn more about redaction with Amazon Transcribe, see Redacting or identifying personally identifiable information.

Here is an output example.

Sentiment analysis

Sentiment analysis estimates how the customer and agent are feeling throughout the call. This metric is provided for every speech segment and is represented as a qualitative value (`positive`, `neutral`, `mixed`, or `negative`).

Using this parameter, you can qualitatively evaluate the overall sentiment for each call participant and the sentiment for each participant during each speech segment. This metric can help identify if your agent is able to delight an upset customer by the time the call ends.

Sentiment analysis with real-time Call Analytics is performed on each complete audio segment.

Sentiment analysis works out-of-the-box and thus doesn't support customization, such as model training or custom categories.

Here's an output example.

Language identification

Language identification automatically recognizes and determines the primary language being spoken within each channel of your audio streams during real-time Call Analytics. Once identified, Call Analytics will process and return the most suitable transcription based on the detected language, delivering this information back through the stream in real-time.

This feature allows you to automatically recognize and identify the dominant language spoken in each channel of your audio stream. Once the language is detected, Call Analytics processes and delivers the appropriate transcription for the identified language in real-time.

Automatic language identification is supported for all Call Analytics streaming supported languages that are currently supported for streaming transcriptions at no additional cost, and is available in the Call Analytics streaming supported AWS Regions.

> **Important**
>
> Call Analytics supports single-language identification only, which identifies the dominant language spoken in your audio channel. Multi-language identification is not supported, meaning each channel can only be transcribed in one language.
> To use language identification, you must provide at least two language codes and at most five language codes, and you can select only one language dialect per language per stream from supported Call Analytics streaming languages. This means that you cannot select en-

US and en-AU as language options for the same transcription. When utilizing this feature, the LanguageCode parameter must remain null in the request, as LanguageCode and IdentifyLanguage are mutually exclusive options.

Warning

If your specified language codes don't match the actual spoken language, the system will select the most similar language from your options, which may result in inaccurate transcriptions.

Using the language identification feature, you can:

- Automatically detect the dominant language in real-time
- Process different languages across separate channels
- Receive confidence scores for language detection
- Apply language-specific custom vocabularies

To use language identification, you must configure the following parameters:

Required parameters:

- `identifyLanguage` - Set to true to enable language identification.
- `languageOptions` - A list of possible language codes to use when identifyLanguage is set to true. You must provide a minimum of two language selections, as single-language selection is not supported.

Optional parameters:

- `preferredLanguage` - Your expected primary language from provided languageOptions. Adding a preferred language can help Call Analytics identify the language faster.
- `vocabularyNames` - Custom vocabulary names for improved accuracy. Note that vocabulary names are case-sensitive, and if the custom vocabulary's language doesn't match the identified media language, it won't be applied to the transcription.
- `vocabularyFilterNames` - Vocabulary filter names to customize the transcript output.

Here's an output example.

Creating categories for real-time transcriptions

Real-time Call Analytics supports the creation of custom categories, which you can use to tailor your transcript analyses to best suit your specific business needs.

You can create as many categories as you like to cover a range of different scenarios. For each category you create, you must create between 1 and 20 rules. Real-time Call Analytics transcriptions only support rules that use `TranscriptFilter` (keyword matches). For more detail on using rules with the `CreateCallAnalyticsCategory` operation, refer to the Rule criteria for real-time Call Analytics categories section.

If the content in your media matches all the rules you've specified in a given category, Amazon Transcribe labels your output with that category. See category event output for an example of a category match in JSON output format.

Here are a few examples of what you can do with custom categories:

- Identify issues that warrant immediate attention by flagging and tracking specific sets of keywords
- Monitor compliance, such as an agent speaking (or omitting) a specific phrase
- Flag specific words and phrases in real time; you can then set your category match to set an immediate alert. For example, if you create a real-time Call Analytics category for a customer saying "*speak to a manager*," you can set an event alert for this real-time category match that notifies the on-duty manager.

Post-call versus real-time categories

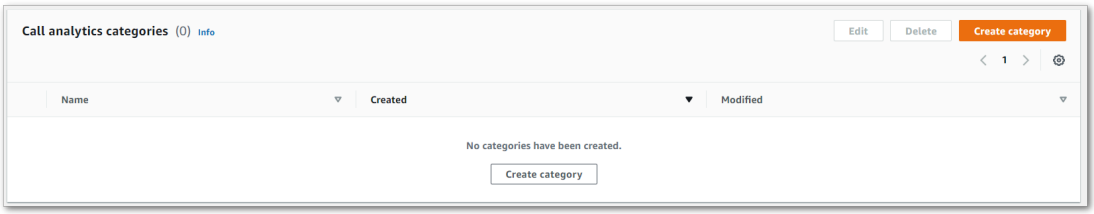
When creating a new category, you can specify whether you want it created as a post-call category (`POST_CALL`) or as a real-time category (`REAL_TIME`). If you don't specify an option, your category is created as a post-call category by default. Real-time category matches can be used to create real-time alerts. For more information, see Creating real-time alerts for category matches.

To create a new category for real-time Call Analytics, you can use the **AWS Management Console**, **AWS CLI**, or **AWS SDKs**; see the following for examples:

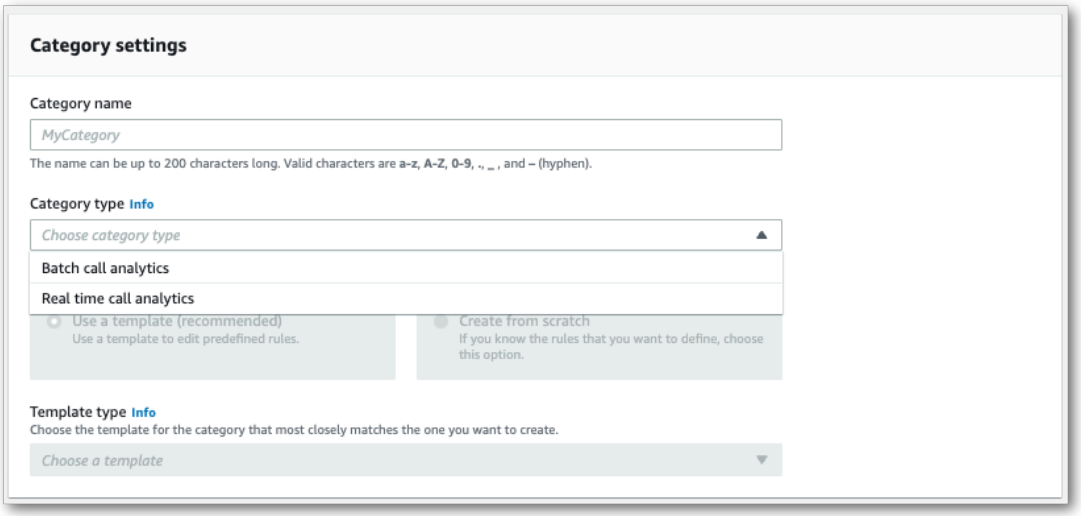
AWS Management Console

1. In the navigation pane, under Amazon Transcribe, choose **Amazon Transcribe Call Analytics**.

2. Choose **Call analytics categories**, which takes you to the **Call analytics categories** page. Select the **Create category** button.

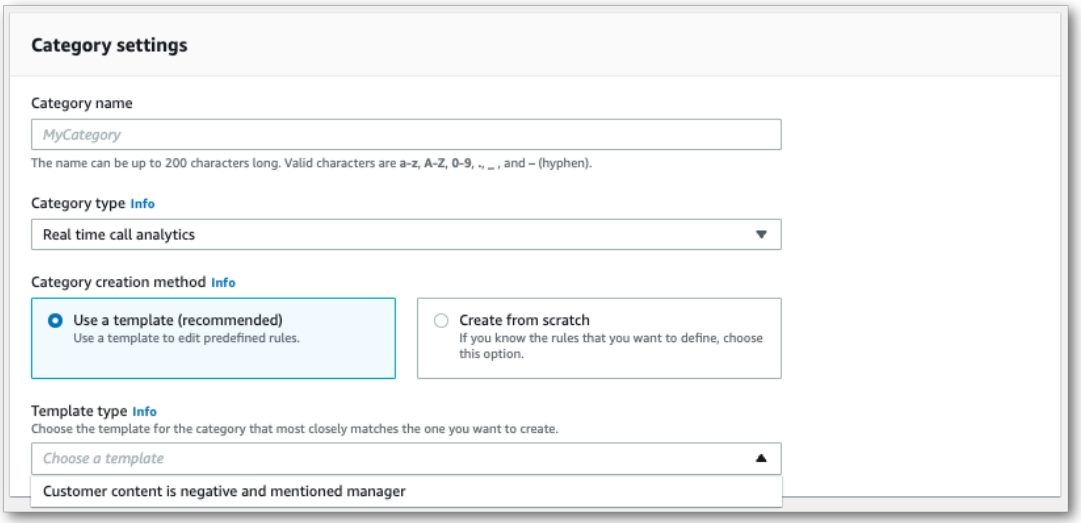


3. You're now on the **Create category page**. Enter a name for your category, then choose 'Real time call analytics' in the **Category type** dropdown menu.

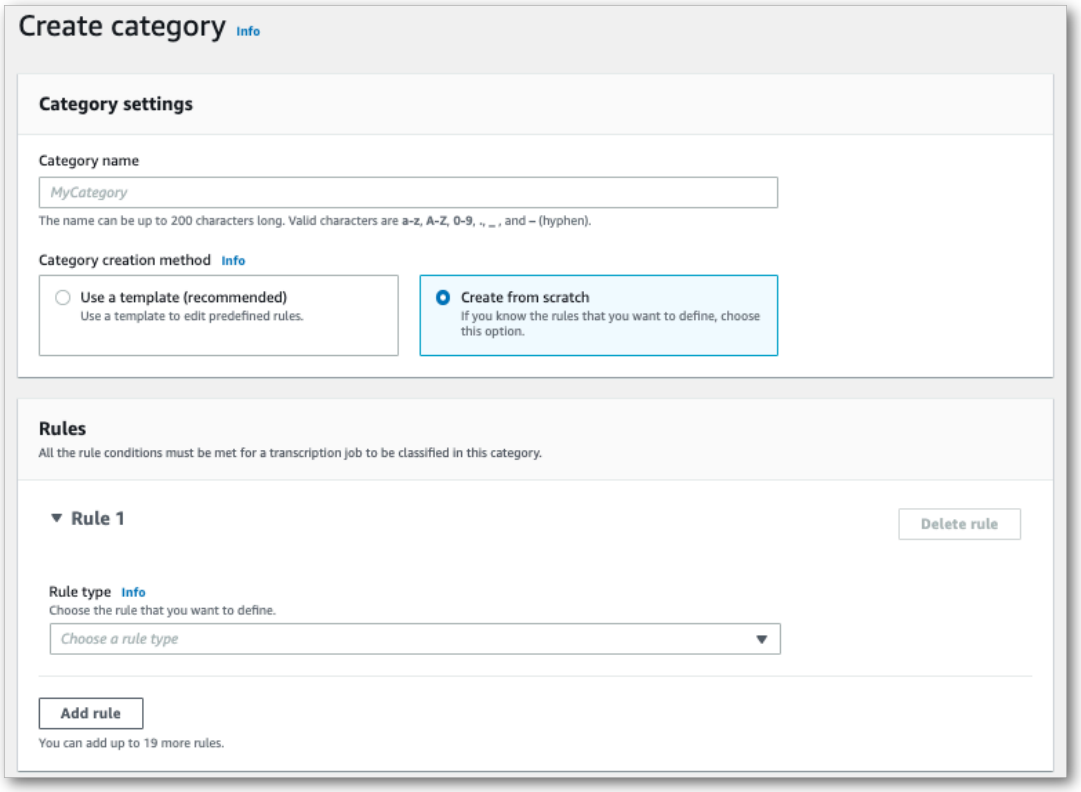


4. You can choose a template to create your category or you can make one from scratch.

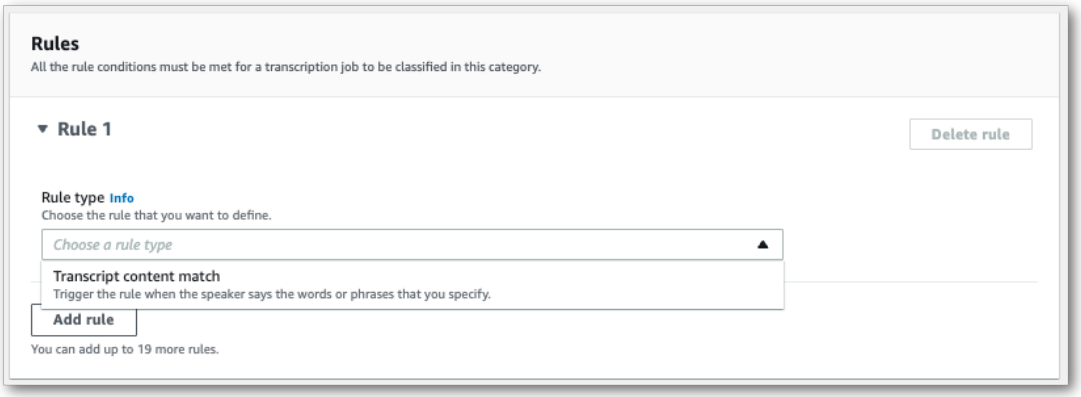
   If using a template: select **Use a template (recommended)**, choose the template you want, then select **Create category**.



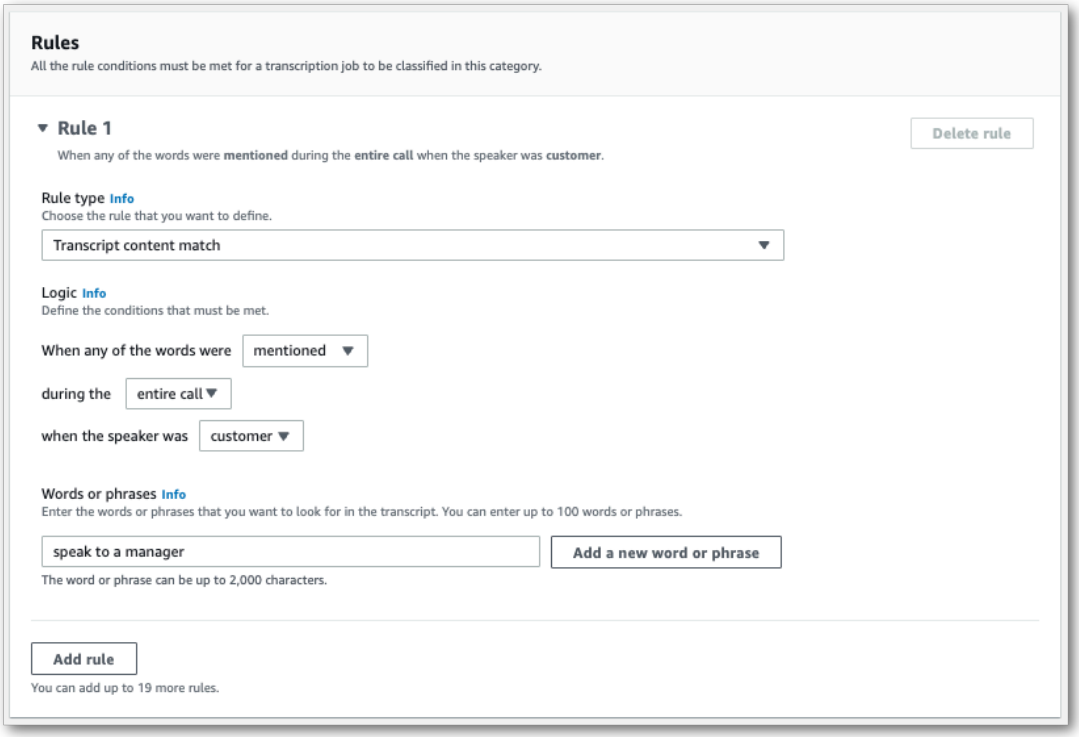
5. If creating a custom category: select **Create from scratch**.



6. Add rules to your category using the dropdown menu. You can add up to 20 rules per category. With real-time Call Analytics transcriptions, you can only include rules that involve transcript content matches. Any matches are flagged in real time.



7. Here's an example of a category with one rule: a customer who says "speak to a manager" at any point in the call.



8. When you're finished adding rules to your category, choose **Create category**.

AWS CLI

This example uses the create-call-analytics-category command. For more information, see CreateCallAnalyticsCategory, CategoryProperties, and Rule.

The following example creates a category with the rule:

- The customer spoke the phrase "speak to the manager" at any point in the call.

This example uses the create-call-analytics-category command, and a request body that adds a rule to your category.

```
aws transcribe create-call-analytics-category \
--cli-input-json file://filepath/my-first-analytics-category.json
```

The file *my-first-analytics-category.json* contains the following request body.

```
{
  "CategoryName": "my-new-real-time-category",
  "InputType": "REAL_TIME",
```

```
  "Rules": [
        {
            "TranscriptFilter": {
                "Negate": false,
                "Targets": [
                    "speak to the manager"
                ],
                "TranscriptFilterType": "EXACT"
            }
        }
    ]
}
```

AWS SDK for Python (Boto3)

This example uses the AWS SDK for Python (Boto3) to create a category using the `CategoryName` and `Rules` arguments for the create_call_analytics_category method. For more information, see `CreateCallAnalyticsCategory`, `CategoryProperties`, and `Rule`.

For additional examples using the AWS SDKs, including feature-specific, scenario, and cross-service examples, refer to the Code examples for Amazon Transcribe using AWS SDKs chapter.

The following example creates a category with the rule:

- The customer spoke the phrase "speak to the manager" at any point in the call.

```
from __future__ import print_function
import time
import boto3
transcribe = boto3.client('transcribe', 'us-west-2')
category_name = "my-new-real-time-category"
transcribe.create_call_analytics_category(
    CategoryName = category_name,
    InputType = "REAL_TIME",
    Rules = [
        {
            'TranscriptFilter': {
                'Negate': False,
                'Targets': [
                    'speak to the manager'
                ],
                'TranscriptFilterType': 'EXACT'
```

```
            }
        }
    ]
)

result = transcribe.get_call_analytics_category(CategoryName = category_name)
print(result)
```

Rule criteria for real-time Call Analytics categories

This section outlines the types of custom `REAL_TIME` rules that you can create using the `CreateCallAnalyticsCategory` API operation.

Issue detection occurs automatically, so you don't need to create any rules or categories to flag issues.

Note that only keyword matches are supported for real-time Call Analytics transcriptions. If you want to create categories that include interruptions, silence, or sentiment, refer to Rule criteria for post-call analytics categories.

Keyword match

Rules using keywords (`TranscriptFilter` data type) are designed to match:

- Custom words or phrases spoken by the agent, the customer, or both
- Custom words or phrases **not** spoken by the agent, the customer, or both
- Custom words or phrases that occur in a specific time frame

Here's an example of the parameters available with `TranscriptFilter`:

```
"TranscriptFilter": {
    "AbsoluteTimeRange": {
       Specify the time frame, in milliseconds, when the match should occur
    },
    "RelativeTimeRange": {
       Specify the time frame, in percentage, when the match should occur
    },
    "Negate": Specify if you want to match the presence or absence of your custom
 keywords,
    "ParticipantRole": Specify if you want to match speech from the agent, the
 customer, or both,
```

```
    "Targets": [ The custom words and phrases you want to match ],
    "TranscriptFilterType": Use this parameter to specify an exact match for the
 specified targets
}
```

Refer to `CreateCallAnalyticsCategory` and `TranscriptFilter` for more information on these parameters and the valid values associated with each.

Post-call analytics with real-time transcriptions

Post-call analytics is an optional feature available with real-time Call Analytics transcriptions. In addition to the standard real-time analytics insights, post-call analytics provides you with the following:

- **Action items**: Lists any action items identified in the call
- **Interruptions**: Measures if and when one participant cuts off the other participant midsentence
- **Issues**: Provides the issues identified in the call
- **Loudness**: Measures the volume at which each participant is speaking
- **Non-talk time**: Measures periods of time that do not contain speech
- **Outcomes**: Provides the outcome, or resolution, identified in the call
- **Talk speed**: Measures the speed at which both participants are speaking
- **Talk time**: Measures the amount of time (in milliseconds) each participant spoke during the call

When enabled, post-call analytics from an audio stream produces a transcript similar to a post-call analytics from an audio file and stores it in the Amazon S3 bucket specified in `OutputLocation`. Additionally, post-call analytics records your audio stream and saves it as an audio file (`WAV` format) in the same Amazon S3 bucket. If you enable redaction, a redacted transcript and a redacted audio file are also stored in the specified Amazon S3 bucket. Enabling post-call analytics with your audio stream produces between two and four files, as described here:

- If redaction is **not** enabled, your output files are:
  1. An unredacted transcript
  2. An unredacted audio file
- If redaction is enabled **without** the unredacted option (`redacted`), your output files are:
  1. A redacted transcript
  2. A redacted audio file

- If redaction is enabled **with** the unredacted option (`redacted_and_unredacted`), your output files are:
  1. A redacted transcript
  2. A redacted audio file
  3. An unredacted transcript
  4. An unredacted audio file

Note that if you enable post-call analytics (`PostCallAnalyticsSettings`) with your request, and you're using `FLAC` or `OPUS-OGG` media, you **do not** get `loudnessScore` in your transcript and no audio recordings of your stream are created. Transcribe may also not be able to provide post-call analytics for long-running audio streams lasting longer than 90 minutes.

For more information on the insights available with post-call analytics for audio streams, refer to the post-call analytics insights section.

> **Tip**
>
> If you enable post-call analytics with your real-time Call Analytics request, all of your `POST_CALL` and `REAL-TIME` categories are applied to your post-call analytics transcription.

Enabling post-call analytics

To enable post-call analytics, you must include the `PostCallAnalyticsSettings` parameter in your real-time Call Analytics request. The following parameters must be included when `PostCallAnalyticsSettings` is enabled:

- `OutputLocation`: The Amazon S3 bucket where you want your post-call transcript stored.
- `DataAccessRoleArn`: The Amazon Resource Name (ARN) of the Amazon S3 role that has permissions to access the specified Amazon S3 bucket. Note that you must also use the Trust policy for real-time analytics.

If you want a redacted version of your transcript, you can include `ContentRedactionOutput` or `ContentRedactionType` in your request. For more information on these parameters, see `StartCallAnalyticsStreamTranscription` in the API Reference.

To start a real-time Call Analytics transcription with post-call analytics enabled, you can use the **AWS Management Console** (demo only), **HTTP/2**, or **WebSockets**. For examples, see Starting a real-time Call Analytics transcription.

> **Important**
>
> Currently, the AWS Management Console only offers a demo for real-time Call Analytics with pre-loaded audio examples. If you want to use your own audio, you must use the API (HTTP/2, WebSockets, or an SDK).

Example post-call analytics output

Post-call transcripts are displayed in a turn-by-turn format by segment. They include call characteristics, sentiment, call summarization, issue detection, and (optionally) PII redaction. If any of your post-call categories are a match to the audio content, these are also present in your output.

To increase accuracy and further customize your transcripts to your use case, such as including industry-specific terms, add custom vocabularies or custom language models to your Call Analytics request. To mask, remove, or tag words you don't want in your transcription results, such as profanity, add vocabulary filtering.

Here is a compiled post-call analytics output example:


```
{
    "JobStatus": "COMPLETED",
    "LanguageCode": "en-US",
    "AccountId": "1234567890",
    "Channel": "VOICE",
    "Participants": [{
        "ParticipantRole": "AGENT"
    },
    {
        "ParticipantRole": "CUSTOMER"
    }],
    "SessionId": "12a3b45c-de6f-78g9-0123-45h6ab78c901",
    "ContentMetadata": {
        "Output": "Raw"
    }
    "Transcript": [{
        "LoudnessScores": [
```

```
            78.63,
            78.37,
            77.98,
            74.18
        ],
        "Content": "[PII], my name is [PII], how can I help?",

            ...

        "Content": "Well, I would like to cancel my recipe subscription.",
            "IssuesDetected": [{
                "CharacterOffsets": {
                    "Begin": 7,
                    "End": 51
                }
            }],

            ...

        "Content": "That's very sad to hear. Can I offer you a 50% discount to have you
stay with us?",
        "Id": "649afe93-1e59-4ae9-a3ba-a0a613868f5d",
        "BeginOffsetMillis": 12180,
        "EndOffsetMillis": 16960,
        "Sentiment": "NEGATIVE",
        "ParticipantRole": "AGENT"
    },
    {
        "LoudnessScores": [
            80.22,
            79.48,
            82.81
        ],
        "Content": "That is a very generous offer. And I accept.",
        "Id": "f9266cba-34df-4ca8-9cea-4f62a52a7981",
        "BeginOffsetMillis": 17140,
        "EndOffsetMillis": 19860,
        "Sentiment": "POSITIVE",
        "ParticipantRole": "CUSTOMER"
    },
            ...

        "Content": "Wonderful. I made all changes to your account and now this discount
is applied, please check.",
```

```
        "OutcomesDetected": [{
        "CharacterOffsets": {
            "Begin": 12,
            "End": 78
        }
        }],

            ...

        "Content": "I will send an email with all the details to you today, and I will
 call you back next week to follow up. Have a wonderful evening.",
        "Id": "78cd0923-cafd-44a5-a66e-09515796572f",
        "BeginOffsetMillis": 31800,
        "EndOffsetMillis": 39450,
        "Sentiment": "POSITIVE",
        "ParticipantRole": "AGENT"
    },
    {
        "LoudnessScores": [
            78.54,
            68.76,
            67.76
        ],
        "Content": "Thank you very much, sir. Goodbye.",
        "Id": "5c5e6be0-8349-4767-8447-986f995af7c3",
        "BeginOffsetMillis": 40040,
        "EndOffsetMillis": 42460,
        "Sentiment": "POSITIVE",
        "ParticipantRole": "CUSTOMER"
    }
    ],

    ...

    "Categories": {
        "MatchedDetails": {
            "positive-resolution": {
                "PointsOfInterest": [{
                    "BeginOffsetMillis": 40040,
                    "EndOffsetMillis": 42460
                }]
            }
        },
        "MatchedCategories": [
```

```
            "positive-resolution"
        ]
    },

    ...

    "ConversationCharacteristics": {
        "NonTalkTime": {
            "Instances": [],
            "TotalTimeMillis": 0
        },
        "Interruptions": {
            "TotalCount": 2,
            "TotalTimeMillis": 10700,
            "InterruptionsByInterrupter": {
                "AGENT": [{
                    "BeginOffsetMillis": 26040,
                    "DurationMillis": 5510,
                    "EndOffsetMillis": 31550
                }],
                "CUSTOMER": [{
                    "BeginOffsetMillis": 770,
                    "DurationMillis": 5190,
                    "EndOffsetMillis": 5960
                }]
            }
        },
        "TotalConversationDurationMillis": 42460,
        "Sentiment": {
            "OverallSentiment": {
                "AGENT": 2.5,
                "CUSTOMER": 2.1
            },
            "SentimentByPeriod": {
                "QUARTER": {
                    "AGENT": [{
                        "Score": 0.0,
                        "BeginOffsetMillis": 0,
                        "EndOffsetMillis": 9862
                    },
                    {
                        "Score": -5.0,
                        "BeginOffsetMillis": 9862,
                        "EndOffsetMillis": 19725
```

```
                },
                {
                    "Score": 5.0,
                    "BeginOffsetMillis": 19725,
                    "EndOffsetMillis": 29587
                },
                {
                   "Score": 5.0,
                    "BeginOffsetMillis": 29587,
                    "EndOffsetMillis": 39450
                }
                ],
                "CUSTOMER": [{
                    "Score": -2.5,
                    "BeginOffsetMillis": 0,
                    "EndOffsetMillis": 10615
                },
                {
                    "Score": 5.0,
                    "BeginOffsetMillis": 10615,
                    "EndOffsetMillis": 21230
                },
                {
                    "Score": 2.5,
                    "BeginOffsetMillis": 21230,
                    "EndOffsetMillis": 31845
                },
                {
                    "Score": 5.0,
                    "BeginOffsetMillis": 31845,
                    "EndOffsetMillis": 42460
                }
                ]
            }
        }
    },
    "TalkSpeed": {
        "DetailsByParticipant": {
            "AGENT": {
                "AverageWordsPerMinute": 150
            },
            "CUSTOMER": {
                "AverageWordsPerMinute": 167
            }
```

```
            }
        },
        "TalkTime": {
            "DetailsByParticipant": {
                "AGENT": {
                    "TotalTimeMillis": 32750
                },
                "CUSTOMER": {
                    "TotalTimeMillis": 18010
                }
            },
            "TotalTimeMillis": 50760
        }
    },
    ...
}
```

Starting a real-time Call Analytics transcription

Before starting a real-time Call Analytics transcription, you must create all the categories you want Amazon Transcribe to match in your call.

> **Note**
>
> Call Analytics transcripts can't be retroactively matched to new categories. Only the categories you create *before* starting a Call Analytics transcription can be applied to that transcription output.

If you've created one or more categories, and your audio matches all the rules within at least one of your categories, Amazon Transcribe flags your output with the matching categories. If you choose not to use categories, or if your audio doesn't match the rules specified in your categories, your transcript isn't flagged.

To include post-call analytics with your real-time Call Analytics transcription, you must provide an Amazon S3 bucket in your request using the `OutputLocation` parameter. You must also include a `DataAccessRoleArn` that has write permissions to the specified bucket. A separate transcript is produced and stored in the specified bucket upon completion of your real-time Call Analytics streaming session.

With real-time Call Analytics, you also have the option to create real-time category alerts; refer to Creating real-time alerts for category matches for instructions.

To start a real-time Call Analytics transcription, you can use the **AWS Management Console**, **HTTP/2**, or **WebSockets**; see the following for examples:

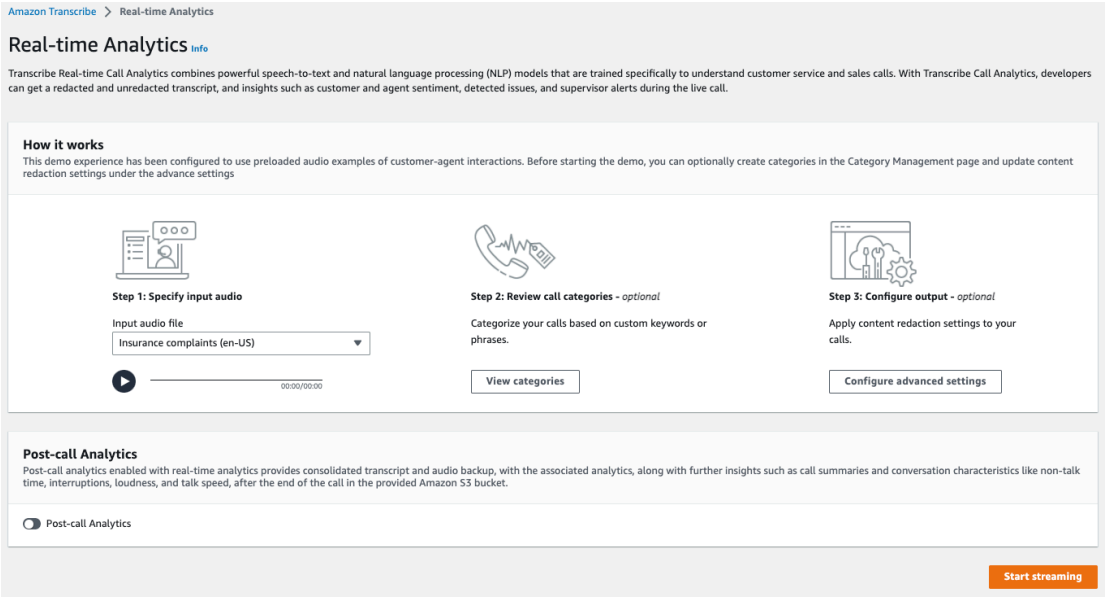
> **Important**
>
> Currently, the AWS Management Console only offers a demo for real-time Call Analytics with pre-loaded audio examples. If you want to use your own audio, you must use the API (HTTP/2, WebSockets, or an SDK).

AWS Management Console

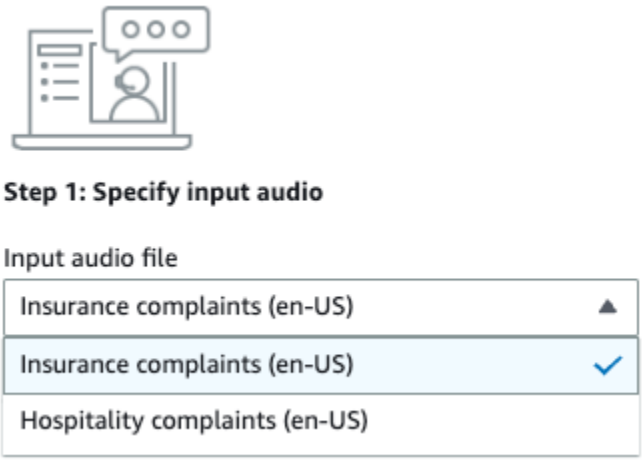
Use the following procedure to start a Call Analytics request. The calls that match all characteristics defined by a category are labeled with that category.

> **Note**
>
> Only a demo is available in the AWS Management Console. To start a custom real-time analytics transcription, you must use the API.

1. In the navigation pane, under Amazon Transcribe Call Analytics, choose **Analyze a real-time call**.

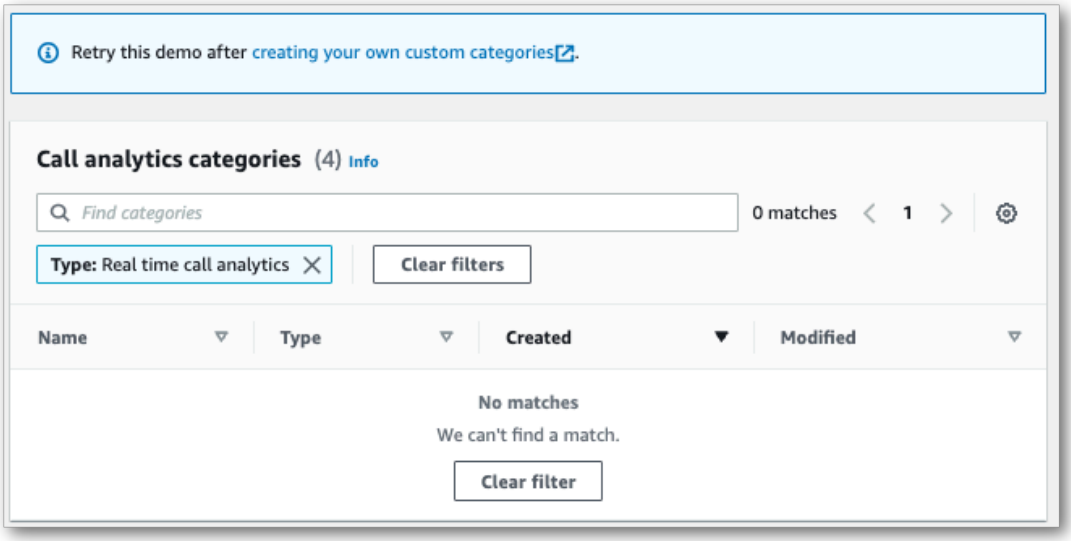


2. For **Step 1: Specify input audio**, choose a demo test file from the dropdown menu.



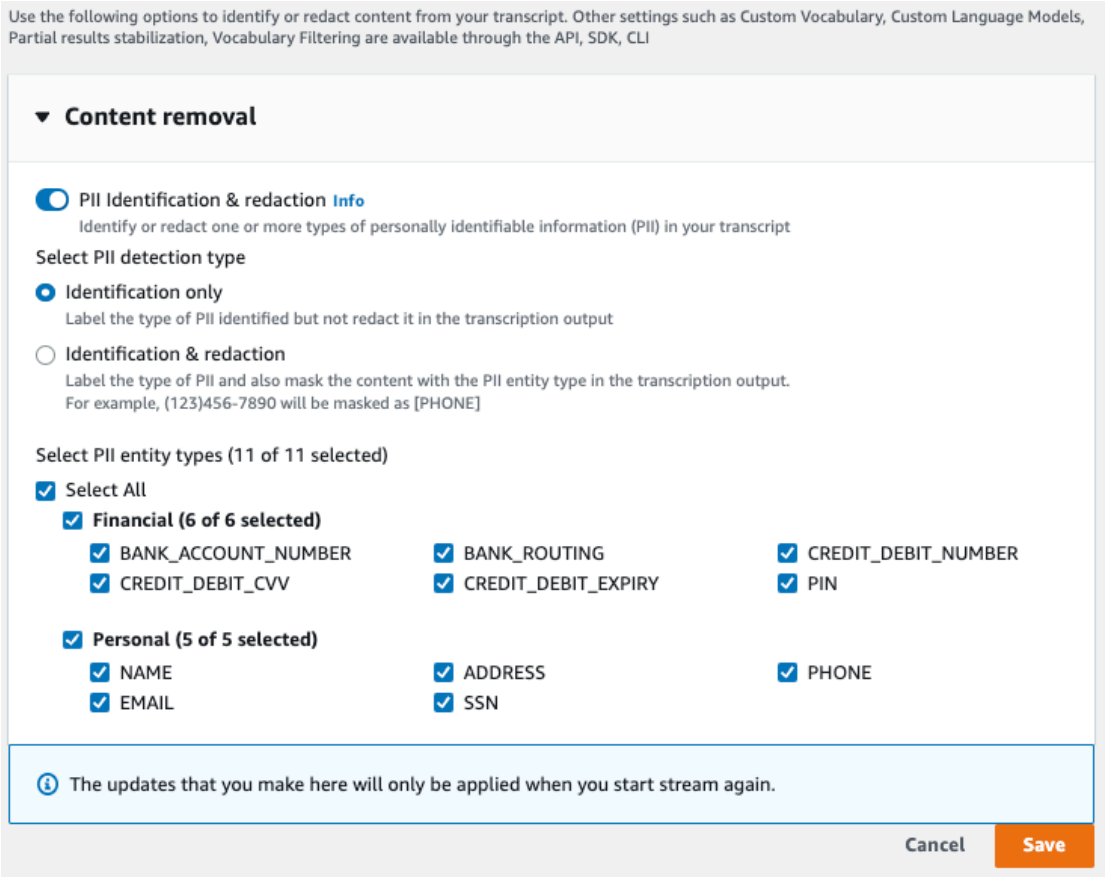
3. For **Step 2: Review call categories**, you have the option to review the real-time Call Analytics categories you previously created. All real-time Call Analytics categories are applied to your transcription.

   Choosing **View categories** opens a new pane that shows your existing real-time Call Analytics categories and provides a link to create new ones.



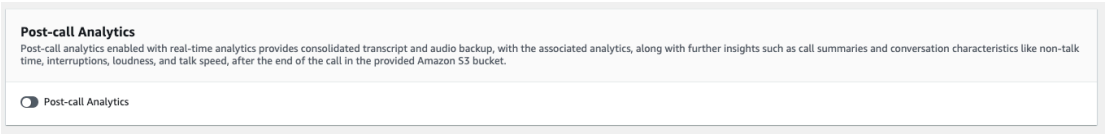
4. For **Step 3: Configure input and output**, you have the option to apply additional settings.

   Choosing **Configure advanced settings** opens a new pane where you can specify content redaction settings.

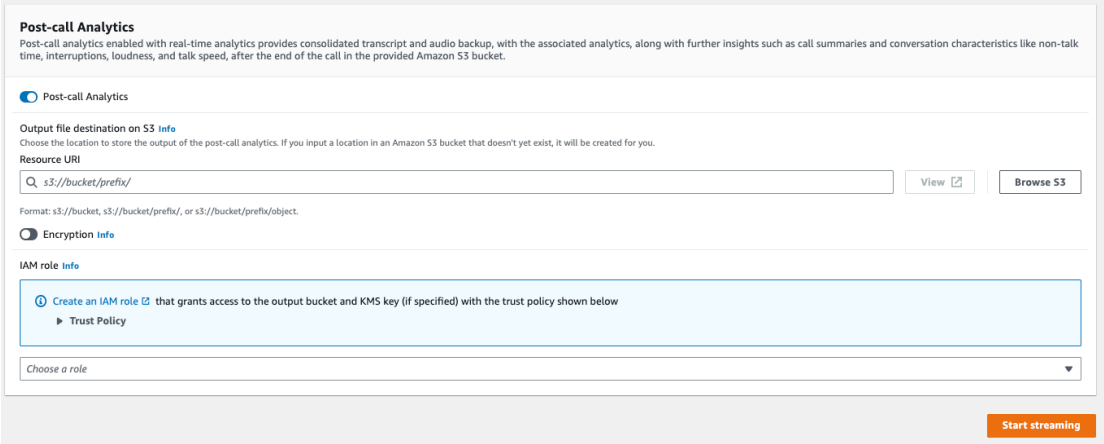


Once you've made all your selections, choose **Save** to return to the main page.

5. To apply additional analytics, you can toggle on **Post-call Analytics**. This provides you with the same analytics as a post-call analytics transcription, including interruptions, loudness, non-talk time, talk speed, talk time, issues, action items, and outcomes. Post-call analytics output is stored in a separate file from your real-time Call Analytics transcript.



If you apply post-call analytics, you must specify an Amazon S3 output file destination and an IAM role. You can optionally choose to encrypt your output.



6. Choose **Start streaming**.

HTTP/2 stream

This example creates an HTTP/2 request with Call Analytics enabled. For more information on using HTTP/2 streaming with Amazon Transcribe, see Setting up an HTTP/2 stream. For more detail on parameters and headers specific to Amazon Transcribe, see `StartCallAnalyticsStreamTranscription`.

This example includes post-call analytics. If you don't want post-call analytics, remove the `PostCallAnalyticsSettings` section from the request.

Note that the configuration event shown in the following example needs to be passed as the first event in the stream.

```
POST /stream-transcription HTTP/2
host: transcribestreaming.us-west-2.amazonaws.com
X-Amz-Target: com.amazonaws.transcribe.Transcribe.StartCallAnalyticsStreamTranscription
Content-Type: application/vnd.amazon.eventstream
X-Amz-Content-Sha256: string
X-Amz-Date: 20220208T235959Z
Authorization: AWS4-HMAC-SHA256 Credential=access-key/20220208/us-west-2/transcribe/
aws4_request, SignedHeaders=content-type;host;x-amz-content-sha256;x-amz-date;x-amz-
target;x-amz-security-token, Signature=string
x-amzn-transcribe-language-code: en-US
x-amzn-transcribe-media-encoding: flac
x-amzn-transcribe-sample-rate: 16000
transfer-encoding: chunked

{
```

```
    "AudioStream": {
        "AudioEvent": {
            "AudioChunk": blob
        },
        "ConfigurationEvent": {
            "ChannelDefinitions": [
                {
                    "ChannelId": 0,
                    "ParticipantRole": "AGENT"
                },
                {
                     "ChannelId": 1,
                    "ParticipantRole": "CUSTOMER"
                }
            ],
            "PostCallAnalyticsSettings": {
                "OutputLocation": "s3://amzn-s3-demo-bucket/my-output-files/",
                "DataAccessRoleArn": "arn:aws:iam::111122223333:role/ExampleRole"
            }
        }
    }
}
```

Parameter definitions can be found in the API Reference; parameters common to all AWS API operations are listed in the Common Parameters section.

WebSocket stream

This example creates a presigned URL that uses Call Analytics in a WebSocket stream. Line breaks have been added for readability. For more information on using WebSocket streams with Amazon Transcribe, see Setting up a WebSocket stream. For more detail on parameters, see `StartCallAnalyticsStreamTranscription`.

This example includes post-call analytics. If you don't want post-call analytics, remove the `PostCallAnalyticsSettings` section from the request.

Note that the configuration event shown in the following example needs to be passed as the first event in the stream.

```
GET wss://transcribestreaming.us-west-2.amazonaws.com:8443/call-analytics-stream-
transcription-websocket?
&X-Amz-Algorithm=AWS4-HMAC-SHA256
```

```
&X-Amz-Credential=AKIAIOSFODNN7EXAMPLE%2F20220208%2Fus-
west-2%2Ftranscribe%2Faws4_request
&X-Amz-Date=20220208T235959Z
&X-Amz-Expires=300
&X-Amz-Security-Token=security-token
&X-Amz-Signature=string
&X-Amz-SignedHeaders=content-type%3Bhost%3Bx-amz-date
&language-code=en-US
&media-encoding=flac
&sample-rate=16000

{
    "AudioStream": {
        "AudioEvent": {
            "AudioChunk": blob
        },
        "ConfigurationEvent": {
            "ChannelDefinitions": [
                {
                    "ChannelId": 0,
                    "ParticipantRole": "AGENT"
                },
                {
                     "ChannelId": 1,
                    "ParticipantRole": "CUSTOMER"
                }
            ],
            "PostCallAnalyticsSettings": {
                "OutputLocation": "s3://amzn-s3-demo-bucket/my-output-files/",
                "DataAccessRoleArn": "arn:aws:iam::111122223333:role/ExampleRole"
            }
        }
    }
}
```

Parameter definitions can be found in the API Reference; parameters common to all AWS API operations are listed in the Common Parameters section.

Tip

The above HTTP/2 and WebSocket examples include post-call analytics. If you don't want post-call analytics, remove the `PostCallAnalyticsSettings` section from the request.

If you enable `PostCallAnalyticsSettings`, you must send a configuration event as the first event. Your configuration event includes settings for `ChannelDenifitions` and `PostStreamAnalyticsSettings`, as shown in the preceding examples.
Binary data are passed as a binary message with `content-type application/octet-stream` and the configuration event is passed as a text message with `content-type application/json`.
For more information, see Setting up a streaming transcription.

Creating real-time alerts for category matches

To set up real-time alerts, you must first create a `TranscriptFilterType` category with the `REAL_TIME` flag. This flag allows your category to be applied to real-time Call Analytics transcriptions.

For instructions on creating a new category, see Creating categories for real-time transcriptions.

When you start your real-time Call Analytics transcription, all categories that have the `REAL_TIME` flag are automatically applied to your transcription output at a segment level. If a `TranscriptFilterType` match occurs, it appears under the `CategoryEvent` section of your transcript. You can then use this parameter and its sub-parameters, `MatchedCategories` and `MatchedDetails`, to set up custom real-time alerts.

Here's an example of real-time Call Analytics transcription output for a `CategoryEvent` match:

```
"CategoryEvent": {
    "MatchedCategories": [ "shipping-complaint" ],
    "MatchedDetails": {
        "my package never arrived" : {
            "TimestampRanges": [
                {
                    "BeginOffsetMillis": 19010,
                    "EndOffsetMillis": 22690
                }
            ]
        }
    }
},
```

The previous example represents an exact text match to the speech "*my package never arrived*," which represents a rule within the 'shipping-complaint' category.

You can set up your real-time alert to include any combination of the listed parameters. For example, you could set your alert to include only the phrase that was matched (`MatchedDetails`) or only the category name (`MatchedCategories`). Or you could set your alert to include all parameters.

How you set up your real-time alerts depends on your organization's interfaces and your desired alert type. For example, you could set a `CategoryEvent` match to send a pop-up notification, an email, a text, or any other alert your system can accept.

Real-time Call Analytics output

Real-time Call Analytics transcripts are displayed in a turn-by-turn format by segment. They include category events, issue detection, sentiment, and PII identification and redaction. Category events allow you to set real-time alerts; see Creating real-time alerts for category matches for more information.

To increase accuracy and further customize your transcripts to your use case, such as including industry-specific terms, add custom vocabularies or custom language models to your Call Analytics request. To mask, remove, or tag words you don't want in your transcription results, such as profanity, add vocabulary filtering.

The following sections show examples of JSON output for real-time Call Analytics transcriptions.

Category events

Here's what a category match looks like in your transcription output. This example shows that the audio from the 19010 millisecond timestamp to the 22690 millisecond timestamp is a match to the 'network-complaint' category. In this case, the custom 'network-complaint' category required that the customer said "*network issues*" (exact word match).

```
"CategoryEvent": {
    "MatchedCategories": [
        "network-complaint"
    ],
    "MatchedDetails": {
        "network issues" : {
            "TimestampRanges": [
                {
```

```
                    "BeginOffsetMillis": 9299375,
                    "EndOffsetMillis": 7899375
                }
            ]
        }
    }
},
```

Issue detection

Here's what an issue detection match looks like in your transcription output. This example shows that the text from character 26 to character 62 describes an issue.

```
"UtteranceEvent": {
    ...
    "Transcript": "Wang Xiulan I'm tired of the network issues my phone is having.",
    ...
    "IssuesDetected": [
        {
            "CharacterOffsets": {
                "BeginOffsetChar": 26,
                "EndOffsetChar": 62
            }
        }
    ]
},
```

Sentiment

Here's what sentiment analysis looks like in your transcription output.

```
"UtteranceEvent": {
    ...
    "Sentiment": "NEGATIVE",
    "Items": [{
        ...
```

PII identification

Here's what PII identification looks like in your transcription output.

```
"Entities": [
```

```
    {
        "Content": "Wang Xiulan",
        "Category": "PII",
        "Type": "NAME",
        "BeginOffsetMillis": 7999375,
        "EndOffsetMillis": 199375,
        "Confidence": 0.9989
    }
],
```

PII redaction

Here's what PII redaction looks like in your transcription output.

```
"Content": "[NAME]. Hi, [NAME]. I'm [NAME] Happy to be helping you today.",
"Redaction": {
    "RedactedTimestamps": [
        {
            "BeginOffsetMillis": 32670,
            "EndOffsetMillis": 33343
        },
        {
            "BeginOffsetMillis": 33518,
            "EndOffsetMillis": 33858
        },
        {
            "BeginOffsetMillis": 34068,
            "EndOffsetMillis": 34488
        }
    ]
},
```

Language identification

Here's what Language identification looks like in your transcription output.

```
{
  "LanguageCode": "en-US",
  "LanguageIdentification": [
    {
      "Code": "en-US",
      "Score": 0.823
    },
```

```
    {
      "Code": "de-DE",
      "Score": 0.177
    }
  ]
}
```

Compiled real-time Call Analytics output

For brevity, some content is replaced with ellipses in the following transcription output.

```
{
    "CallAnalyticsTranscriptResultStream": {
        "BadRequestException": {},
        "ConflictException": {},
        "InternalFailureException": {},
        "LimitExceededException": {},
        "ServiceUnavailableException": {},
        "UtteranceEvent": {
            "UtteranceId": "58c27f92-7277-11ec-90d6-0242ac120003",
            "ParticipantRole": "CUSTOMER",
            "IsPartial": false,
            "Transcript": "Wang Xiulan I'm tired of the network issues my phone is
 having.",
            "BeginOffsetMillis": 19010,
            "EndOffsetMillis": 22690,
            "Sentiment": "NEGATIVE",
            "Items": [{
                    "Content": "Wang",
                    "BeginOffsetMillis": 379937,
                    "EndOffsetMillis": 299375,
                    "Type": "pronunciation",
                    "Confidence": 0.9961,
                    "VocabularyFilterMatch": false
                },
                {
                    "Content": "Xiulan",
                    "EndOffsetMillis": 5899375,
                    "BeginOffsetMillis": 3899375,
                    "Type": "pronunciation",
                    "Confidence": 0.9961,
                    "VocabularyFilterMatch": false
                },
```

```
            ...
            {
                "Content": "network",
                "EndOffsetMillis": 199375,
                "BeginOffsetMillis": 9299375,
                "Type": "pronunciation",
                "Confidence": 0.9961,
                "VocabularyFilterMatch": false
            },
            {
                "Content": "issues",
                "EndOffsetMillis": 7899375,
                "BeginOffsetMillis": 5999375,
                "Type": "pronunciation",
                "Confidence": 0.9961,
                "VocabularyFilterMatch": false
            },
            {
                "Content": "my",
                "EndOffsetMillis": 9199375,
                "BeginOffsetMillis": 7999375,
                "Type": "pronunciation",
                "Confidence": 0.9961,
                "VocabularyFilterMatch": false
            },
            {
                "Content": "phone",
                "EndOffsetMillis": 199375,
                "BeginOffsetMillis": 9299375,
                "Type": "pronunciation",
                "Confidence": 0.9961,
                "VocabularyFilterMatch": false
            },
            ...
        ],
        "Entities": [{
            "Content": "Wang Xiulan",
            "Category": "PII",
            "Type": "NAME",
            "BeginOffsetMillis": 7999375,
            "EndOffsetMillis": 199375,
            "Confidence": 0.9989
        }],
        "IssuesDetected": [{
```

```
                "CharacterOffsets": {
                    "BeginOffsetChar": 26,
                    "EndOffsetChar": 62
                }
            }],
            "LanguageCode": "en-US",
            "LanguageIdentification": [
                {
                    "Code": "en-US",
                    "Score": 0.823
                },
                {
                    "Code": "de-DE",
                    "Score": 0.177
                }
            ]
        },
        "CategoryEvent": {
            "MatchedCategories": [
                "network-complaint"
            ],
            "MatchedDetails": {
                "network issues" : {
                    "TimestampRanges": [
                        {
                            "BeginOffsetMillis": 9299375,
                            "EndOffsetMillis": 7899375
                        }
                    ]
                }
            }
        }
    }
}
```

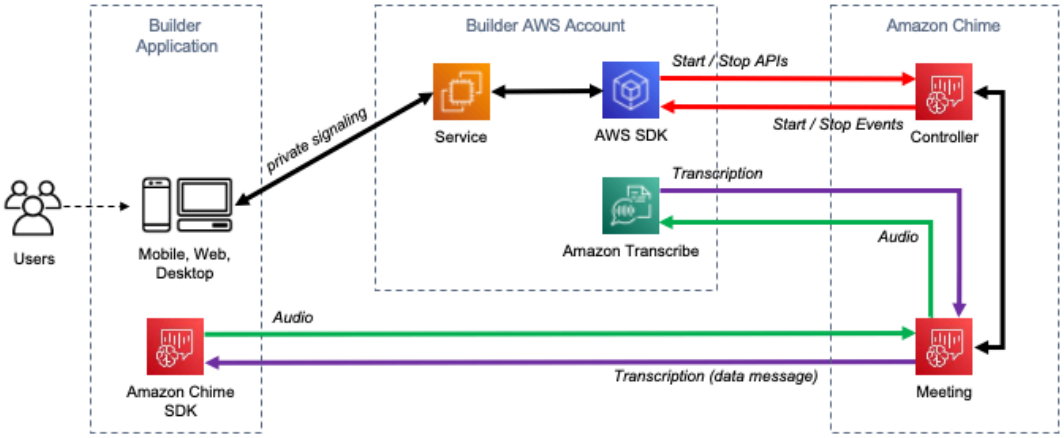
Transcribing your Amazon Chime calls in real time

Amazon Transcribe is integrated with the Amazon Chime SDK, facilitating real-time transcriptions of your Amazon Chime calls.

When you request a transcription using the Amazon Chime SDK API, Amazon Chime begins streaming audio to Amazon Transcribe and continues to do so for the duration of the call.

The Amazon Chime SDK uses its 'active talker' algorithm to select the top two active talkers, and then sends their audio to Amazon Transcribe as two separate channels via a single stream. Meeting participants receive user-attributed transcriptions via Amazon Chime SDK data messages. You can view delivery examples in the *Amazon Chime SDK Developer Guide*.

The data flow of an Amazon Chime transcription is depicted in the following diagram:



For additional information and detailed instructions on how to set up real-time Amazon Chime transcriptions, refer to Using Amazon Chime SDK live transcription in the *Amazon Chime SDK Developer Guide*. For API operations, refer to the *Amazon Chime SDK API Reference*.

Dive deeper with the AWS Machine Learning Blog

To learn more about improving accuracy with real-time transcriptions, see:

- Amazon Chime SDK meetings now support live transcription with Amazon Transcribe and Amazon Transcribe Medical
- Amazon Chime SDK for Telemedicine Solution

Code examples for Amazon Transcribe using AWS SDKs

The following code examples show how to use Amazon Transcribe with an AWS software development kit (SDK).

Actions are code excerpts from larger programs and must be run in context. While actions show you how to call individual service functions, you can see actions in context in their related scenarios.

Scenarios are code examples that show you how to accomplish specific tasks by calling multiple functions within a service or combined with other AWS services.

For a complete list of AWS SDK developer guides and code examples, see Using this service with an AWS SDK. This topic also includes information about getting started and details about previous SDK versions.

Code examples

- Basic examples for Amazon Transcribe using AWS SDKs
  - Actions for Amazon Transcribe using AWS SDKs
    - Use CreateVocabulary with an AWS SDK or CLI
    - Use DeleteMedicalTranscriptionJob with an AWS SDK or CLI
    - Use DeleteTranscriptionJob with an AWS SDK or CLI
    - Use DeleteVocabulary with an AWS SDK or CLI
    - Use GetTranscriptionJob with an AWS SDK or CLI
    - Use GetVocabulary with an AWS SDK or CLI
    - Use ListMedicalTranscriptionJobs with an AWS SDK or CLI
    - Use ListTranscriptionJobs with an AWS SDK or CLI
    - Use ListVocabularies with an AWS SDK or CLI
    - Use StartMedicalTranscriptionJob with an AWS SDK or CLI
    - Use StartTranscriptionJob with an AWS SDK or CLI
    - Use UpdateVocabulary with an AWS SDK or CLI
- Scenarios for Amazon Transcribe using AWS SDKs
  - Build an Amazon Transcribe streaming app
  - Convert text to speech and back to text using an AWS SDK
  - Create and refine an Amazon Transcribe custom vocabulary using an AWS SDK

- Transcribe audio and get job data with Amazon Transcribe using an AWS SDK

Basic examples for Amazon Transcribe using AWS SDKs

The following code examples show how to use the basics of Amazon Transcribe with AWS SDKs.

Examples

- Actions for Amazon Transcribe using AWS SDKs
  - Use CreateVocabulary with an AWS SDK or CLI
  - Use DeleteMedicalTranscriptionJob with an AWS SDK or CLI
  - Use DeleteTranscriptionJob with an AWS SDK or CLI
  - Use DeleteVocabulary with an AWS SDK or CLI
  - Use GetTranscriptionJob with an AWS SDK or CLI
  - Use GetVocabulary with an AWS SDK or CLI
  - Use ListMedicalTranscriptionJobs with an AWS SDK or CLI
  - Use ListTranscriptionJobs with an AWS SDK or CLI
  - Use ListVocabularies with an AWS SDK or CLI
  - Use StartMedicalTranscriptionJob with an AWS SDK or CLI
  - Use StartTranscriptionJob with an AWS SDK or CLI
  - Use UpdateVocabulary with an AWS SDK or CLI

Actions for Amazon Transcribe using AWS SDKs

The following code examples demonstrate how to perform individual Amazon Transcribe actions with AWS SDKs. Each example includes a link to GitHub, where you can find instructions for setting up and running the code.

These excerpts call the Amazon Transcribe API and are code excerpts from larger programs that must be run in context. You can see actions in context in Scenarios for Amazon Transcribe using AWS SDKs.

The following examples include only the most commonly used actions. For a complete list, see the Amazon Transcribe API Reference.

Examples

- Use CreateVocabulary with an AWS SDK or CLI
- Use DeleteMedicalTranscriptionJob with an AWS SDK or CLI
- Use DeleteTranscriptionJob with an AWS SDK or CLI
- Use DeleteVocabulary with an AWS SDK or CLI
- Use GetTranscriptionJob with an AWS SDK or CLI
- Use GetVocabulary with an AWS SDK or CLI
- Use ListMedicalTranscriptionJobs with an AWS SDK or CLI
- Use ListTranscriptionJobs with an AWS SDK or CLI
- Use ListVocabularies with an AWS SDK or CLI
- Use StartMedicalTranscriptionJob with an AWS SDK or CLI
- Use StartTranscriptionJob with an AWS SDK or CLI
- Use UpdateVocabulary with an AWS SDK or CLI

Use `CreateVocabulary` with an AWS SDK or CLI

The following code examples show how to use `CreateVocabulary`.

Action examples are code excerpts from larger programs and must be run in context. You can see this action in context in the following code example:

- Create and refine a custom vocabulary

.NET

SDK for .NET

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
/// <summary>
/// Create a custom vocabulary using a list of phrases. Custom vocabularies
/// improve transcription accuracy for one or more specific words.
```

```
    /// </summary>
    /// <param name="languageCode">The language code of the vocabulary.</param>
    /// <param name="phrases">Phrases to use in the vocabulary.</param>
    /// <param name="vocabularyName">Name for the vocabulary.</param>
    /// <returns>The state of the custom vocabulary.</returns>
    public async Task<VocabularyState> CreateCustomVocabulary(LanguageCode
 languageCode,
        List<string> phrases, string vocabularyName)
    {
        var response = await _amazonTranscribeService.CreateVocabularyAsync(
            new CreateVocabularyRequest
            {
                LanguageCode = languageCode,
                Phrases = phrases,
                VocabularyName = vocabularyName
            });
        return response.VocabularyState;
    }
```

- For API details, see CreateVocabulary in *AWS SDK for .NET API Reference*.

CLI

AWS CLI

To create a custom vocabulary

The following `create-vocabulary` example creates a custom vocabulary. To create a custom vocabulary, you must have created a text file with all the terms that you want to transcribe more accurately. For vocabulary-file-uri, specify the Amazon Simple Storage Service (Amazon S3) URI of that text file. For language-code, specify a language code corresponding to the language of your custom vocabulary. For vocabulary-name, specify what you want to call your custom vocabulary.

```
aws transcribe create-vocabulary \
    --language-code language-code \
    --vocabulary-name cli-vocab-example \
    --vocabulary-file-uri s3://amzn-s3-demo-bucket/Amazon-S3-prefix/the-text-
file-for-the-custom-vocabulary.txt
```

Output:

```
{
    "VocabularyName": "cli-vocab-example",
    "LanguageCode": "language-code",
    "VocabularyState": "PENDING"
}
```

For more information, see Custom Vocabularies in the *Amazon Transcribe Developer Guide*.

- For API details, see CreateVocabulary in *AWS CLI Command Reference*.

Python

SDK for Python (Boto3)

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
def create_vocabulary(
    vocabulary_name, language_code, transcribe_client, phrases=None,
 table_uri=None
):
    """
    Creates a custom vocabulary that can be used to improve the accuracy of
    transcription jobs. This function returns as soon as the vocabulary
 processing
    is started. Call get_vocabulary to get the current status of the vocabulary.
    The vocabulary is ready to use when its status is 'READY'.

    :param vocabulary_name: The name of the custom vocabulary.
    :param language_code: The language code of the vocabulary.
                          For example, en-US or nl-NL.
    :param transcribe_client: The Boto3 Transcribe client.
    :param phrases: A list of comma-separated phrases to include in the
 vocabulary.
    :param table_uri: A table of phrases and pronunciation hints to include in
 the
```

```
                    vocabulary.
    :return: Information about the newly created vocabulary.
    """
    try:
        vocab_args = {"VocabularyName": vocabulary_name, "LanguageCode":
language_code}
        if phrases is not None:
            vocab_args["Phrases"] = phrases
        elif table_uri is not None:
            vocab_args["VocabularyFileUri"] = table_uri
        response = transcribe_client.create_vocabulary(**vocab_args)
        logger.info("Created custom vocabulary %s.", response["VocabularyName"])
    except ClientError:
        logger.exception("Couldn't create custom vocabulary %s.",
vocabulary_name)
        raise
    else:
        return response
```

- For API details, see CreateVocabulary in *AWS SDK for Python (Boto3) API Reference*.

SAP ABAP

SDK for SAP ABAP

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
    TRY.
        IF it_phrases IS NOT INITIAL.
          oo_result = lo_tnb->createvocabulary(
            iv_vocabularyname = iv_vocabulary_name
            iv_languagecode = iv_language_code
            it_phrases = it_phrases ).
        ELSEIF iv_vocab_file_uri IS NOT INITIAL.
          oo_result = lo_tnb->createvocabulary(
```

```
            iv_vocabularyname = iv_vocabulary_name
            iv_languagecode = iv_language_code
            iv_vocabularyfileuri = iv_vocab_file_uri ).
        ENDIF.
        MESSAGE 'Custom vocabulary created.' TYPE 'I'.
      CATCH /aws1/cx_tnbbadrequestex INTO DATA(lo_bad_request_ex).
        MESSAGE lo_bad_request_ex TYPE 'I'.
        RAISE EXCEPTION lo_bad_request_ex.
      CATCH /aws1/cx_tnblimitexceededex INTO DATA(lo_limit_ex).
        MESSAGE lo_limit_ex TYPE 'I'.
        RAISE EXCEPTION lo_limit_ex.
      CATCH /aws1/cx_tnbinternalfailureex INTO DATA(lo_internal_ex).
        MESSAGE lo_internal_ex TYPE 'I'.
        RAISE EXCEPTION lo_internal_ex.
      CATCH /aws1/cx_tnbconflictexception INTO DATA(lo_conflict_ex).
        MESSAGE lo_conflict_ex TYPE 'I'.
        RAISE EXCEPTION lo_conflict_ex.
    ENDTRY.
```

- For API details, see CreateVocabulary in *AWS SDK for SAP ABAP API reference*.

For a complete list of AWS SDK developer guides and code examples, see Using this service with an AWS SDK. This topic also includes information about getting started and details about previous SDK versions.

Use `DeleteMedicalTranscriptionJob` with an AWS SDK or CLI

The following code examples show how to use `DeleteMedicalTranscriptionJob`.

.NET

SDK for .NET

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
    /// <summary>
    /// Delete a medical transcription job. Also deletes the transcript
 associated with the job.
    /// </summary>
    /// <param name="jobName">Name of the medical transcription job to delete.</
param>
    /// <returns>True if successful.</returns>
    public async Task<bool> DeleteMedicalTranscriptionJob(string jobName)
    {
        var response = await
 _amazonTranscribeService.DeleteMedicalTranscriptionJobAsync(
            new DeleteMedicalTranscriptionJobRequest()
            {
                MedicalTranscriptionJobName = jobName
            });
        return response.HttpStatusCode == HttpStatusCode.OK;
    }
```

- For API details, see DeleteMedicalTranscriptionJob in *AWS SDK for .NET API Reference*.

CLI

AWS CLI

To delete a medical transcription job

The following `delete-medical-transcription-job` example deletes a medical transcription job.

```
aws transcribe delete-medical-transcription-job \
    --medical-transcription-job-name medical-transcription-job-name
```

This command produces no output.

For more information, see DeleteMedicalTranscriptionJob in the *Amazon Transcribe Developer Guide*.

- For API details, see DeleteMedicalTranscriptionJob in *AWS CLI Command Reference*.

JavaScript

SDK for JavaScript (v3)

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

Create the client.

```
import { TranscribeClient } from "@aws-sdk/client-transcribe";
// Set the AWS Region.
const REGION = "REGION"; //e.g. "us-east-1"
// Create an Amazon Transcribe service client object.
const transcribeClient = new TranscribeClient({ region: REGION });
export { transcribeClient };
```

Delete a medical transcription job.

```
// Import the required AWS SDK clients and commands for Node.js
import { DeleteMedicalTranscriptionJobCommand } from "@aws-sdk/client-
transcribe";
import { transcribeClient } from "./libs/transcribeClient.js";

// Set the parameters
export const params = {
  MedicalTranscriptionJobName: "MEDICAL_JOB_NAME", // For example,
 'medical_transciption_demo'
};

export const run = async () => {
  try {
    const data = await transcribeClient.send(
      new DeleteMedicalTranscriptionJobCommand(params),
    );
    console.log("Success - deleted");
    return data; // For unit tests.
  } catch (err) {
    console.log("Error", err);
```

```
  }
};
run();
```

- For more information, see AWS SDK for JavaScript Developer Guide.
- For API details, see DeleteMedicalTranscriptionJob in *AWS SDK for JavaScript API Reference*.

For a complete list of AWS SDK developer guides and code examples, see Using this service with an AWS SDK. This topic also includes information about getting started and details about previous SDK versions.

Use `DeleteTranscriptionJob` with an AWS SDK or CLI

The following code examples show how to use `DeleteTranscriptionJob`.

Action examples are code excerpts from larger programs and must be run in context. You can see this action in context in the following code example:

- Create and refine a custom vocabulary

.NET

SDK for .NET

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
    /// <summary>
    /// Delete a transcription job. Also deletes the transcript associated with
the job.
    /// </summary>
    /// <param name="jobName">Name of the transcription job to delete.</param>
    /// <returns>True if successful.</returns>
```

```
    public async Task<bool> DeleteTranscriptionJob(string jobName)
    {
        var response = await
_amazonTranscribeService.DeleteTranscriptionJobAsync(
            new DeleteTranscriptionJobRequest()
            {
                TranscriptionJobName = jobName
            });
        return response.HttpStatusCode == HttpStatusCode.OK;
    }
```

- For API details, see DeleteTranscriptionJob in *AWS SDK for .NET API Reference*.

CLI

AWS CLI

To delete one of your transcription jobs

The following `delete-transcription-job` example deletes one of your transcription jobs.

```
aws transcribe delete-transcription-job \
    --transcription-job-name your-transcription-job
```

This command produces no output.

For more information, see DeleteTranscriptionJob in the *Amazon Transcribe Developer Guide*.

- For API details, see DeleteTranscriptionJob in *AWS CLI Command Reference*.

JavaScript

SDK for JavaScript (v3)

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

Delete a transcription job.

```
// Import the required AWS SDK clients and commands for Node.js
import { DeleteTranscriptionJobCommand } from "@aws-sdk/client-transcribe";
import { transcribeClient } from "./libs/transcribeClient.js";

// Set the parameters
export const params = {
  TranscriptionJobName: "JOB_NAME", // Required. For example, 'transciption_demo'
};

export const run = async () => {
  try {
    const data = await transcribeClient.send(
      new DeleteTranscriptionJobCommand(params),
    );
    console.log("Success - deleted");
    return data; // For unit tests.
  } catch (err) {
    console.log("Error", err);
  }
};
run();
```

Create the client.

```
import { TranscribeClient } from "@aws-sdk/client-transcribe";
// Set the AWS Region.
const REGION = "REGION"; //e.g. "us-east-1"
// Create an Amazon Transcribe service client object.
const transcribeClient = new TranscribeClient({ region: REGION });
export { transcribeClient };
```

- For more information, see AWS SDK for JavaScript Developer Guide.
- For API details, see DeleteTranscriptionJob in *AWS SDK for JavaScript API Reference*.

Python

SDK for Python (Boto3)

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
def delete_job(job_name, transcribe_client):
    """
    Deletes a transcription job. This also deletes the transcript associated with
    the job.

    :param job_name: The name of the job to delete.
    :param transcribe_client: The Boto3 Transcribe client.
    """
    try:
        transcribe_client.delete_transcription_job(TranscriptionJobName=job_name)
        logger.info("Deleted job %s.", job_name)
    except ClientError:
        logger.exception("Couldn't delete job %s.", job_name)
        raise
```

- For API details, see DeleteTranscriptionJob in *AWS SDK for Python (Boto3) API Reference*.

SAP ABAP

SDK for SAP ABAP

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
    TRY.
        lo_tnb->deletetranscriptionjob( iv_job_name ).
        MESSAGE 'Transcription job deleted.' TYPE 'I'.
      CATCH /aws1/cx_tnbbadrequestex INTO DATA(lo_bad_request_ex).
        MESSAGE lo_bad_request_ex TYPE 'I'.
        RAISE EXCEPTION lo_bad_request_ex.
      CATCH /aws1/cx_tnblimitexceededex INTO DATA(lo_limit_ex).
        MESSAGE lo_limit_ex TYPE 'I'.
        RAISE EXCEPTION lo_limit_ex.
      CATCH /aws1/cx_tnbinternalfailureex INTO DATA(lo_internal_ex).
        MESSAGE lo_internal_ex TYPE 'I'.
        RAISE EXCEPTION lo_internal_ex.
    ENDTRY.
```

- For API details, see DeleteTranscriptionJob in *AWS SDK for SAP ABAP API reference*.

For a complete list of AWS SDK developer guides and code examples, see Using this service with an AWS SDK. This topic also includes information about getting started and details about previous SDK versions.

Use `DeleteVocabulary` with an AWS SDK or CLI

The following code examples show how to use `DeleteVocabulary`.

Action examples are code excerpts from larger programs and must be run in context. You can see this action in context in the following code example:

- Create and refine a custom vocabulary

.NET

SDK for .NET

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
    /// <summary>
    /// Delete an existing custom vocabulary.
    /// </summary>
    /// <param name="vocabularyName">Name of the vocabulary to delete.</param>
    /// <returns>True if successful.</returns>
    public async Task<bool> DeleteCustomVocabulary(string vocabularyName)
    {
        var response = await _amazonTranscribeService.DeleteVocabularyAsync(
            new DeleteVocabularyRequest
            {
                VocabularyName = vocabularyName
            });
        return response.HttpStatusCode == HttpStatusCode.OK;
    }
```

- For API details, see DeleteVocabulary in *AWS SDK for .NET API Reference*.

CLI

AWS CLI

To delete a custom vocabulary

The following `delete-vocabulary` example deletes a custom vocabulary.

```
aws transcribe delete-vocabulary \
    --vocabulary-name vocabulary-name
```

This command produces no output.

For more information, see Custom Vocabularies in the *Amazon Transcribe Developer Guide*.

- For API details, see DeleteVocabulary in *AWS CLI Command Reference*.

Python

SDK for Python (Boto3)

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
def delete_vocabulary(vocabulary_name, transcribe_client):
    """
    Deletes a custom vocabulary.

    :param vocabulary_name: The name of the vocabulary to delete.
    :param transcribe_client: The Boto3 Transcribe client.
    """
    try:
        transcribe_client.delete_vocabulary(VocabularyName=vocabulary_name)
        logger.info("Deleted vocabulary %s.", vocabulary_name)
    except ClientError:
        logger.exception("Couldn't delete vocabulary %s.", vocabulary_name)
        raise
```

- For API details, see DeleteVocabulary in *AWS SDK for Python (Boto3) API Reference*.

SAP ABAP

SDK for SAP ABAP

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
    TRY.
```

```
        lo_tnb->deletevocabulary( iv_vocabulary_name ).
        MESSAGE 'Vocabulary deleted.' TYPE 'I'.
      CATCH /aws1/cx_tnbbadrequestex INTO DATA(lo_bad_request_ex).
        MESSAGE lo_bad_request_ex TYPE 'I'.
      CATCH /aws1/cx_tnblimitexceededex INTO DATA(lo_limit_ex).
        MESSAGE lo_limit_ex TYPE 'I'.
        RAISE EXCEPTION lo_limit_ex.
      CATCH /aws1/cx_tnbnotfoundexception INTO DATA(lo_not_found_ex).
        MESSAGE lo_not_found_ex TYPE 'I'.
      CATCH /aws1/cx_tnbinternalfailureex INTO DATA(lo_internal_ex).
        MESSAGE lo_internal_ex TYPE 'I'.
        RAISE EXCEPTION lo_internal_ex.
    ENDTRY.
```

- For API details, see DeleteVocabulary in *AWS SDK for SAP ABAP API reference*.

For a complete list of AWS SDK developer guides and code examples, see Using this service with an AWS SDK. This topic also includes information about getting started and details about previous SDK versions.

Use `GetTranscriptionJob` with an AWS SDK or CLI

The following code examples show how to use `GetTranscriptionJob`.

Action examples are code excerpts from larger programs and must be run in context. You can see this action in context in the following code examples:

- Create and refine a custom vocabulary
- Transcribe audio and get job data

.NET

SDK for .NET

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
    /// <summary>
    /// Get details about a transcription job.
    /// </summary>
    /// <param name="jobName">A unique name for the transcription job.</param>
    /// <returns>A TranscriptionJob instance with information on the requested
 job.</returns>
    public async Task<TranscriptionJob> GetTranscriptionJob(string jobName)
    {
        var response = await _amazonTranscribeService.GetTranscriptionJobAsync(
            new GetTranscriptionJobRequest()
            {
                TranscriptionJobName = jobName
            });
        return response.TranscriptionJob;
    }
```

- For API details, see GetTranscriptionJob in *AWS SDK for .NET API Reference*.

CLI

AWS CLI

To get information about a specific transcription job

The following `get-transcription-job` example gets information about a specific transcription job. To access the transcription results, use the TranscriptFileUri parameter. Use the MediaFileUri parameter to see which audio file you transcribed with this job. You can use the Settings object to see the optional features you've enabled in the transcription job.

```
aws transcribe get-transcription-job \
    --transcription-job-name your-transcription-job
```

Output:

```
{
    "TranscriptionJob": {
        "TranscriptionJobName": "your-transcription-job",
        "TranscriptionJobStatus": "COMPLETED",
```

```
        "LanguageCode": "language-code",
        "MediaSampleRateHertz": 48000,
        "MediaFormat": "mp4",
        "Media": {
            "MediaFileUri": "s3://amzn-s3-demo-bucket/your-audio-file.file-
extension"
        },
        "Transcript": {
            "TranscriptFileUri": "https://Amazon-S3-file-location-of-
transcription-output"
        },
        "StartTime": "2020-09-18T22:27:23.970000+00:00",
        "CreationTime": "2020-09-18T22:27:23.948000+00:00",
        "CompletionTime": "2020-09-18T22:28:21.197000+00:00",
        "Settings": {
            "ChannelIdentification": false,
            "ShowAlternatives": false
        },
        "IdentifyLanguage": true,
        "IdentifiedLanguageScore": 0.8672199249267578
    }
}
```

For more information, see Getting Started (AWS Command Line Interface) in the *Amazon Transcribe Developer Guide*.

- For API details, see GetTranscriptionJob in *AWS CLI Command Reference*.

Python

SDK for Python (Boto3)

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
def get_job(job_name, transcribe_client):
    """
    Gets details about a transcription job.
```

```
    :param job_name: The name of the job to retrieve.
    :param transcribe_client: The Boto3 Transcribe client.
    :return: The retrieved transcription job.
    """
    try:
        response = transcribe_client.get_transcription_job(
            TranscriptionJobName=job_name
        )
        job = response["TranscriptionJob"]
        logger.info("Got job %s.", job["TranscriptionJobName"])
    except ClientError:
        logger.exception("Couldn't get job %s.", job_name)
        raise
    else:
        return job
```

- For API details, see GetTranscriptionJob in *AWS SDK for Python (Boto3) API Reference*.

SAP ABAP

SDK for SAP ABAP

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
    TRY.
        oo_result = lo_tnb->gettranscriptionjob( iv_job_name ).
        DATA(lo_job) = oo_result->get_transcriptionjob( ).
        MESSAGE 'Retrieved transcription job details.' TYPE 'I'.
      CATCH /aws1/cx_tnbbadrequestex INTO DATA(lo_bad_request_ex).
        MESSAGE lo_bad_request_ex TYPE 'I'.
        RAISE EXCEPTION lo_bad_request_ex.
      CATCH /aws1/cx_tnbnotfoundexception INTO DATA(lo_not_found_ex).
        MESSAGE lo_not_found_ex TYPE 'I'.
        RAISE EXCEPTION lo_not_found_ex.
      CATCH /aws1/cx_tnbinternalfailureex INTO DATA(lo_internal_ex).
```

```
        MESSAGE lo_internal_ex TYPE 'I'.
        RAISE EXCEPTION lo_internal_ex.
    ENDTRY.
```

- For API details, see GetTranscriptionJob in *AWS SDK for SAP ABAP API reference*.

For a complete list of AWS SDK developer guides and code examples, see Using this service with an AWS SDK. This topic also includes information about getting started and details about previous SDK versions.

Use `GetVocabulary` with an AWS SDK or CLI

The following code examples show how to use `GetVocabulary`.

Action examples are code excerpts from larger programs and must be run in context. You can see this action in context in the following code example:

- Create and refine a custom vocabulary

.NET

SDK for .NET

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
/// <summary>
/// Get information about a custom vocabulary.
/// </summary>
/// <param name="vocabularyName">Name of the vocabulary.</param>
/// <returns>The state of the custom vocabulary.</returns>
public async Task<VocabularyState> GetCustomVocabulary(string vocabularyName)
{
    var response = await _amazonTranscribeService.GetVocabularyAsync(
        new GetVocabularyRequest()
```

```
            {
                VocabularyName = vocabularyName
            });
        return response.VocabularyState;
    }
```

- For API details, see GetVocabulary in *AWS SDK for .NET API Reference*.

CLI

AWS CLI

To get information about a custom vocabulary

The following `get-vocabulary` example gets information on a previously created custom vocabulary.

```
aws transcribe get-vocabulary \
    --vocabulary-name cli-vocab-1
```

Output:

```
{
    "VocabularyName": "cli-vocab-1",
    "LanguageCode": "language-code",
    "VocabularyState": "READY",
    "LastModifiedTime": "2020-09-19T23:22:32.836000+00:00",
    "DownloadUri": "https://link-to-download-the-text-file-used-to-create-your-
custom-vocabulary"
}
```

For more information, see Custom Vocabularies in the *Amazon Transcribe Developer Guide*.

- For API details, see GetVocabulary in *AWS CLI Command Reference*.

Python

SDK for Python (Boto3)

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
def get_vocabulary(vocabulary_name, transcribe_client):
    """
    Gets information about a custom vocabulary.

    :param vocabulary_name: The name of the vocabulary to retrieve.
    :param transcribe_client: The Boto3 Transcribe client.
    :return: Information about the vocabulary.
    """
    try:
        response =
 transcribe_client.get_vocabulary(VocabularyName=vocabulary_name)
        logger.info("Got vocabulary %s.", response["VocabularyName"])
    except ClientError:
        logger.exception("Couldn't get vocabulary %s.", vocabulary_name)
        raise
    else:
        return response
```

- For API details, see GetVocabulary in *AWS SDK for Python (Boto3) API Reference*.

SAP ABAP

SDK for SAP ABAP

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
TRY.
    oo_result = lo_tnb->getvocabulary( iv_vocabulary_name ).
    MESSAGE 'Retrieved vocabulary details.' TYPE 'I'.
  CATCH /aws1/cx_tnbbadrequestex INTO DATA(lo_bad_request_ex).
    MESSAGE lo_bad_request_ex TYPE 'I'.
    RAISE EXCEPTION lo_bad_request_ex.
  CATCH /aws1/cx_tnbnotfoundexception INTO DATA(lo_not_found_ex).
    MESSAGE lo_not_found_ex TYPE 'I'.
    RAISE EXCEPTION lo_not_found_ex.
  CATCH /aws1/cx_tnbinternalfailureex INTO DATA(lo_internal_ex).
    MESSAGE lo_internal_ex TYPE 'I'.
    RAISE EXCEPTION lo_internal_ex.
ENDTRY.
```

- For API details, see GetVocabulary in *AWS SDK for SAP ABAP API reference*.

For a complete list of AWS SDK developer guides and code examples, see Using this service with an AWS SDK. This topic also includes information about getting started and details about previous SDK versions.

Use `ListMedicalTranscriptionJobs` with an AWS SDK or CLI

The following code examples show how to use `ListMedicalTranscriptionJobs`.

.NET

SDK for .NET

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
    /// <summary>
    /// List medical transcription jobs, optionally with a name filter.
    /// </summary>
    /// <param name="jobNameContains">Optional name filter for the medical
transcription jobs.</param>
    /// <returns>A list of summaries about medical transcription jobs.</returns>
    public async Task<List<MedicalTranscriptionJobSummary>>
ListMedicalTranscriptionJobs(
        string? jobNameContains = null)
    {
        var response = await
_amazonTranscribeService.ListMedicalTranscriptionJobsAsync(
            new ListMedicalTranscriptionJobsRequest()
            {
                JobNameContains = jobNameContains
            });
        return response.MedicalTranscriptionJobSummaries;
    }
```

- For API details, see ListMedicalTranscriptionJobs in *AWS SDK for .NET API Reference*.

CLI

AWS CLI

To list your medical transcription jobs

The following `list-medical-transcription-jobs` example lists the medical transcription jobs associated with your AWS account and Region. To get more information about a particular transcription job, copy the value of a MedicalTranscriptionJobName parameter in the transcription output, and specify that value for the `MedicalTranscriptionJobName` option of the `get-medical-transcription-job` command. To see more of your transcription jobs, copy the value of the NextToken parameter, run the `list-medical-transcription-jobs` command again, and specify that value in the `--next-token` option.

```
aws transcribe list-medical-transcription-jobs
```

Output:


```
{
    "NextToken": "3/PblzkiGhzjER3KHuQt2fmbPLF7cDYafjFMEoGn44ON/
gsuUSTIkGyanvRE6WMXFd/ZTEc2EZj+P9eii/
z1O2FDYli6RLI0WoRX4RwMisVrh9G0Kie0Y8ikBCdtqlZB10Wa9McC+ebOl
+LaDtZPC4u6ttoHLRlEfzqstHXSgapXg3tEBtm9piIaPB6MOM5BB6t86+qtmocTR/
qrteHZBBudhTfbCwhsxaqujHiiUvFdm3BQbKKWIW06yV9b+4f38oD2lVIan
+vfUs3gBYAl5VTDmXXzQPBQOHPjtwmFI+IWX15nSUjWuN3TUylHgPWzDaYT8qBtu0Z+3UG4V6b
+K2CC0XszXg5rBq9hYgNzy4XoFh/6s5DoSnzq49Q9xHgHdT2yBADFmvFK7myZBsj75+2vQZOSVpWUPy3WT/32zFAc
+mFYfUjtTZ8n/jq7aQEjQ42A
+X/7K6JgOcdVPtEg8PlDr5kgYYG3q3OmYXX37U3FZuJmnTI63VtIXsNnOU5eGoYObtpk00Nq9UkzgSJxqj84ZD5n
+S0EGy9ZUYBJRRcGeYUM3Q4DbSJfUwSAqcFdLIWZdp8qIREMQIBWy7BLwSdyqsQo2vRrd53hm5aWM7SVf6pPq6X/
IXR5+1eUOOD8/coaTT4ES2DerbV6RkV4o0VT1d0SdVX/
MmtkNG8nYj8PqU07w7988quh1ZP6D80veJS1q73tUUR9MjnGernW2tAnvnLNhdefBcD
+sZVfYq3iBMFY7wTy1P1G6NqW9GrYDYoX3tTPWlD7phpbVSyKrh/
PdYrps5UxnsGoA1b7L/FfAXDfUoGrGUB4N3JsPYXX9D++g+6gV1qBBs/
WfF934aKqfD6UTggm/zV3GAOWiBpfvAZRvEb924i6yGHyMC7y54O1ZAwSBupmI
+FFd13CaPO4kN1vJlth6aM5vUPXg4BpyUhtbRhwD/KxCvf9K0tLJGyL1A==",
    "MedicalTranscriptionJobSummaries": [
        {
            "MedicalTranscriptionJobName": "vocabulary-dictation-medical-
transcription-job",
            "CreationTime": "2020-09-21T21:17:27.016000+00:00",
            "StartTime": "2020-09-21T21:17:27.045000+00:00",
            "CompletionTime": "2020-09-21T21:17:59.561000+00:00",
            "LanguageCode": "en-US",
            "TranscriptionJobStatus": "COMPLETED",
            "OutputLocationType": "CUSTOMER_BUCKET",
            "Specialty": "PRIMARYCARE",
            "Type": "DICTATION"
```

```
        },
        {
            "MedicalTranscriptionJobName": "alternatives-dictation-medical-
transcription-job",
            "CreationTime": "2020-09-21T21:01:14.569000+00:00",
            "StartTime": "2020-09-21T21:01:14.592000+00:00",
            "CompletionTime": "2020-09-21T21:01:43.606000+00:00",
            "LanguageCode": "en-US",
            "TranscriptionJobStatus": "COMPLETED",
            "OutputLocationType": "CUSTOMER_BUCKET",
            "Specialty": "PRIMARYCARE",
            "Type": "DICTATION"
        },
        {
            "MedicalTranscriptionJobName": "alternatives-conversation-medical-
transcription-job",
            "CreationTime": "2020-09-21T19:09:18.171000+00:00",
            "StartTime": "2020-09-21T19:09:18.199000+00:00",
            "CompletionTime": "2020-09-21T19:10:22.516000+00:00",
            "LanguageCode": "en-US",
            "TranscriptionJobStatus": "COMPLETED",
            "OutputLocationType": "CUSTOMER_BUCKET",
            "Specialty": "PRIMARYCARE",
            "Type": "CONVERSATION"
        },
        {
            "MedicalTranscriptionJobName": "speaker-id-conversation-medical-
transcription-job",
            "CreationTime": "2020-09-21T18:43:37.157000+00:00",
            "StartTime": "2020-09-21T18:43:37.265000+00:00",
            "CompletionTime": "2020-09-21T18:44:21.192000+00:00",
            "LanguageCode": "en-US",
            "TranscriptionJobStatus": "COMPLETED",
            "OutputLocationType": "CUSTOMER_BUCKET",
            "Specialty": "PRIMARYCARE",
            "Type": "CONVERSATION"
        },
        {
            "MedicalTranscriptionJobName": "multichannel-conversation-medical-
transcription-job",
            "CreationTime": "2020-09-20T23:46:44.053000+00:00",
            "StartTime": "2020-09-20T23:46:44.081000+00:00",
            "CompletionTime": "2020-09-20T23:47:35.851000+00:00",
            "LanguageCode": "en-US",
```

```
            "TranscriptionJobStatus": "COMPLETED",
            "OutputLocationType": "CUSTOMER_BUCKET",
            "Specialty": "PRIMARYCARE",
            "Type": "CONVERSATION"
        }
    ]
}
```

For more information, see Amazon Transcribe Medical in the *Amazon Transcribe Developer Guide*.

- For API details, see ListMedicalTranscriptionJobs in *AWS CLI Command Reference*.

JavaScript

SDK for JavaScript (v3)

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

Create the client.

```
import { TranscribeClient } from "@aws-sdk/client-transcribe";
// Set the AWS Region.
const REGION = "REGION"; //e.g. "us-east-1"
// Create an Amazon Transcribe service client object.
const transcribeClient = new TranscribeClient({ region: REGION });
export { transcribeClient };
```

List medical transcription jobs.

```
// Import the required AWS SDK clients and commands for Node.js
import { StartMedicalTranscriptionJobCommand } from "@aws-sdk/client-transcribe";
import { transcribeClient } from "./libs/transcribeClient.js";

// Set the parameters
export const params = {
```

```
  MedicalTranscriptionJobName: "MEDICAL_JOB_NAME", // Required
  OutputBucketName: "OUTPUT_BUCKET_NAME", // Required
  Specialty: "PRIMARYCARE", // Required. Possible values are 'PRIMARYCARE'
  Type: "JOB_TYPE", // Required. Possible values are 'CONVERSATION' and
 'DICTATION'
  LanguageCode: "LANGUAGE_CODE", // For example, 'en-US'
  MediaFormat: "SOURCE_FILE_FORMAT", // For example, 'wav'
  Media: {
    MediaFileUri: "SOURCE_FILE_LOCATION",
    // The S3 object location of the input media file. The URI must be in the
 same region
    // as the API endpoint that you are calling.For example,
    // "https://transcribe-demo.s3-REGION.amazonaws.com/hello_world.wav"
  },
};

export const run = async () => {
  try {
    const data = await transcribeClient.send(
      new StartMedicalTranscriptionJobCommand(params),
    );
    console.log("Success - put", data);
    return data; // For unit tests.
  } catch (err) {
    console.log("Error", err);
  }
};
run();
```

- For more information, see AWS SDK for JavaScript Developer Guide.
- For API details, see ListMedicalTranscriptionJobs in *AWS SDK for JavaScript API Reference*.

For a complete list of AWS SDK developer guides and code examples, see Using this service with an AWS SDK. This topic also includes information about getting started and details about previous SDK versions.

Use `ListTranscriptionJobs` with an AWS SDK or CLI

The following code examples show how to use `ListTranscriptionJobs`.

.NET

SDK for .NET

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
    /// <summary>
    /// List transcription jobs, optionally with a name filter.
    /// </summary>
    /// <param name="jobNameContains">Optional name filter for the transcription
jobs.</param>
    /// <returns>A list of transcription job summaries.</returns>
    public async Task<List<TranscriptionJobSummary>>
ListTranscriptionJobs(string? jobNameContains = null)
    {
        var response = await _amazonTranscribeService.ListTranscriptionJobsAsync(
            new ListTranscriptionJobsRequest()
            {
                JobNameContains = jobNameContains
            });
        return response.TranscriptionJobSummaries;
    }
```

- For API details, see ListTranscriptionJobs in *AWS SDK for .NET API Reference*.

CLI

AWS CLI

To list your transcription jobs

The following `list-transcription-jobs` example lists the transcription jobs associated with your AWS account and Region.

```
aws transcribe list-transcription-jobs
```

Output:

```
{
    "NextToken": "NextToken",
    "TranscriptionJobSummaries": [
        {
            "TranscriptionJobName": "speak-id-job-1",
            "CreationTime": "2020-08-17T21:06:15.391000+00:00",
            "StartTime": "2020-08-17T21:06:15.416000+00:00",
            "CompletionTime": "2020-08-17T21:07:05.098000+00:00",
            "LanguageCode": "language-code",
            "TranscriptionJobStatus": "COMPLETED",
            "OutputLocationType": "SERVICE_BUCKET"
        },
        {
            "TranscriptionJobName": "job-1",
            "CreationTime": "2020-08-17T20:50:24.207000+00:00",
            "StartTime": "2020-08-17T20:50:24.230000+00:00",
            "CompletionTime": "2020-08-17T20:52:18.737000+00:00",
            "LanguageCode": "language-code",
            "TranscriptionJobStatus": "COMPLETED",
            "OutputLocationType": "SERVICE_BUCKET"
        },
        {
            "TranscriptionJobName": "sdk-test-job-4",
            "CreationTime": "2020-08-17T20:32:27.917000+00:00",
            "StartTime": "2020-08-17T20:32:27.956000+00:00",
            "CompletionTime": "2020-08-17T20:33:15.126000+00:00",
            "LanguageCode": "language-code",
            "TranscriptionJobStatus": "COMPLETED",
            "OutputLocationType": "SERVICE_BUCKET"
        },
        {
            "TranscriptionJobName": "Diarization-speak-id",
            "CreationTime": "2020-08-10T22:10:09.066000+00:00",
            "StartTime": "2020-08-10T22:10:09.116000+00:00",
            "CompletionTime": "2020-08-10T22:26:48.172000+00:00",
            "LanguageCode": "language-code",
            "TranscriptionJobStatus": "COMPLETED",
            "OutputLocationType": "SERVICE_BUCKET"
        },
```

```
        {
            "TranscriptionJobName": "your-transcription-job-name",
            "CreationTime": "2020-07-29T17:45:09.791000+00:00",
            "StartTime": "2020-07-29T17:45:09.826000+00:00",
            "CompletionTime": "2020-07-29T17:46:20.831000+00:00",
            "LanguageCode": "language-code",
            "TranscriptionJobStatus": "COMPLETED",
            "OutputLocationType": "SERVICE_BUCKET"
        }
    ]
}
```

For more information, see Getting Started (AWS Command Line Interface) in the *Amazon Transcribe Developer Guide*.

- For API details, see ListTranscriptionJobs in *AWS CLI Command Reference*.

Java

SDK for Java 2.x

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
public class ListTranscriptionJobs {
    public static void main(String[] args) {
        TranscribeClient transcribeClient = TranscribeClient.builder()
            .region(Region.US_EAST_1)
            .build();

            listTranscriptionJobs(transcribeClient);
        }

        public static void listTranscriptionJobs(TranscribeClient
 transcribeClient) {
            ListTranscriptionJobsRequest listJobsRequest =
 ListTranscriptionJobsRequest.builder()
                .build();
```

```
transcribeClient.listTranscriptionJobsPaginator(listJobsRequest).stream()
                .flatMap(response ->
response.transcriptionJobSummaries().stream())
                .forEach(jobSummary -> {
                    System.out.println("Job Name: " +
jobSummary.transcriptionJobName());
                    System.out.println("Job Status: " +
jobSummary.transcriptionJobStatus());
                    System.out.println("Output Location: " +
jobSummary.outputLocationType());
                    // Add more information as needed

                    // Retrieve additional details for the job if necessary
                    GetTranscriptionJobResponse jobDetails =
transcribeClient.getTranscriptionJob(
                        GetTranscriptionJobRequest.builder()

.transcriptionJobName(jobSummary.transcriptionJobName())
                            .build());

                    // Display additional details
                    System.out.println("Language Code: " +
jobDetails.transcriptionJob().languageCode());
                    System.out.println("Media Format: " +
jobDetails.transcriptionJob().mediaFormat());
                    // Add more details as needed

                    System.out.println("--------------");
                });
        }
    }
```

- For API details, see ListTranscriptionJobs in *AWS SDK for Java 2.x API Reference*.

JavaScript

SDK for JavaScript (v3)

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

List transcription jobs.

```
// Import the required AWS SDK clients and commands for Node.js

import { ListTranscriptionJobsCommand } from "@aws-sdk/client-transcribe";
import { transcribeClient } from "./libs/transcribeClient.js";

// Set the parameters
export const params = {
  JobNameContains: "KEYWORD", // Not required. Returns only transcription
  // job names containing this string
};

export const run = async () => {
  try {
    const data = await transcribeClient.send(
      new ListTranscriptionJobsCommand(params),
    );
    console.log("Success", data.TranscriptionJobSummaries);
    return data; // For unit tests.
  } catch (err) {
    console.log("Error", err);
  }
};
run();
```

Create the client.

```
import { TranscribeClient } from "@aws-sdk/client-transcribe";
// Set the AWS Region.
const REGION = "REGION"; //e.g. "us-east-1"
```

```
// Create an Amazon Transcribe service client object.
const transcribeClient = new TranscribeClient({ region: REGION });
export { transcribeClient };
```

- For more information, see AWS SDK for JavaScript Developer Guide.
- For API details, see ListTranscriptionJobs in *AWS SDK for JavaScript API Reference*.

Python

SDK for Python (Boto3)

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
def list_jobs(job_filter, transcribe_client):
    """
    Lists summaries of the transcription jobs for the current AWS account.

    :param job_filter: The list of returned jobs must contain this string in
 their
                       names.
    :param transcribe_client: The Boto3 Transcribe client.
    :return: The list of retrieved transcription job summaries.
    """
    try:
        response =
 transcribe_client.list_transcription_jobs(JobNameContains=job_filter)
        jobs = response["TranscriptionJobSummaries"]
        next_token = response.get("NextToken")
        while next_token is not None:
            response = transcribe_client.list_transcription_jobs(
                JobNameContains=job_filter, NextToken=next_token
            )
            jobs += response["TranscriptionJobSummaries"]
            next_token = response.get("NextToken")
        logger.info("Got %s jobs with filter %s.", len(jobs), job_filter)
    except ClientError:
```

```
            logger.exception("Couldn't get jobs with filter %s.", job_filter)
            raise
        else:
            return jobs
```

- For API details, see ListTranscriptionJobs in *AWS SDK for Python (Boto3) API Reference*.

SAP ABAP

SDK for SAP ABAP

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
    TRY.
        IF iv_job_filter IS NOT INITIAL.
          oo_result = lo_tnb->listtranscriptionjobs( iv_jobnamecontains =
iv_job_filter ).
        ELSE.
          oo_result = lo_tnb->listtranscriptionjobs( ).
        ENDIF.
        MESSAGE 'Retrieved transcription jobs list.' TYPE 'I'.
      CATCH /aws1/cx_tnbbadrequestex INTO DATA(lo_bad_request_ex).
        MESSAGE lo_bad_request_ex TYPE 'I'.
        RAISE EXCEPTION lo_bad_request_ex.
      CATCH /aws1/cx_tnbinternalfailureex INTO DATA(lo_internal_ex).
        MESSAGE lo_internal_ex TYPE 'I'.
        RAISE EXCEPTION lo_internal_ex.
    ENDTRY.
```

- For API details, see ListTranscriptionJobs in *AWS SDK for SAP ABAP API reference*.

For a complete list of AWS SDK developer guides and code examples, see Using this service with an AWS SDK. This topic also includes information about getting started and details about previous SDK versions.

Use `ListVocabularies` with an AWS SDK or CLI

The following code examples show how to use `ListVocabularies`.

Action examples are code excerpts from larger programs and must be run in context. You can see this action in context in the following code example:

- Create and refine a custom vocabulary

.NET

SDK for .NET

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
   /// <summary>
   /// List custom vocabularies for the current account. Optionally specify a
name
   /// filter and a specific state to filter the vocabularies list.
   /// </summary>
   /// <param name="nameContains">Optional string the vocabulary name must
contain.</param>
   /// <param name="stateEquals">Optional state of the vocabulary.</param>
   /// <returns>List of information about the vocabularies.</returns>
   public async Task<List<VocabularyInfo>> ListCustomVocabularies(string?
nameContains = null,
       VocabularyState? stateEquals = null)
   {
       var response = await _amazonTranscribeService.ListVocabulariesAsync(
           new ListVocabulariesRequest()
           {
               NameContains = nameContains,
```

```
                StateEquals = stateEquals
            });
        return response.Vocabularies;
    }
```

- For API details, see ListVocabularies in *AWS SDK for .NET API Reference*.

CLI

AWS CLI

To list your custom vocabularies

The following `list-vocabularies` example lists the custom vocabularies associated with your AWS account and Region.

```
aws transcribe list-vocabularies
```

Output:

```
{
    "NextToken": "NextToken",
    "Vocabularies": [
        {
            "VocabularyName": "ards-test-1",
            "LanguageCode": "language-code",
            "LastModifiedTime": "2020-04-27T22:00:27.330000+00:00",
            "VocabularyState": "READY"
        },
        {
            "VocabularyName": "sample-test",
            "LanguageCode": "language-code",
            "LastModifiedTime": "2020-04-24T23:04:11.044000+00:00",
            "VocabularyState": "READY"
        },
        {
            "VocabularyName": "CRLF-to-LF-test-3-1",
            "LanguageCode": "language-code",
            "LastModifiedTime": "2020-04-24T22:12:22.277000+00:00",
            "VocabularyState": "READY"
```

```
        },
        {
            "VocabularyName": "CRLF-to-LF-test-2",
            "LanguageCode": "language-code",
            "LastModifiedTime": "2020-04-24T21:53:50.455000+00:00",
            "VocabularyState": "READY"
        },
        {
            "VocabularyName": "CRLF-to-LF-1-1",
            "LanguageCode": "language-code",
            "LastModifiedTime": "2020-04-24T21:39:33.356000+00:00",
            "VocabularyState": "READY"
        }
    ]
}
```

For more information, see Custom Vocabularies in the *Amazon Transcribe Developer Guide*.

- For API details, see ListVocabularies in *AWS CLI Command Reference*.

Python

SDK for Python (Boto3)

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
def list_vocabularies(vocabulary_filter, transcribe_client):
    """
    Lists the custom vocabularies created for this AWS account.

    :param vocabulary_filter: The returned vocabularies must contain this string
 in
                              their names.
    :param transcribe_client: The Boto3 Transcribe client.
    :return: The list of retrieved vocabularies.
    """
    try:
```

```
        response =
transcribe_client.list_vocabularies(NameContains=vocabulary_filter)
        vocabs = response["Vocabularies"]
        next_token = response.get("NextToken")
        while next_token is not None:
            response = transcribe_client.list_vocabularies(
                NameContains=vocabulary_filter, NextToken=next_token
            )
            vocabs += response["Vocabularies"]
            next_token = response.get("NextToken")
        logger.info(
            "Got %s vocabularies with filter %s.", len(vocabs), vocabulary_filter
        )
    except ClientError:
        logger.exception(
            "Couldn't list vocabularies with filter %s.", vocabulary_filter
        )
        raise
    else:
        return vocabs
```

- For API details, see ListVocabularies in *AWS SDK for Python (Boto3) API Reference*.

SAP ABAP

SDK for SAP ABAP

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
    TRY.
        IF iv_vocab_filter IS NOT INITIAL.
          oo_result = lo_tnb->listvocabularies( iv_namecontains =
iv_vocab_filter ).
        ELSE.
          oo_result = lo_tnb->listvocabularies( ).
```

```
        ENDIF.
        MESSAGE 'Retrieved vocabularies list.' TYPE 'I'.
      CATCH /aws1/cx_tnbbadrequestex INTO DATA(lo_bad_request_ex).
        MESSAGE lo_bad_request_ex TYPE 'I'.
        RAISE EXCEPTION lo_bad_request_ex.
      CATCH /aws1/cx_tnbinternalfailureex INTO DATA(lo_internal_ex).
        MESSAGE lo_internal_ex TYPE 'I'.
        RAISE EXCEPTION lo_internal_ex.
    ENDTRY.
```

- For API details, see ListVocabularies in *AWS SDK for SAP ABAP API reference*.

For a complete list of AWS SDK developer guides and code examples, see Using this service with an AWS SDK. This topic also includes information about getting started and details about previous SDK versions.

Use `StartMedicalTranscriptionJob` with an AWS SDK or CLI

The following code examples show how to use `StartMedicalTranscriptionJob`.

.NET

SDK for .NET

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
    /// <summary>
    /// Start a medical transcription job for a media file. This method returns
    /// as soon as the job is started.
    /// </summary>
    /// <param name="jobName">A unique name for the medical transcription job.</
param>
    /// <param name="mediaFileUri">The URI of the media file, typically an Amazon
 S3 location.</param>
    /// <param name="mediaFormat">The format of the media file.</param>
```

```
   /// <param name="outputBucketName">Location for the output, typically an
Amazon S3 location.</param>
   /// <param name="transcriptionType">Conversation or dictation transcription
type.</param>
   /// <returns>A MedicalTransactionJob instance with information on the new
job.</returns>
   public async Task<MedicalTranscriptionJob> StartMedicalTranscriptionJob(
       string jobName, string mediaFileUri,
       MediaFormat mediaFormat, string outputBucketName,
Amazon.TranscribeService.Type transcriptionType)
   {
       var response = await
_amazonTranscribeService.StartMedicalTranscriptionJobAsync(
           new StartMedicalTranscriptionJobRequest()
           {
               MedicalTranscriptionJobName = jobName,
               Media = new Media()
               {
                   MediaFileUri = mediaFileUri
               },
               MediaFormat = mediaFormat,
               LanguageCode =
                   LanguageCode
                       .EnUS, // The value must be en-US for medical
transcriptions.
               OutputBucketName = outputBucketName,
               OutputKey =
                   jobName, // The value is a key used to fetch the output of
the transcription.
               Specialty = Specialty.PRIMARYCARE, // The value PRIMARYCARE must
be set.
               Type = transcriptionType
           });
       return response.MedicalTranscriptionJob;
   }
```

- For API details, see StartMedicalTranscriptionJob in *AWS SDK for .NET API Reference*.

CLI

AWS CLI

Example 1: To transcribe a medical dictation stored as an audio file

The following `start-medical-transcription-job` example transcribes an audio file. You specify the location of the transcription output in the `OutputBucketName` parameter.

```
aws transcribe start-medical-transcription-job \
    --cli-input-json file://myfile.json
```

Contents of `myfile.json`:

```
{
    "MedicalTranscriptionJobName": "simple-dictation-medical-transcription-job",
    "LanguageCode": "language-code",
    "Specialty": "PRIMARYCARE",
    "Type": "DICTATION",
    "OutputBucketName":"amzn-s3-demo-bucket",
    "Media": {
        "MediaFileUri": "s3://amzn-s3-demo-bucket/your-audio-file.extension"
    }
}
```

Output:

```
{
    "MedicalTranscriptionJob": {
        "MedicalTranscriptionJobName": "simple-dictation-medical-transcription-
job",
        "TranscriptionJobStatus": "IN_PROGRESS",
        "LanguageCode": "language-code",
        "Media": {
            "MediaFileUri": "s3://amzn-s3-demo-bucket/your-audio-file.extension"
        },
        "StartTime": "2020-09-20T00:35:22.256000+00:00",
        "CreationTime": "2020-09-20T00:35:22.218000+00:00",
        "Specialty": "PRIMARYCARE",
        "Type": "DICTATION"
    }
}
```

For more information, see Transcribing an audio file of a medical conversation in the *Amazon Transcribe Developer Guide*.

Example 2: To transcribe a clinician-patient dialogue stored as an audio file

The following `start-medical-transcription-job` example transcribes an audio file containing a clinician-patient dialogue. You specify the location of the transcription output in the OutputBucketName parameter.

```
aws transcribe start-medical-transcription-job \
    --cli-input-json file://mysecondfile.json
```

Contents of `mysecondfile.json`:

```
{
    "MedicalTranscriptionJobName": "simple-dictation-medical-transcription-job",
    "LanguageCode": "language-code",
    "Specialty": "PRIMARYCARE",
    "Type": "CONVERSATION",
    "OutputBucketName":"amzn-s3-demo-bucket",
    "Media": {
        "MediaFileUri": "s3://amzn-s3-demo-bucket/your-audio-file.extension"
    }
}
```

Output:

```
{
    "MedicalTranscriptionJob": {
        "MedicalTranscriptionJobName": "simple-conversation-medical-
transcription-job",
        "TranscriptionJobStatus": "IN_PROGRESS",
        "LanguageCode": "language-code",
        "Media": {
            "MediaFileUri": "s3://amzn-s3-demo-bucket/your-audio-file.extension"
        },
        "StartTime": "2020-09-20T23:19:49.965000+00:00",
        "CreationTime": "2020-09-20T23:19:49.941000+00:00",
        "Specialty": "PRIMARYCARE",
        "Type": "CONVERSATION"
    }
```

```
}
```

For more information, see Transcribing an audio file of a medical conversation in the *Amazon Transcribe Developer Guide*.

Example 3: To transcribe a multichannel audio file of a clinician-patient dialogue

The following `start-medical-transcription-job` example transcribes the audio from each channel in the audio file and merges the separate transcriptions from each channel into a single transcription output. You specify the location of the transcription output in the `OutputBucketName` parameter.

```
aws transcribe start-medical-transcription-job \
    --cli-input-json file://mythirdfile.json
```

Contents of `mythirdfile.json`:

```
{
    "MedicalTranscriptionJobName": "multichannel-conversation-medical-
transcription-job",
    "LanguageCode": "language-code",
    "Specialty": "PRIMARYCARE",
    "Type": "CONVERSATION",
    "OutputBucketName":"amzn-s3-demo-bucket",
    "Media": {
        "MediaFileUri": "s3://amzn-s3-demo-bucket/your-audio-file.extension"
    },
    "Settings":{
        "ChannelIdentification": true
    }
}
```

Output:

```
{
    "MedicalTranscriptionJob": {
        "MedicalTranscriptionJobName": "multichannel-conversation-medical-
transcription-job",
        "TranscriptionJobStatus": "IN_PROGRESS",
        "LanguageCode": "language-code",
        "Media": {
```

```
            "MediaFileUri": "s3://amzn-s3-demo-bucket/your-audio-file.extension"
        },
        "StartTime": "2020-09-20T23:46:44.081000+00:00",
        "CreationTime": "2020-09-20T23:46:44.053000+00:00",
        "Settings": {
            "ChannelIdentification": true
        },
        "Specialty": "PRIMARYCARE",
        "Type": "CONVERSATION"
    }
}
```

For more information, see Transcribing multi-channel audio in the *Amazon Transcribe Developer Guide*.

Example 4: To transcribe an audio file of a clinician-patient dialogue and identify the speakers in the transcription output

The following `start-medical-transcription-job` example transcribes an audio file and labels the speech of each speaker in the transcription output. You specify the location of the transcription output in the `OutputBucketName` parameter.

```
aws transcribe start-medical-transcription-job \
    --cli-input-json file://myfourthfile.json
```

Contents of `myfourthfile.json`:

```
{
    "MedicalTranscriptionJobName": "speaker-id-conversation-medical-
transcription-job",
    "LanguageCode": "language-code",
    "Specialty": "PRIMARYCARE",
    "Type": "CONVERSATION",
    "OutputBucketName":"amzn-s3-demo-bucket",
    "Media": {
        "MediaFileUri": "s3://amzn-s3-demo-bucket/your-audio-file.extension"
    },
    "Settings":{
        "ShowSpeakerLabels": true,
        "MaxSpeakerLabels": 2
    }
}
```

Output:

```
{
    "MedicalTranscriptionJob": {
        "MedicalTranscriptionJobName": "speaker-id-conversation-medical-
transcription-job",
        "TranscriptionJobStatus": "IN_PROGRESS",
        "LanguageCode": "language-code",
        "Media": {
            "MediaFileUri": "s3://amzn-s3-demo-bucket/your-audio-file.extension"
        },
        "StartTime": "2020-09-21T18:43:37.265000+00:00",
        "CreationTime": "2020-09-21T18:43:37.157000+00:00",
        "Settings": {
            "ShowSpeakerLabels": true,
            "MaxSpeakerLabels": 2
        },
        "Specialty": "PRIMARYCARE",
        "Type": "CONVERSATION"
    }
}
```

For more information, see Enabling speaker partitioning in the *Amazon Transcribe Developer Guide*.

Example 5: To transcribe a medical conversation stored as an audio file with up to two transcription alternatives

The following `start-medical-transcription-job` example creates up to two alternative transcriptions from a single audio file. Every transcriptions has a level of confidence associated with it. By default, Amazon Transcribe returns the transcription with the highest confidence level. You can specify that Amazon Transcribe return additional transcriptions with lower confidence levels. You specify the location of the transcription output in the `OutputBucketName` parameter.

```
aws transcribe start-medical-transcription-job \
    --cli-input-json file://myfifthfile.json
```

Contents of `myfifthfile.json`:

```
{
```

```
    "MedicalTranscriptionJobName": "alternatives-conversation-medical-
transcription-job",
    "LanguageCode": "language-code",
    "Specialty": "PRIMARYCARE",
    "Type": "CONVERSATION",
    "OutputBucketName":"amzn-s3-demo-bucket",
    "Media": {
        "MediaFileUri": "s3://amzn-s3-demo-bucket/your-audio-file.extension"
    },
    "Settings":{
        "ShowAlternatives": true,
        "MaxAlternatives": 2
    }
}
```

Output:

```
{
    "MedicalTranscriptionJob": {
        "MedicalTranscriptionJobName": "alternatives-conversation-medical-
transcription-job",
        "TranscriptionJobStatus": "IN_PROGRESS",
        "LanguageCode": "language-code",
        "Media": {
            "MediaFileUri": "s3://amzn-s3-demo-bucket/your-audio-file.extension"
        },
        "StartTime": "2020-09-21T19:09:18.199000+00:00",
        "CreationTime": "2020-09-21T19:09:18.171000+00:00",
        "Settings": {
            "ShowAlternatives": true,
            "MaxAlternatives": 2
        },
        "Specialty": "PRIMARYCARE",
        "Type": "CONVERSATION"
    }
}
```

For more information, see Alternative transcriptions in the *Amazon Transcribe Developer Guide*.

Example 6: To transcribe an audio file of a medical dictation with up to two alternative transcriptions

The following `start-medical-transcription-job` example transcribes an audio file and uses a vocabulary filter to mask any unwanted words. You specify the location of the transcription output in the OutputBucketName parameter.

```
aws transcribe start-medical-transcription-job \
    --cli-input-json file://mysixthfile.json
```

Contents of `mysixthfile.json`:

```
{
    "MedicalTranscriptionJobName": "alternatives-conversation-medical-
transcription-job",
    "LanguageCode": "language-code",
    "Specialty": "PRIMARYCARE",
    "Type": "DICTATION",
    "OutputBucketName":"amzn-s3-demo-bucket",
    "Media": {
        "MediaFileUri": "s3://amzn-s3-demo-bucket/your-audio-file.extension"
    },
    "Settings":{
        "ShowAlternatives": true,
        "MaxAlternatives": 2
    }
}
```

Output:

```
{
    "MedicalTranscriptionJob": {
        "MedicalTranscriptionJobName": "alternatives-dictation-medical-
transcription-job",
        "TranscriptionJobStatus": "IN_PROGRESS",
        "LanguageCode": "language-code",
        "Media": {
            "MediaFileUri": "s3://amzn-s3-demo-bucket/your-audio-file.extension"
        },
        "StartTime": "2020-09-21T21:01:14.592000+00:00",
        "CreationTime": "2020-09-21T21:01:14.569000+00:00",
        "Settings": {
            "ShowAlternatives": true,
            "MaxAlternatives": 2
        },
```

```
        "Specialty": "PRIMARYCARE",
        "Type": "DICTATION"
    }
}
```

For more information, see Alternative transcriptions in the *Amazon Transcribe Developer Guide*.

Example 7: To transcribe an audio file of a medical dictation with increased accuracy by using a custom vocabulary

The following `start-medical-transcription-job` example transcribes an audio file and uses a medical custom vocabulary you've previously created to increase the transcription accuracy. You specify the location of the transcription output in the `OutputBucketName` parameter.

```
aws transcribe start-medical-transcription-job \
    --cli-input-json file://myseventhfile.json
```

Contents of `myseventhfile.json`:

```
{
    "MedicalTranscriptionJobName": "vocabulary-dictation-medical-transcription-
job",
    "LanguageCode": "language-code",
    "Specialty": "PRIMARYCARE",
    "Type": "DICTATION",
    "OutputBucketName":"amzn-s3-demo-bucket",
    "Media": {
        "MediaFileUri": "s3://amzn-s3-demo-bucket/your-audio-file.extension"
    },
    "Settings":{
        "VocabularyName": "cli-medical-vocab-1"
    }
}
```

Output:

```
{
    "MedicalTranscriptionJob": {
        "MedicalTranscriptionJobName": "vocabulary-dictation-medical-
transcription-job",
```

```
        "TranscriptionJobStatus": "IN_PROGRESS",
        "LanguageCode": "language-code",
        "Media": {
            "MediaFileUri": "s3://amzn-s3-demo-bucket/your-audio-file.extension"
        },
        "StartTime": "2020-09-21T21:17:27.045000+00:00",
        "CreationTime": "2020-09-21T21:17:27.016000+00:00",
        "Settings": {
            "VocabularyName": "cli-medical-vocab-1"
        },
        "Specialty": "PRIMARYCARE",
        "Type": "DICTATION"
    }
}
```

For more information, see Improving transcription accuracy with medical custom vocabularies in the *Amazon Transcribe Developer Guide*.

- For API details, see StartMedicalTranscriptionJob in *AWS CLI Command Reference*.

JavaScript

SDK for JavaScript (v3)

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

Create the client.

```
import { TranscribeClient } from "@aws-sdk/client-transcribe";
// Set the AWS Region.
const REGION = "REGION"; //e.g. "us-east-1"
// Create an Amazon Transcribe service client object.
const transcribeClient = new TranscribeClient({ region: REGION });
export { transcribeClient };
```

Start a medical transcription job.

```
// Import the required AWS SDK clients and commands for Node.js
import { StartMedicalTranscriptionJobCommand } from "@aws-sdk/client-transcribe";
import { transcribeClient } from "./libs/transcribeClient.js";

// Set the parameters
export const params = {
  MedicalTranscriptionJobName: "MEDICAL_JOB_NAME", // Required
  OutputBucketName: "OUTPUT_BUCKET_NAME", // Required
  Specialty: "PRIMARYCARE", // Required. Possible values are 'PRIMARYCARE'
  Type: "JOB_TYPE", // Required. Possible values are 'CONVERSATION' and
 'DICTATION'
  LanguageCode: "LANGUAGE_CODE", // For example, 'en-US'
  MediaFormat: "SOURCE_FILE_FORMAT", // For example, 'wav'
  Media: {
    MediaFileUri: "SOURCE_FILE_LOCATION",
    // The S3 object location of the input media file. The URI must be in the
 same region
    // as the API endpoint that you are calling.For example,
    // "https://transcribe-demo.s3-REGION.amazonaws.com/hello_world.wav"
  },
};

export const run = async () => {
  try {
    const data = await transcribeClient.send(
      new StartMedicalTranscriptionJobCommand(params),
    );
    console.log("Success - put", data);
    return data; // For unit tests.
  } catch (err) {
    console.log("Error", err);
  }
};
run();
```

- For more information, see AWS SDK for JavaScript Developer Guide.
- For API details, see StartMedicalTranscriptionJob in *AWS SDK for JavaScript API Reference*.

For a complete list of AWS SDK developer guides and code examples, see Using this service with an AWS SDK. This topic also includes information about getting started and details about previous SDK versions.

Use `StartTranscriptionJob` with an AWS SDK or CLI

The following code examples show how to use `StartTranscriptionJob`.

Action examples are code excerpts from larger programs and must be run in context. You can see this action in context in the following code examples:

- Create and refine a custom vocabulary
- Transcribe audio and get job data

.NET

SDK for .NET

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
    /// <summary>
    /// Start a transcription job for a media file. This method returns
    /// as soon as the job is started.
    /// </summary>
    /// <param name="jobName">A unique name for the transcription job.</param>
    /// <param name="mediaFileUri">The URI of the media file, typically an Amazon
 S3 location.</param>
    /// <param name="mediaFormat">The format of the media file.</param>
    /// <param name="languageCode">The language code of the media file, such as
 en-US.</param>
    /// <param name="vocabularyName">Optional name of a custom vocabulary.</
param>
    /// <returns>A TranscriptionJob instance with information on the new job.</
returns>
    public async Task<TranscriptionJob> StartTranscriptionJob(string jobName,
 string mediaFileUri,
```

```
        MediaFormat mediaFormat, LanguageCode languageCode, string?
 vocabularyName)
    {
        var response = await _amazonTranscribeService.StartTranscriptionJobAsync(
            new StartTranscriptionJobRequest()
            {
                TranscriptionJobName = jobName,
                Media = new Media()
                {
                    MediaFileUri = mediaFileUri
                },
                MediaFormat = mediaFormat,
                LanguageCode = languageCode,
                Settings = vocabularyName != null ? new Settings()
                {
                    VocabularyName = vocabularyName
                } : null
            });
        return response.TranscriptionJob;
    }
```

- For API details, see StartTranscriptionJob in *AWS SDK for .NET API Reference*.

CLI

AWS CLI

Example 1: To transcribe an audio file

The following `start-transcription-job` example transcribes your audio file.

```
aws transcribe start-transcription-job \
    --cli-input-json file://myfile.json
```

Contents of `myfile.json`:

```
{
    "TranscriptionJobName": "cli-simple-transcription-job",
    "LanguageCode": "the-language-of-your-transcription-job",
    "Media": {
```

```
        "MediaFileUri": "s3://amzn-s3-demo-bucket/Amazon-S3-prefix/your-media-
file-name.file-extension"
    }
}
```

Output:

```
{
    "TranscriptionJob": {
        "TranscriptionJobName": "cli-simple-transcription-job",
        "TranscriptionJobStatus": "IN_PROGRESS",
        "LanguageCode": "the-language-of-your-transcription-job",
        "Media": {
            "MediaFileUri": "s3://amzn-s3-demo-bucket/Amazon-S3-prefix/your-
media-file-name.file-extension"
        },
        "StartTime": "2020-09-17T16:01:42.912000+00:00",
        "CreationTime": "2020-09-17T16:01:42.886000+00:00"
    }
}
```

For more information, see Getting Started (AWS Command Line Interface) in the *Amazon Transcribe Developer Guide*.

Example 2: To transcribe a multi-channel audio file

The following `start-transcription-job` example transcribes your multi-channel audio file.

```
aws transcribe start-transcription-job \
    --cli-input-json file://mysecondfile.json
```

Contents of `mysecondfile.json`:

```
{
    "TranscriptionJobName": "cli-channelid-job",
    "LanguageCode": "the-language-of-your-transcription-job",
    "Media": {
        "MediaFileUri": "s3://amzn-s3-demo-bucket/Amazon-S3-prefix/your-media-
file-name.file-extension"
    },
    "Settings":{
```

```
        "ChannelIdentification":true
    }
}
```

Output:

```
{
    "TranscriptionJob": {
        "TranscriptionJobName": "cli-channelid-job",
        "TranscriptionJobStatus": "IN_PROGRESS",
        "LanguageCode": "the-language-of-your-transcription-job",
        "Media": {
            "MediaFileUri": "s3://amzn-s3-demo-bucket/Amazon-S3-prefix/your-
media-file-name.file-extension"
        },
        "StartTime": "2020-09-17T16:07:56.817000+00:00",
        "CreationTime": "2020-09-17T16:07:56.784000+00:00",
        "Settings": {
            "ChannelIdentification": true
        }
    }
}
```

For more information, see Transcribing Multi-Channel Audio in the *Amazon Transcribe Developer Guide*.

Example 3: To transcribe an audio file and identify the different speakers

The following `start-transcription-job` example transcribes your audio file and identifies the speakers in the transcription output.

```
aws transcribe start-transcription-job \
    --cli-input-json file://mythirdfile.json
```

Contents of `mythirdfile.json`:

```
{
    "TranscriptionJobName": "cli-speakerid-job",
    "LanguageCode": "the-language-of-your-transcription-job",
    "Media": {
        "MediaFileUri": "s3://amzn-s3-demo-bucket/Amazon-S3-prefix/your-media-
file-name.file-extension"
```

```
    },
    "Settings":{
        "ShowSpeakerLabels": true,
        "MaxSpeakerLabels": 2
    }
}
```

Output:

```
{
    "TranscriptionJob": {
        "TranscriptionJobName": "cli-speakerid-job",
        "TranscriptionJobStatus": "IN_PROGRESS",
        "LanguageCode": "the-language-of-your-transcription-job",
        "Media": {
            "MediaFileUri": "s3://amzn-s3-demo-bucket/Amazon-S3-prefix/your-
media-file-name.file-extension"
        },
        "StartTime": "2020-09-17T16:22:59.696000+00:00",
        "CreationTime": "2020-09-17T16:22:59.676000+00:00",
        "Settings": {
            "ShowSpeakerLabels": true,
            "MaxSpeakerLabels": 2
        }
    }
}
```

For more information, see Identifying Speakers in the *Amazon Transcribe Developer Guide*.

Example 4: To transcribe an audio file and mask any unwanted words in the transcription output

The following `start-transcription-job` example transcribes your audio file and uses a vocabulary filter you've previously created to mask any unwanted words.

```
aws transcribe start-transcription-job \
    --cli-input-json file://myfourthfile.json
```

Contents of `myfourthfile.json`:

```
{
    "TranscriptionJobName": "cli-filter-mask-job",
```

```
    "LanguageCode": "the-language-of-your-transcription-job",
    "Media": {
        "MediaFileUri": "s3://amzn-s3-demo-bucket/Amazon-S3-prefix/your-media-
file-name.file-extension"
    },
    "Settings":{
        "VocabularyFilterName": "your-vocabulary-filter",
        "VocabularyFilterMethod": "mask"
    }
}
```

Output:

```
{
    "TranscriptionJob": {
        "TranscriptionJobName": "cli-filter-mask-job",
        "TranscriptionJobStatus": "IN_PROGRESS",
        "LanguageCode": "the-language-of-your-transcription-job",
        "Media": {
            "MediaFileUri": "s3://Amazon-S3-Prefix/your-media-file.file-
extension"
        },
        "StartTime": "2020-09-18T16:36:18.568000+00:00",
        "CreationTime": "2020-09-18T16:36:18.547000+00:00",
        "Settings": {
            "VocabularyFilterName": "your-vocabulary-filter",
            "VocabularyFilterMethod": "mask"
        }
    }
}
```

For more information, see Using custom vocabulary filters to delete, mask, or flag words in the *Amazon Transcribe Developer Guide*.

Example 5: To transcribe an audio file and remove any unwanted words in the transcription output

The following `start-transcription-job` example transcribes your audio file and uses a vocabulary filter you've previously created to mask any unwanted words.

```
aws transcribe start-transcription-job \
    --cli-input-json file://myfifthfile.json
```

Contents of `myfifthfile.json`:

```
{
    "TranscriptionJobName": "cli-filter-remove-job",
    "LanguageCode": "the-language-of-your-transcription-job",
    "Media": {
        "MediaFileUri": "s3://amzn-s3-demo-bucket/Amazon-S3-prefix/your-media-
file-name.file-extension"
    },
    "Settings":{
        "VocabularyFilterName": "your-vocabulary-filter",
        "VocabularyFilterMethod": "remove"
    }
}
```

Output:

```
{
    "TranscriptionJob": {
        "TranscriptionJobName": "cli-filter-remove-job",
        "TranscriptionJobStatus": "IN_PROGRESS",
        "LanguageCode": "the-language-of-your-transcription-job",
        "Media": {
            "MediaFileUri": "s3://amzn-s3-demo-bucket/Amazon-S3-prefix/your-
media-file-name.file-extension"
        },
        "StartTime": "2020-09-18T16:36:18.568000+00:00",
        "CreationTime": "2020-09-18T16:36:18.547000+00:00",
        "Settings": {
            "VocabularyFilterName": "your-vocabulary-filter",
            "VocabularyFilterMethod": "remove"
        }
    }
}
```

For more information, see Using custom vocabulary filters to delete, mask, or flag words in the *Amazon Transcribe Developer Guide*.

Example 6: To transcribe an audio file with increased accuracy using a custom vocabulary

The following `start-transcription-job` example transcribes your audio file and uses a vocabulary filter you've previously created to mask any unwanted words.

```
aws transcribe start-transcription-job \
    --cli-input-json file://mysixthfile.json
```

Contents of `mysixthfile.json`:

```
{
    "TranscriptionJobName": "cli-vocab-job",
    "LanguageCode": "the-language-of-your-transcription-job",
    "Media": {
        "MediaFileUri": "s3://amzn-s3-demo-bucket/Amazon-S3-prefix/your-media-
file-name.file-extension"
    },
    "Settings":{
        "VocabularyName": "your-vocabulary"
    }
}
```

Output:

```
{
    "TranscriptionJob": {
        "TranscriptionJobName": "cli-vocab-job",
        "TranscriptionJobStatus": "IN_PROGRESS",
        "LanguageCode": "the-language-of-your-transcription-job",
        "Media": {
            "MediaFileUri": "s3://amzn-s3-demo-bucket/Amazon-S3-prefix/your-
media-file-name.file-extension"
        },
        "StartTime": "2020-09-18T16:36:18.568000+00:00",
        "CreationTime": "2020-09-18T16:36:18.547000+00:00",
        "Settings": {
            "VocabularyName": "your-vocabulary"
        }
    }
}
```

For more information, see Using custom vocabulary filters to delete, mask, or flag words in the *Amazon Transcribe Developer Guide*.

Example 7: To identify the language of an audio file and transcribe it

The following `start-transcription-job` example transcribes your audio file and uses a vocabulary filter you've previously created to mask any unwanted words.

```
aws transcribe start-transcription-job \
    --cli-input-json file://myseventhfile.json
```

Contents of `myseventhfile.json`:

```
{
    "TranscriptionJobName": "cli-identify-language-transcription-job",
    "IdentifyLanguage": true,
    "Media": {
        "MediaFileUri": "s3://amzn-s3-demo-bucket/Amazon-S3-prefix/your-media-
file-name.file-extension"
    }
}
```

Output:

```
{
    "TranscriptionJob": {
        "TranscriptionJobName": "cli-identify-language-transcription-job",
        "TranscriptionJobStatus": "IN_PROGRESS",
        "Media": {
            "MediaFileUri": "s3://amzn-s3-demo-bucket/Amazon-S3-prefix/your-
media-file-name.file-extension"
        },
        "StartTime": "2020-09-18T22:27:23.970000+00:00",
        "CreationTime": "2020-09-18T22:27:23.948000+00:00",
        "IdentifyLanguage": true
    }
}
```

For more information, see Identifying the Language in the *Amazon Transcribe Developer Guide*.

Example 8: To transcribe an audio file with personally identifiable information redacted

The following `start-transcription-job` example transcribes your audio file and redacts any personally identifiable information in the transcription output.

```
aws transcribe start-transcription-job \
```

```
    --cli-input-json file://myeighthfile.json
```

Contents of `myeighthfile.json`:

```
{
    "TranscriptionJobName": "cli-redaction-job",
    "LanguageCode": "language-code",
    "Media": {
        "MediaFileUri": "s3://Amazon-S3-Prefix/your-media-file.file-extension"
    },
    "ContentRedaction": {
        "RedactionOutput":"redacted",
        "RedactionType":"PII"
    }
}
```

Output:

```
{
    "TranscriptionJob": {
        "TranscriptionJobName": "cli-redaction-job",
        "TranscriptionJobStatus": "IN_PROGRESS",
        "LanguageCode": "language-code",
        "Media": {
            "MediaFileUri": "s3://Amazon-S3-Prefix/your-media-file.file-
extension"
        },
        "StartTime": "2020-09-25T23:49:13.195000+00:00",
        "CreationTime": "2020-09-25T23:49:13.176000+00:00",
        "ContentRedaction": {
            "RedactionType": "PII",
            "RedactionOutput": "redacted"
        }
    }
}
```

For more information, see Automatic Content Redaction in the *Amazon Transcribe Developer Guide*.

Example 9: To generate a transcript with personally identifiable information (PII) redacted and an unredacted transcript

The following `start-transcription-job` example generates two transcrptions of your audio file, one with the personally identifiable information redacted, and the other without any redactions.

```
aws transcribe start-transcription-job \
    --cli-input-json file://myninthfile.json
```

Contents of `myninthfile.json`:

```
{
    "TranscriptionJobName": "cli-redaction-job-with-unredacted-transcript",
    "LanguageCode": "language-code",
    "Media": {
        "MediaFileUri": "s3://Amazon-S3-Prefix/your-media-file.file-extension"
    },
    "ContentRedaction": {
        "RedactionOutput":"redacted_and_unredacted",
        "RedactionType":"PII"
    }
}
```

Output:

```
{
    "TranscriptionJob": {
        "TranscriptionJobName": "cli-redaction-job-with-unredacted-transcript",
        "TranscriptionJobStatus": "IN_PROGRESS",
        "LanguageCode": "language-code",
        "Media": {
            "MediaFileUri": "s3://Amazon-S3-Prefix/your-media-file.file-
extension"
        },
        "StartTime": "2020-09-25T23:59:47.677000+00:00",
        "CreationTime": "2020-09-25T23:59:47.653000+00:00",
        "ContentRedaction": {
            "RedactionType": "PII",
            "RedactionOutput": "redacted_and_unredacted"
        }
    }
}
```

For more information, see Automatic Content Redaction in the *Amazon Transcribe Developer Guide*.

Example 10: To use a custom language model you've previously created to transcribe an audio file.

The following `start-transcription-job` example transcribes your audio file with a custom language model you've previously created.

```
aws transcribe start-transcription-job \
    --cli-input-json file://mytenthfile.json
```

Contents of `mytenthfile.json`:

```
{
    "TranscriptionJobName": "cli-clm-2-job-1",
    "LanguageCode": "language-code",
    "Media": {
        "MediaFileUri": "s3://amzn-s3-demo-bucket/your-audio-file.file-extension"
    },
    "ModelSettings": {
        "LanguageModelName":"cli-clm-2"
    }
}
```

Output:

```
{
    "TranscriptionJob": {
        "TranscriptionJobName": "cli-clm-2-job-1",
        "TranscriptionJobStatus": "IN_PROGRESS",
        "LanguageCode": "language-code",
        "Media": {
            "MediaFileUri": "s3://amzn-s3-demo-bucket/your-audio-file.file-
extension"
        },
        "StartTime": "2020-09-28T17:56:01.835000+00:00",
        "CreationTime": "2020-09-28T17:56:01.801000+00:00",
        "ModelSettings": {
            "LanguageModelName": "cli-clm-2"
        }
    }
```

```
}
```

For more information, see Improving Domain-Specific Transcription Accuracy with Custom Language Models in the *Amazon Transcribe Developer Guide*.

- For API details, see StartTranscriptionJob in *AWS CLI Command Reference*.

JavaScript

SDK for JavaScript (v3)

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

Start a transcription job.

```
// Import the required AWS SDK clients and commands for Node.js
import { StartTranscriptionJobCommand } from "@aws-sdk/client-transcribe";
import { transcribeClient } from "./libs/transcribeClient.js";

// Set the parameters
export const params = {
  TranscriptionJobName: "JOB_NAME",
  LanguageCode: "LANGUAGE_CODE", // For example, 'en-US'
  MediaFormat: "SOURCE_FILE_FORMAT", // For example, 'wav'
  Media: {
    MediaFileUri: "SOURCE_LOCATION",
    // For example, "https://transcribe-demo.s3-REGION.amazonaws.com/
hello_world.wav"
  },
  OutputBucketName: "OUTPUT_BUCKET_NAME",
};

export const run = async () => {
  try {
    const data = await transcribeClient.send(
      new StartTranscriptionJobCommand(params),
    );
    console.log("Success - put", data);
```

```
    return data; // For unit tests.
  } catch (err) {
    console.log("Error", err);
  }
};
run();
```

Create the client.

```
import { TranscribeClient } from "@aws-sdk/client-transcribe";
// Set the AWS Region.
const REGION = "REGION"; //e.g. "us-east-1"
// Create an Amazon Transcribe service client object.
const transcribeClient = new TranscribeClient({ region: REGION });
export { transcribeClient };
```

- For more information, see AWS SDK for JavaScript Developer Guide.
- For API details, see StartTranscriptionJob in *AWS SDK for JavaScript API Reference*.

Python

SDK for Python (Boto3)

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
def start_job(
    job_name,
    media_uri,
    media_format,
    language_code,
    transcribe_client,
    vocabulary_name=None,
):
    """
```

```
    Starts a transcription job. This function returns as soon as the job is
started.
    To get the current status of the job, call get_transcription_job. The job is
    successfully completed when the job status is 'COMPLETED'.

    :param job_name: The name of the transcription job. This must be unique for
                     your AWS account.
    :param media_uri: The URI where the audio file is stored. This is typically
                      in an Amazon S3 bucket.
    :param media_format: The format of the audio file. For example, mp3 or wav.
    :param language_code: The language code of the audio file.
                          For example, en-US or ja-JP
    :param transcribe_client: The Boto3 Transcribe client.
    :param vocabulary_name: The name of a custom vocabulary to use when
transcribing
                            the audio file.
    :return: Data about the job.
    """
    try:
        job_args = {
            "TranscriptionJobName": job_name,
            "Media": {"MediaFileUri": media_uri},
            "MediaFormat": media_format,
            "LanguageCode": language_code,
        }
        if vocabulary_name is not None:
            job_args["Settings"] = {"VocabularyName": vocabulary_name}
        response = transcribe_client.start_transcription_job(**job_args)
        job = response["TranscriptionJob"]
        logger.info("Started transcription job %s.", job_name)
    except ClientError:
        logger.exception("Couldn't start transcription job %s.", job_name)
        raise
    else:
        return job
```

- For API details, see StartTranscriptionJob in *AWS SDK for Python (Boto3) API Reference*.

SAP ABAP

SDK for SAP ABAP

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
    TRY.
        DATA(lo_media) = NEW /aws1/cl_tnbmedia( iv_mediafileuri = iv_media_uri ).
        DATA(lo_settings) = NEW /aws1/cl_tnbsettings( ).
        IF iv_vocabulary_name IS NOT INITIAL.
          lo_settings = NEW /aws1/cl_tnbsettings( iv_vocabularyname =
iv_vocabulary_name ).
        ENDIF.

        oo_result = lo_tnb->starttranscriptionjob(
          iv_transcriptionjobname = iv_job_name
          io_media = lo_media
          iv_mediaformat = iv_media_format
          iv_languagecode = iv_language_code
          io_settings = lo_settings ).

        MESSAGE 'Transcription job started.' TYPE 'I'.
      CATCH /aws1/cx_tnbbadrequestex INTO DATA(lo_bad_request_ex).
        MESSAGE lo_bad_request_ex TYPE 'I'.
        RAISE EXCEPTION lo_bad_request_ex.
      CATCH /aws1/cx_tnblimitexceededex INTO DATA(lo_limit_ex).
        MESSAGE lo_limit_ex TYPE 'I'.
        RAISE EXCEPTION lo_limit_ex.
      CATCH /aws1/cx_tnbinternalfailureex INTO DATA(lo_internal_ex).
        MESSAGE lo_internal_ex TYPE 'I'.
        RAISE EXCEPTION lo_internal_ex.
      CATCH /aws1/cx_tnbconflictexception INTO DATA(lo_conflict_ex).
        MESSAGE lo_conflict_ex TYPE 'I'.
        RAISE EXCEPTION lo_conflict_ex.
    ENDTRY.
```

- For API details, see StartTranscriptionJob in *AWS SDK for SAP ABAP API reference*.

For a complete list of AWS SDK developer guides and code examples, see Using this service with an AWS SDK. This topic also includes information about getting started and details about previous SDK versions.

Use `UpdateVocabulary` with an AWS SDK or CLI

The following code examples show how to use `UpdateVocabulary`.

Action examples are code excerpts from larger programs and must be run in context. You can see this action in context in the following code example:

- Create and refine a custom vocabulary

.NET

SDK for .NET

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
   /// <summary>
   /// Update a custom vocabulary with new values. Update overwrites all
existing information.
   /// </summary>
   /// <param name="languageCode">The language code of the vocabulary.</param>
   /// <param name="phrases">Phrases to use in the vocabulary.</param>
   /// <param name="vocabularyName">Name for the vocabulary.</param>
   /// <returns>The state of the custom vocabulary.</returns>
   public async Task<VocabularyState> UpdateCustomVocabulary(LanguageCode
languageCode,
       List<string> phrases, string vocabularyName)
   {
       var response = await _amazonTranscribeService.UpdateVocabularyAsync(
           new UpdateVocabularyRequest()
           {
               LanguageCode = languageCode,
               Phrases = phrases,
```

```
                VocabularyName = vocabularyName
            });
        return response.VocabularyState;
    }
```

- For API details, see UpdateVocabulary in *AWS SDK for .NET API Reference*.

CLI

AWS CLI

To update a custom vocabulary with new terms.

The following `update-vocabulary` example overwrites the terms used to create a custom vocabulary with the new ones that you provide. Prerequisite: to replace the terms in a custom vocabulary, you need a file with new terms.

```
aws transcribe update-vocabulary \
    --vocabulary-file-uri s3://amzn-s3-demo-bucket/Amazon-S3-Prefix/custom-vocabulary.txt \
    --vocabulary-name custom-vocabulary \
    --language-code language-code
```

Output:

```
{
    "VocabularyName": "custom-vocabulary",
    "LanguageCode": "language",
    "VocabularyState": "PENDING"
}
```

For more information, see Custom Vocabularies in the *Amazon Transcribe Developer Guide*.

- For API details, see UpdateVocabulary in *AWS CLI Command Reference*.

Python

SDK for Python (Boto3)

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
def update_vocabulary(
    vocabulary_name, language_code, transcribe_client, phrases=None,
 table_uri=None
):
    """
    Updates an existing custom vocabulary. The entire vocabulary is replaced with
    the contents of the update.

    :param vocabulary_name: The name of the vocabulary to update.
    :param language_code: The language code of the vocabulary.
    :param transcribe_client: The Boto3 Transcribe client.
    :param phrases: A list of comma-separated phrases to include in the
 vocabulary.
    :param table_uri: A table of phrases and pronunciation hints to include in
 the
                      vocabulary.
    """
    try:
        vocab_args = {"VocabularyName": vocabulary_name, "LanguageCode":
 language_code}
        if phrases is not None:
            vocab_args["Phrases"] = phrases
        elif table_uri is not None:
            vocab_args["VocabularyFileUri"] = table_uri
        response = transcribe_client.update_vocabulary(**vocab_args)
        logger.info("Updated custom vocabulary %s.", response["VocabularyName"])
    except ClientError:
        logger.exception("Couldn't update custom vocabulary %s.",
 vocabulary_name)
        raise
```

- For API details, see UpdateVocabulary in *AWS SDK for Python (Boto3) API Reference*.

SAP ABAP

SDK for SAP ABAP

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
TRY.
    IF it_phrases IS NOT INITIAL.
      oo_result = lo_tnb->updatevocabulary(
        iv_vocabularyname = iv_vocabulary_name
        iv_languagecode = iv_language_code
        it_phrases = it_phrases ).
    ELSEIF iv_vocab_file_uri IS NOT INITIAL.
      oo_result = lo_tnb->updatevocabulary(
        iv_vocabularyname = iv_vocabulary_name
        iv_languagecode = iv_language_code
        iv_vocabularyfileuri = iv_vocab_file_uri ).
    ENDIF.
    MESSAGE 'Vocabulary updated.' TYPE 'I'.
  CATCH /aws1/cx_tnbbadrequestex INTO DATA(lo_bad_request_ex).
    MESSAGE lo_bad_request_ex TYPE 'I'.
  CATCH /aws1/cx_tnblimitexceededex INTO DATA(lo_limit_ex).
    MESSAGE lo_limit_ex TYPE 'I'.
    RAISE EXCEPTION lo_limit_ex.
  CATCH /aws1/cx_tnbnotfoundexception INTO DATA(lo_not_found_ex).
    MESSAGE lo_not_found_ex TYPE 'I'.
  CATCH /aws1/cx_tnbinternalfailureex INTO DATA(lo_internal_ex).
    MESSAGE lo_internal_ex TYPE 'I'.
    RAISE EXCEPTION lo_internal_ex.
  CATCH /aws1/cx_tnbconflictexception INTO DATA(lo_conflict_ex).
    MESSAGE lo_conflict_ex TYPE 'I'.
    RAISE EXCEPTION lo_conflict_ex.
ENDTRY.
```

- For API details, see UpdateVocabulary in *AWS SDK for SAP ABAP API reference*.

For a complete list of AWS SDK developer guides and code examples, see Using this service with an AWS SDK. This topic also includes information about getting started and details about previous SDK versions.

Scenarios for Amazon Transcribe using AWS SDKs

The following code examples show you how to implement common scenarios in Amazon Transcribe with AWS SDKs. These scenarios show you how to accomplish specific tasks by calling multiple functions within Amazon Transcribe or combined with other AWS services. Each scenario includes a link to the complete source code, where you can find instructions on how to set up and run the code.

Scenarios target an intermediate level of experience to help you understand service actions in context.

Examples

- Build an Amazon Transcribe streaming app
- Convert text to speech and back to text using an AWS SDK
- Create and refine an Amazon Transcribe custom vocabulary using an AWS SDK
- Transcribe audio and get job data with Amazon Transcribe using an AWS SDK

Build an Amazon Transcribe streaming app

The following code example shows how to build an app that records, transcribes, and translates live audio in real-time, and emails the results.

JavaScript

SDK for JavaScript (v3)

Shows how to use Amazon Transcribe to build an app that records, transcribes, and translates live audio in real-time, and emails the results using Amazon Simple Email Service (Amazon SES).

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

Services used in this example

- Amazon Comprehend
- Amazon SES
- Amazon Transcribe
- Amazon Translate

For a complete list of AWS SDK developer guides and code examples, see Using this service with an AWS SDK. This topic also includes information about getting started and details about previous SDK versions.

Convert text to speech and back to text using an AWS SDK

The following code example shows how to:

- Use Amazon Polly to synthesize a plain text (UTF-8) input file to an audio file.
- Upload the audio file to an Amazon S3 bucket.
- Use Amazon Transcribe to convert the audio file to text.
- Display the text.

Rust

SDK for Rust

Use Amazon Polly to synthesize a plain text (UTF-8) input file to an audio file, upload the audio file to an Amazon S3 bucket, use Amazon Transcribe to convert that audio file to text, and display the text.

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

Services used in this example

- Amazon Polly
- Amazon S3
- Amazon Transcribe

For a complete list of AWS SDK developer guides and code examples, see Using this service with an AWS SDK. This topic also includes information about getting started and details about previous SDK versions.

Create and refine an Amazon Transcribe custom vocabulary using an AWS SDK

The following code example shows how to:

- Upload an audio file to Amazon S3.
- Run an Amazon Transcribe job to transcribe the file and get the results.
- Create and refine a custom vocabulary to improve transcription accuracy.
- Run jobs with custom vocabularies and get the results.

Python

SDK for Python (Boto3)

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

Transcribe an audio file that contains a reading of Jabberwocky by Lewis Carroll. Start by creating functions that wrap Amazon Transcribe actions.

```
def start_job(
    job_name,
    media_uri,
    media_format,
    language_code,
    transcribe_client,
    vocabulary_name=None,
):
    """
    Starts a transcription job. This function returns as soon as the job is
 started.
    To get the current status of the job, call get_transcription_job. The job is
```

```
    successfully completed when the job status is 'COMPLETED'.

    :param job_name: The name of the transcription job. This must be unique for
                     your AWS account.
    :param media_uri: The URI where the audio file is stored. This is typically
                      in an Amazon S3 bucket.
    :param media_format: The format of the audio file. For example, mp3 or wav.
    :param language_code: The language code of the audio file.
                          For example, en-US or ja-JP
    :param transcribe_client: The Boto3 Transcribe client.
    :param vocabulary_name: The name of a custom vocabulary to use when
 transcribing
                            the audio file.
    :return: Data about the job.
    """
    try:
        job_args = {
            "TranscriptionJobName": job_name,
            "Media": {"MediaFileUri": media_uri},
            "MediaFormat": media_format,
            "LanguageCode": language_code,
        }
        if vocabulary_name is not None:
            job_args["Settings"] = {"VocabularyName": vocabulary_name}
        response = transcribe_client.start_transcription_job(**job_args)
        job = response["TranscriptionJob"]
        logger.info("Started transcription job %s.", job_name)
    except ClientError:
        logger.exception("Couldn't start transcription job %s.", job_name)
        raise
    else:
        return job



def get_job(job_name, transcribe_client):
    """
    Gets details about a transcription job.

    :param job_name: The name of the job to retrieve.
    :param transcribe_client: The Boto3 Transcribe client.
    :return: The retrieved transcription job.
    """
    try:
```

```
        response = transcribe_client.get_transcription_job(
            TranscriptionJobName=job_name
        )
        job = response["TranscriptionJob"]
        logger.info("Got job %s.", job["TranscriptionJobName"])
    except ClientError:
        logger.exception("Couldn't get job %s.", job_name)
        raise
    else:
        return job



def delete_job(job_name, transcribe_client):
    """
    Deletes a transcription job. This also deletes the transcript associated with
    the job.

    :param job_name: The name of the job to delete.
    :param transcribe_client: The Boto3 Transcribe client.
    """
    try:
        transcribe_client.delete_transcription_job(TranscriptionJobName=job_name)
        logger.info("Deleted job %s.", job_name)
    except ClientError:
        logger.exception("Couldn't delete job %s.", job_name)
        raise



def create_vocabulary(
    vocabulary_name, language_code, transcribe_client, phrases=None,
 table_uri=None
):
    """
    Creates a custom vocabulary that can be used to improve the accuracy of
    transcription jobs. This function returns as soon as the vocabulary
 processing
    is started. Call get_vocabulary to get the current status of the vocabulary.
    The vocabulary is ready to use when its status is 'READY'.

    :param vocabulary_name: The name of the custom vocabulary.
    :param language_code: The language code of the vocabulary.
                          For example, en-US or nl-NL.
```

```
    :param transcribe_client: The Boto3 Transcribe client.
    :param phrases: A list of comma-separated phrases to include in the
 vocabulary.
    :param table_uri: A table of phrases and pronunciation hints to include in
 the
                      vocabulary.
    :return: Information about the newly created vocabulary.
    """
    try:
        vocab_args = {"VocabularyName": vocabulary_name, "LanguageCode":
 language_code}
        if phrases is not None:
            vocab_args["Phrases"] = phrases
        elif table_uri is not None:
            vocab_args["VocabularyFileUri"] = table_uri
        response = transcribe_client.create_vocabulary(**vocab_args)
        logger.info("Created custom vocabulary %s.", response["VocabularyName"])
    except ClientError:
        logger.exception("Couldn't create custom vocabulary %s.",
 vocabulary_name)
        raise
    else:
        return response



def get_vocabulary(vocabulary_name, transcribe_client):
    """
    Gets information about a custom vocabulary.

    :param vocabulary_name: The name of the vocabulary to retrieve.
    :param transcribe_client: The Boto3 Transcribe client.
    :return: Information about the vocabulary.
    """
    try:
        response =
 transcribe_client.get_vocabulary(VocabularyName=vocabulary_name)
        logger.info("Got vocabulary %s.", response["VocabularyName"])
    except ClientError:
        logger.exception("Couldn't get vocabulary %s.", vocabulary_name)
        raise
    else:
        return response
```

```
def update_vocabulary(
    vocabulary_name, language_code, transcribe_client, phrases=None,
 table_uri=None
):
    """
    Updates an existing custom vocabulary. The entire vocabulary is replaced with
    the contents of the update.

    :param vocabulary_name: The name of the vocabulary to update.
    :param language_code: The language code of the vocabulary.
    :param transcribe_client: The Boto3 Transcribe client.
    :param phrases: A list of comma-separated phrases to include in the
 vocabulary.
    :param table_uri: A table of phrases and pronunciation hints to include in
 the
                      vocabulary.
    """
    try:
        vocab_args = {"VocabularyName": vocabulary_name, "LanguageCode":
 language_code}
        if phrases is not None:
            vocab_args["Phrases"] = phrases
        elif table_uri is not None:
            vocab_args["VocabularyFileUri"] = table_uri
        response = transcribe_client.update_vocabulary(**vocab_args)
        logger.info("Updated custom vocabulary %s.", response["VocabularyName"])
    except ClientError:
        logger.exception("Couldn't update custom vocabulary %s.",
 vocabulary_name)
        raise



def list_vocabularies(vocabulary_filter, transcribe_client):
    """
    Lists the custom vocabularies created for this AWS account.

    :param vocabulary_filter: The returned vocabularies must contain this string
 in
                              their names.
    :param transcribe_client: The Boto3 Transcribe client.
    :return: The list of retrieved vocabularies.
```

```
    """
    try:
        response =
 transcribe_client.list_vocabularies(NameContains=vocabulary_filter)
        vocabs = response["Vocabularies"]
        next_token = response.get("NextToken")
        while next_token is not None:
            response = transcribe_client.list_vocabularies(
                NameContains=vocabulary_filter, NextToken=next_token
            )
            vocabs += response["Vocabularies"]
            next_token = response.get("NextToken")
        logger.info(
            "Got %s vocabularies with filter %s.", len(vocabs), vocabulary_filter
        )
    except ClientError:
        logger.exception(
            "Couldn't list vocabularies with filter %s.", vocabulary_filter
        )
        raise
    else:
        return vocabs



def delete_vocabulary(vocabulary_name, transcribe_client):
    """
    Deletes a custom vocabulary.

    :param vocabulary_name: The name of the vocabulary to delete.
    :param transcribe_client: The Boto3 Transcribe client.
    """
    try:
        transcribe_client.delete_vocabulary(VocabularyName=vocabulary_name)
        logger.info("Deleted vocabulary %s.", vocabulary_name)
    except ClientError:
        logger.exception("Couldn't delete vocabulary %s.", vocabulary_name)
        raise
```

Call the wrapper functions to transcribe audio without a custom vocabulary and then with different versions of a custom vocabulary to see improved results.

```
def usage_demo():
    """Shows how to use the Amazon Transcribe service."""
    logging.basicConfig(level=logging.INFO, format="%(levelname)s: %(message)s")

    s3_resource = boto3.resource("s3")
    transcribe_client = boto3.client("transcribe")

    print("-" * 88)
    print("Welcome to the Amazon Transcribe demo!")
    print("-" * 88)

    bucket_name = f"jabber-bucket-{time.time_ns()}"
    print(f"Creating bucket {bucket_name}.")
    bucket = s3_resource.create_bucket(
        Bucket=bucket_name,
        CreateBucketConfiguration={
            "LocationConstraint": transcribe_client.meta.region_name
        },
    )
    media_file_name = ".media/Jabberwocky.mp3"
    media_object_key = "Jabberwocky.mp3"
    print(f"Uploading media file {media_file_name}.")
    bucket.upload_file(media_file_name, media_object_key)
    media_uri = f"s3://{bucket.name}/{media_object_key}"

    job_name_simple = f"Jabber-{time.time_ns()}"
    print(f"Starting transcription job {job_name_simple}.")
    start_job(
        job_name_simple,
        f"s3://{bucket_name}/{media_object_key}",
        "mp3",
        "en-US",
        transcribe_client,
    )
    transcribe_waiter = TranscribeCompleteWaiter(transcribe_client)
    transcribe_waiter.wait(job_name_simple)
    job_simple = get_job(job_name_simple, transcribe_client)
    transcript_simple = requests.get(
        job_simple["Transcript"]["TranscriptFileUri"]
    ).json()
```

```
    print(f"Transcript for job {transcript_simple['jobName']}:")
    print(transcript_simple["results"]["transcripts"][0]["transcript"])

    print("-" * 88)
    print(
        "Creating a custom vocabulary that lists the nonsense words to try to "
        "improve the transcription."
    )
    vocabulary_name = f"Jabber-vocabulary-{time.time_ns()}"
    create_vocabulary(
        vocabulary_name,
        "en-US",
        transcribe_client,
        phrases=[
            "brillig",
            "slithy",
            "borogoves",
            "mome",
            "raths",
            "Jub-Jub",
            "frumious",
            "manxome",
            "Tumtum",
            "uffish",
            "whiffling",
            "tulgey",
            "thou",
            "frabjous",
            "callooh",
            "callay",
            "chortled",
        ],
    )
    vocabulary_ready_waiter = VocabularyReadyWaiter(transcribe_client)
    vocabulary_ready_waiter.wait(vocabulary_name)

    job_name_vocabulary_list = f"Jabber-vocabulary-list-{time.time_ns()}"
    print(f"Starting transcription job {job_name_vocabulary_list}.")
    start_job(
        job_name_vocabulary_list,
        media_uri,
        "mp3",
        "en-US",
        transcribe_client,
```

```
        vocabulary_name,
    )
    transcribe_waiter.wait(job_name_vocabulary_list)
    job_vocabulary_list = get_job(job_name_vocabulary_list, transcribe_client)
    transcript_vocabulary_list = requests.get(
        job_vocabulary_list["Transcript"]["TranscriptFileUri"]
    ).json()
    print(f"Transcript for job {transcript_vocabulary_list['jobName']}:")
    print(transcript_vocabulary_list["results"]["transcripts"][0]["transcript"])

    print("-" * 88)
    print(
        "Updating the custom vocabulary with table data that provides additional
"
        "pronunciation hints."
    )
    table_vocab_file = "jabber-vocabulary-table.txt"
    bucket.upload_file(table_vocab_file, table_vocab_file)
    update_vocabulary(
        vocabulary_name,
        "en-US",
        transcribe_client,
        table_uri=f"s3://{bucket.name}/{table_vocab_file}",
    )
    vocabulary_ready_waiter.wait(vocabulary_name)

    job_name_vocab_table = f"Jabber-vocab-table-{time.time_ns()}"
    print(f"Starting transcription job {job_name_vocab_table}.")
    start_job(
        job_name_vocab_table,
        media_uri,
        "mp3",
        "en-US",
        transcribe_client,
        vocabulary_name=vocabulary_name,
    )
    transcribe_waiter.wait(job_name_vocab_table)
    job_vocab_table = get_job(job_name_vocab_table, transcribe_client)
    transcript_vocab_table = requests.get(
        job_vocab_table["Transcript"]["TranscriptFileUri"]
    ).json()
    print(f"Transcript for job {transcript_vocab_table['jobName']}:")
    print(transcript_vocab_table["results"]["transcripts"][0]["transcript"])
```

```
    print("-" * 88)
    print("Getting data for jobs and vocabularies.")
    jabber_jobs = list_jobs("Jabber", transcribe_client)
    print(f"Found {len(jabber_jobs)} jobs:")
    for job_sum in jabber_jobs:
        job = get_job(job_sum["TranscriptionJobName"], transcribe_client)
        print(
            f"\t{job['TranscriptionJobName']}, {job['Media']['MediaFileUri']}, "
            f"{job['Settings'].get('VocabularyName')}"
        )

    jabber_vocabs = list_vocabularies("Jabber", transcribe_client)
    print(f"Found {len(jabber_vocabs)} vocabularies:")
    for vocab_sum in jabber_vocabs:
        vocab = get_vocabulary(vocab_sum["VocabularyName"], transcribe_client)
        vocab_content = requests.get(vocab["DownloadUri"]).text
        print(f"\t{vocab['VocabularyName']} contents:")
        print(vocab_content)

    print("-" * 88)
    print("Deleting demo jobs.")
    for job_name in [job_name_simple, job_name_vocabulary_list,
job_name_vocab_table]:
        delete_job(job_name, transcribe_client)
    print("Deleting demo vocabulary.")
    delete_vocabulary(vocabulary_name, transcribe_client)
    print("Deleting demo bucket.")
    bucket.objects.delete()
    bucket.delete()
    print("Thanks for watching!")
```

- For API details, see the following topics in *AWS SDK for Python (Boto3) API Reference*.
  - CreateVocabulary
  - DeleteTranscriptionJob
  - DeleteVocabulary
  - GetTranscriptionJob
  - GetVocabulary
  - ListVocabularies

- StartTranscriptionJob
- UpdateVocabulary

For a complete list of AWS SDK developer guides and code examples, see Using this service with an AWS SDK. This topic also includes information about getting started and details about previous SDK versions.

Transcribe audio and get job data with Amazon Transcribe using an AWS SDK

The following code examples show how to:

- Start a transcription job with Amazon Transcribe.
- Wait for the job to complete.
- Get the URI where the transcript is stored.

For more information, see Getting started with Amazon Transcribe.

Java

SDK for Java 2.x

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

Transcribes a PCM file.

```
/**
 * To run this AWS code example, ensure that you have set up your development
 * environment, including your AWS credentials.
 *
 * For information, see this documentation topic:
 *
 * https://docs.aws.amazon.com/sdk-for-java/latest/developer-guide/get-
started.html
```

```
 */

public class TranscribeStreamingDemoFile {
    private static final Region REGION = Region.US_EAST_1;
    private static TranscribeStreamingAsyncClient client;

    public static void main(String args[]) throws ExecutionException,
 InterruptedException {

        final String USAGE = "\n" +
                "Usage:\n" +
                "    <file> \n\n" +
                "Where:\n" +
                "    file - the location of a PCM file to transcribe. In this
 example, ensure the PCM file is 16 hertz (Hz). \n";

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

        String file = args[0];
        client = TranscribeStreamingAsyncClient.builder()
                .region(REGION)
                .build();

        CompletableFuture<Void> result =
 client.startStreamTranscription(getRequest(16_000),
                new AudioStreamPublisher(getStreamFromFile(file)),
                getResponseHandler());

        result.get();
        client.close();
    }

    private static InputStream getStreamFromFile(String file) {
        try {
            File inputFile = new File(file);
            InputStream audioStream = new FileInputStream(inputFile);
            return audioStream;

        } catch (FileNotFoundException e) {
            throw new RuntimeException(e);
        }
```

```
    }

    private static StartStreamTranscriptionRequest getRequest(Integer
 mediaSampleRateHertz) {
        return StartStreamTranscriptionRequest.builder()
                .languageCode(LanguageCode.EN_US)
                .mediaEncoding(MediaEncoding.PCM)
                .mediaSampleRateHertz(mediaSampleRateHertz)
                .build();
    }

    private static StartStreamTranscriptionResponseHandler getResponseHandler() {
        return StartStreamTranscriptionResponseHandler.builder()
                .onResponse(r -> {
                    System.out.println("Received Initial response");
                })
                .onError(e -> {
                    System.out.println(e.getMessage());
                    StringWriter sw = new StringWriter();
                    e.printStackTrace(new PrintWriter(sw));
                    System.out.println("Error Occurred: " + sw.toString());
                })
                .onComplete(() -> {
                    System.out.println("=== All records stream successfully
 ===");
                })
                .subscriber(event -> {
                    List<Result> results = ((TranscriptEvent)
 event).transcript().results();
                    if (results.size() > 0) {
                        if (!
results.get(0).alternatives().get(0).transcript().isEmpty()) {

 System.out.println(results.get(0).alternatives().get(0).transcript());
                        }
                    }
                })
                .build();
    }

    private static class AudioStreamPublisher implements Publisher<AudioStream> {
        private final InputStream inputStream;
        private static Subscription currentSubscription;
```

```
        private AudioStreamPublisher(InputStream inputStream) {
            this.inputStream = inputStream;
        }

        @Override
        public void subscribe(Subscriber<? super AudioStream> s) {

            if (this.currentSubscription == null) {
                this.currentSubscription = new SubscriptionImpl(s, inputStream);
            } else {
                this.currentSubscription.cancel();
                this.currentSubscription = new SubscriptionImpl(s, inputStream);
            }
            s.onSubscribe(currentSubscription);
        }
    }

    public static class SubscriptionImpl implements Subscription {
        private static final int CHUNK_SIZE_IN_BYTES = 1024 * 1;
        private final Subscriber<? super AudioStream> subscriber;
        private final InputStream inputStream;
        private ExecutorService executor = Executors.newFixedThreadPool(1);
        private AtomicLong demand = new AtomicLong(0);

        SubscriptionImpl(Subscriber<? super AudioStream> s, InputStream
inputStream) {
            this.subscriber = s;
            this.inputStream = inputStream;
        }

        @Override
        public void request(long n) {
            if (n <= 0) {
                subscriber.onError(new IllegalArgumentException("Demand must be
positive"));
            }

            demand.getAndAdd(n);

            executor.submit(() -> {
                try {
                    do {
                        ByteBuffer audioBuffer = getNextEvent();
                        if (audioBuffer.remaining() > 0) {
```

```
                            AudioEvent audioEvent =
audioEventFromBuffer(audioBuffer);
                            subscriber.onNext(audioEvent);
                        } else {
                            subscriber.onComplete();
                            break;
                        }
                    } while (demand.decrementAndGet() > 0);
                } catch (Exception e) {
                    subscriber.onError(e);
                }
            });
        }

        @Override
        public void cancel() {
            executor.shutdown();
        }

        private ByteBuffer getNextEvent() {
            ByteBuffer audioBuffer = null;
            byte[] audioBytes = new byte[CHUNK_SIZE_IN_BYTES];

            int len = 0;
            try {
                len = inputStream.read(audioBytes);

                if (len <= 0) {
                    audioBuffer = ByteBuffer.allocate(0);
                } else {
                    audioBuffer = ByteBuffer.wrap(audioBytes, 0, len);
                }
            } catch (IOException e) {
                throw new UncheckedIOException(e);
            }

            return audioBuffer;
        }

        private AudioEvent audioEventFromBuffer(ByteBuffer bb) {
            return AudioEvent.builder()
                    .audioChunk(SdkBytes.fromByteBuffer(bb))
                    .build();
        }
```

```
    }
}
```

Transcribes streaming audio from your computer's microphone.

```
public class TranscribeStreamingDemoApp {
    private static final Region REGION = Region.US_EAST_1;
    private static TranscribeStreamingAsyncClient client;

    public static void main(String[] args)
            throws URISyntaxException, ExecutionException, InterruptedException,
 LineUnavailableException {

        client = TranscribeStreamingAsyncClient.builder()
                .credentialsProvider(getCredentials())
                .region(REGION)
                .build();

        CompletableFuture<Void> result =
 client.startStreamTranscription(getRequest(16_000),
                new AudioStreamPublisher(getStreamFromMic()),
                getResponseHandler());

        result.get();
        client.close();
    }

    private static InputStream getStreamFromMic() throws LineUnavailableException
 {

        // Signed PCM AudioFormat with 16kHz, 16 bit sample size, mono
        int sampleRate = 16000;
        AudioFormat format = new AudioFormat(sampleRate, 16, 1, true, false);
        DataLine.Info info = new DataLine.Info(TargetDataLine.class, format);

        if (!AudioSystem.isLineSupported(info)) {
            System.out.println("Line not supported");
            System.exit(0);
        }

        TargetDataLine line = (TargetDataLine) AudioSystem.getLine(info);
        line.open(format);
```

```
        line.start();

        InputStream audioStream = new AudioInputStream(line);
        return audioStream;
    }

    private static AwsCredentialsProvider getCredentials() {
        return DefaultCredentialsProvider.create();
    }

    private static StartStreamTranscriptionRequest getRequest(Integer
 mediaSampleRateHertz) {
        return StartStreamTranscriptionRequest.builder()
                .languageCode(LanguageCode.EN_US.toString())
                .mediaEncoding(MediaEncoding.PCM)
                .mediaSampleRateHertz(mediaSampleRateHertz)
                .build();
    }

    private static StartStreamTranscriptionResponseHandler getResponseHandler() {
        return StartStreamTranscriptionResponseHandler.builder()
                .onResponse(r -> {
                    System.out.println("Received Initial response");
                })
                .onError(e -> {
                    System.out.println(e.getMessage());
                    StringWriter sw = new StringWriter();
                    e.printStackTrace(new PrintWriter(sw));
                    System.out.println("Error Occurred: " + sw);
                })
                .onComplete(() -> {
                    System.out.println("=== All records stream successfully
 ===");
                })
                .subscriber(event -> {
                    List<Result> results = ((TranscriptEvent)
 event).transcript().results();
                    if (results.size() > 0) {
                        if (!
results.get(0).alternatives().get(0).transcript().isEmpty()) {

 System.out.println(results.get(0).alternatives().get(0).transcript());
                        }
                    }
```

```
                })
                .build();
    }


    private static class AudioStreamPublisher implements Publisher<AudioStream> {
        private static Subscription currentSubscription;
        private final InputStream inputStream;

        private AudioStreamPublisher(InputStream inputStream) {
            this.inputStream = inputStream;
        }

        @Override
        public void subscribe(Subscriber<? super AudioStream> s) {

            if (currentSubscription == null) {
                currentSubscription = new SubscriptionImpl(s, inputStream);
            } else {
                currentSubscription.cancel();
                currentSubscription = new SubscriptionImpl(s, inputStream);
            }
            s.onSubscribe(currentSubscription);
        }
    }

    public static class SubscriptionImpl implements Subscription {
        private static final int CHUNK_SIZE_IN_BYTES = 1024;
        private final Subscriber<? super AudioStream> subscriber;
        private final InputStream inputStream;
        private final ExecutorService executor = Executors.newFixedThreadPool(1);
        private final AtomicLong demand = new AtomicLong(0);

        SubscriptionImpl(Subscriber<? super AudioStream> s, InputStream
inputStream) {
            this.subscriber = s;
            this.inputStream = inputStream;
        }

        @Override
        public void request(long n) {
            if (n <= 0) {
                subscriber.onError(new IllegalArgumentException("Demand must be
positive"));
```

```
            }

            demand.getAndAdd(n);

            executor.submit(() -> {
                try {
                    do {
                        ByteBuffer audioBuffer = getNextEvent();
                        if (audioBuffer.remaining() > 0) {
                            AudioEvent audioEvent =
audioEventFromBuffer(audioBuffer);
                            subscriber.onNext(audioEvent);
                        } else {
                            subscriber.onComplete();
                            break;
                        }
                    } while (demand.decrementAndGet() > 0);
                } catch (Exception e) {
                    subscriber.onError(e);
                }
            });
        }

        @Override
        public void cancel() {
            executor.shutdown();
        }

        private ByteBuffer getNextEvent() {
            ByteBuffer audioBuffer = null;
            byte[] audioBytes = new byte[CHUNK_SIZE_IN_BYTES];

            int len = 0;
            try {
                len = inputStream.read(audioBytes);

                if (len <= 0) {
                    audioBuffer = ByteBuffer.allocate(0);
                } else {
                    audioBuffer = ByteBuffer.wrap(audioBytes, 0, len);
                }
            } catch (IOException e) {
                throw new UncheckedIOException(e);
            }
```

```
            return audioBuffer;
        }

        private AudioEvent audioEventFromBuffer(ByteBuffer bb) {
            return AudioEvent.builder()
                    .audioChunk(SdkBytes.fromByteBuffer(bb))
                    .build();
        }
    }
}
```

- For API details, see the following topics in *AWS SDK for Java 2.x API Reference*.
  - GetTranscriptionJob
  - StartTranscriptionJob

Python

SDK for Python (Boto3)

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
import time
import boto3


def transcribe_file(job_name, file_uri, transcribe_client):
    transcribe_client.start_transcription_job(
        TranscriptionJobName=job_name,
        Media={"MediaFileUri": file_uri},
        MediaFormat="wav",
        LanguageCode="en-US",
    )

    max_tries = 60
    while max_tries > 0:
```

```
        max_tries -= 1
        job =
 transcribe_client.get_transcription_job(TranscriptionJobName=job_name)
        job_status = job["TranscriptionJob"]["TranscriptionJobStatus"]
        if job_status in ["COMPLETED", "FAILED"]:
            print(f"Job {job_name} is {job_status}.")
            if job_status == "COMPLETED":
                print(
                    f"Download the transcript from\n"
                    f"\t{job['TranscriptionJob']['Transcript']
['TranscriptFileUri']}."
                )
            break
        else:
            print(f"Waiting for {job_name}. Current status is {job_status}.")
        time.sleep(10)


def main():
    transcribe_client = boto3.client("transcribe")
    file_uri = "s3://test-transcribe/answer2.wav"
    transcribe_file("Example-job", file_uri, transcribe_client)


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

- For API details, see the following topics in *AWS SDK for Python (Boto3) API Reference*.
  - GetTranscriptionJob
  - StartTranscriptionJob

For a complete list of AWS SDK developer guides and code examples, see Using this service with an AWS SDK. This topic also includes information about getting started and details about previous SDK versions.

Security in Amazon Transcribe

Cloud security at AWS is the highest priority. As an AWS customer, you benefit from a data center and network architecture built to meet the requirements of the most security-sensitive organizations.

Security is a shared responsibility between AWS and you. The describes this as security *of* the cloud and security *in* the cloud:

- **Security of the cloud**: AWS is responsible for protecting the infrastructure that runs AWS services in the AWS Cloud. AWS also provides you with services you can use securely. Third-party auditors regularly test and verify the effectiveness of our security as part of the .
- **Security in the cloud**: Your responsibility is determined by the AWS service that you use. You are also responsible for other factors including the sensitivity of your data, your company's requirements, and applicable laws and regulations.

This documentation helps you understand how to apply the shared responsibility model when using Amazon Transcribe. The following topics show you how to configure Amazon Transcribe to meet your security and compliance objectives. You also learn how to use other AWS services to monitor and secure your Amazon Transcribe resources.

Topics

- Identity and Access Management for Amazon Transcribe
- Data protection in Amazon Transcribe
- Monitoring Amazon Transcribe
- Compliance validation for Amazon Transcribe
- Resilience in Amazon Transcribe
- Infrastructure security in Amazon Transcribe
- Vulnerability analysis and management in Amazon Transcribe
- Security best practices for Amazon Transcribe

Identity and Access Management for Amazon Transcribe

AWS Identity and Access Management (IAM) is an AWS service that helps an administrator securely control access to AWS resources. IAM administrators control who can be *authenticated* (signed in) and *authorized* (have permissions) to use Amazon Transcribe resources. IAM is an AWS service that you can use with no additional charge.

Topics

- Audience
- Authenticating with identities
- Managing access using policies
- How Amazon Transcribe works with IAM
- Cross-service confused deputy prevention
- Amazon Transcribe identity-based policy examples
- Troubleshooting Amazon Transcribe identity and access

Audience

How you use AWS Identity and Access Management (IAM) differs based on your role:

- **Service user** - request permissions from your administrator if you cannot access features (see Troubleshooting Amazon Transcribe identity and access)
- **Service administrator** - determine user access and submit permission requests (see How Amazon Transcribe works with IAM)
- **IAM administrator** - write policies to manage access (see Amazon Transcribe identity-based policy examples)

Authenticating with identities

Authentication is how you sign in to AWS using your identity credentials. You must be authenticated as the AWS account root user, an IAM user, or by assuming an IAM role.

You can sign in as a federated identity using credentials from an identity source like AWS IAM Identity Center (IAM Identity Center), single sign-on authentication, or Google/Facebook

credentials. For more information about signing in, see How to sign in to your AWS account in the *AWS Sign-In User Guide*.

For programmatic access, AWS provides an SDK and CLI to cryptographically sign requests. For more information, see AWS Signature Version 4 for API requests in the *IAM User Guide*.

AWS account root user

When you create an AWS account, you begin with one sign-in identity called the AWS account *root user* that has complete access to all AWS services and resources. We strongly recommend that you don't use the root user for everyday tasks. For tasks that require root user credentials, see Tasks that require root user credentials in the *IAM User Guide*.

Federated identity

As a best practice, require human users to use federation with an identity provider to access AWS services using temporary credentials.

A *federated identity* is a user from your enterprise directory, web identity provider, or Directory Service that accesses AWS services using credentials from an identity source. Federated identities assume roles that provide temporary credentials.

For centralized access management, we recommend AWS IAM Identity Center. For more information, see What is IAM Identity Center? in the *AWS IAM Identity Center User Guide*.

IAM users and groups

An *IAM user* is an identity with specific permissions for a single person or application. We recommend using temporary credentials instead of IAM users with long-term credentials. For more information, see Require human users to use federation with an identity provider to access AWS using temporary credentials in the *IAM User Guide*.

An *IAM group* specifies a collection of IAM users and makes permissions easier to manage for large sets of users. For more information, see Use cases for IAM users in the *IAM User Guide*.

IAM roles

An *IAM role* is an identity with specific permissions that provides temporary credentials. You can assume a role by switching from a user to an IAM role (console) or by calling an AWS CLI or AWS API operation. For more information, see Methods to assume a role in the *IAM User Guide*.

IAM roles are useful for federated user access, temporary IAM user permissions, cross-account access, cross-service access, and applications running on Amazon EC2. For more information, see Cross account resource access in IAM in the *IAM User Guide*.

Managing access using policies

You control access in AWS by creating policies and attaching them to AWS identities or resources. A policy defines permissions when associated with an identity or resource. AWS evaluates these policies when a principal makes a request. Most policies are stored in AWS as JSON documents. For more information about JSON policy documents, see Overview of JSON policies in the *IAM User Guide*.

Using policies, administrators specify who has access to what by defining which **principal** can perform **actions** on what **resources**, and under what **conditions**.

By default, users and roles have no permissions. An IAM administrator creates IAM policies and adds them to roles, which users can then assume. IAM policies define permissions regardless of the method used to perform the operation.

Identity-based policies

Identity-based policies are JSON permissions policy documents that you attach to an identity (user, group, or role). These policies control what actions identities can perform, on which resources, and under what conditions. To learn how to create an identity-based policy, see Define custom IAM permissions with customer managed policies in the *IAM User Guide*.

Identity-based policies can be *inline policies* (embedded directly into a single identity) or *managed policies* (standalone policies attached to multiple identities). To learn how to choose between managed and inline policies, see Choose between managed policies and inline policies in the *IAM User Guide*.

Resource-based policies

Resource-based policies are JSON policy documents that you attach to a resource. Examples include IAM *role trust policies* and Amazon S3 *bucket policies*. In services that support resource-based policies, service administrators can use them to control access to a specific resource. You must specify a principal in a resource-based policy.

Resource-based policies are inline policies that are located in that service. You can't use AWS managed policies from IAM in a resource-based policy.

Other policy types

AWS supports additional policy types that can set the maximum permissions granted by more common policy types:

- **Permissions boundaries** – Set the maximum permissions that an identity-based policy can grant to an IAM entity. For more information, see Permissions boundaries for IAM entities in the *IAM User Guide*.
- **Service control policies (SCPs)** – Specify the maximum permissions for an organization or organizational unit in AWS Organizations. For more information, see Service control policies in the *AWS Organizations User Guide*.
- **Resource control policies (RCPs)** – Set the maximum available permissions for resources in your accounts. For more information, see Resource control policies (RCPs) in the *AWS Organizations User Guide*.
- **Session policies** – Advanced policies passed as a parameter when creating a temporary session for a role or federated user. For more information, see Session policies in the *IAM User Guide*.

Multiple policy types

When multiple types of policies apply to a request, the resulting permissions are more complicated to understand. To learn how AWS determines whether to allow a request when multiple policy types are involved, see Policy evaluation logic in the *IAM User Guide*.

How Amazon Transcribe works with IAM

Before you use IAM to manage access to Amazon Transcribe, learn what IAM features are available to use with Amazon Transcribe.

IAM features you can use with Amazon Transcribe

IAM feature	Amazon Transcribe support
Identity-based policies	Yes
Resource-based policies	No
Policy actions	Yes

IAM feature	Amazon Transcribe support
Policy resources	Yes
Policy condition keys (service-specific)	Yes
ACLs	No
ABAC (tags in policies)	Partial
Temporary credentials	Yes
Principal permissions	Yes
Service roles	Yes
Service-linked roles	No

To get a high-level view of how Amazon Transcribe and other AWS services work with most IAM features, see AWS services that work with IAM in the *IAM User Guide*.

Identity-based policies for Amazon Transcribe

Supports identity-based policies: Yes

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

With IAM identity-based policies, you can specify allowed or denied actions and resources as well as the conditions under which actions are allowed or denied. To learn about all of the elements that you can use in a JSON policy, see IAM JSON policy elements reference in the *IAM User Guide*.

Identity-based policy examples for Amazon Transcribe

To view examples of Amazon Transcribe identity-based policies, see Amazon Transcribe identity-based policy examples.

Resource-based policies within Amazon Transcribe

Supports resource-based policies: No

Resource-based policies are JSON policy documents that you attach to a resource. Examples of resource-based policies are IAM *role trust policies* and Amazon S3 *bucket policies*. In services that support resource-based policies, service administrators can use them to control access to a specific resource. For the resource where the policy is attached, the policy defines what actions a specified principal can perform on that resource and under what conditions. You must specify a principal in a resource-based policy. Principals can include accounts, users, roles, federated users, or AWS services.

To enable cross-account access, you can specify an entire account or IAM entities in another account as the principal in a resource-based policy. For more information, see Cross account resource access in IAM in the *IAM User Guide*.

Policy actions for Amazon Transcribe

Supports policy actions: Yes

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

The `Action` element of a JSON policy describes the actions that you can use to allow or deny access in a policy. Include actions in a policy to grant permissions to perform the associated operation.

To see a list of Amazon Transcribe actions, see Actions defined by Amazon Transcribe in the *Service Authorization Reference*.

Policy actions in Amazon Transcribe use the `transcribe` prefix before the action. To specify multiple actions in a single statement, separate them with commas.

```
"Action": [
      "transcribe:action1",
      "transcribe:action2"
]
```

You can specify multiple actions using wildcards (*). For example, to specify all actions that begin with the word `List`, include the following action:

```
"Action": "transcribe:List*"
```

To view examples of Amazon Transcribe identity-based policies, see Amazon Transcribe identity-based policy examples.

Policy resources for Amazon Transcribe

Supports policy resources: Yes

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

The `Resource` JSON policy element specifies the object or objects to which the action applies. As a best practice, specify a resource using its Amazon Resource Name (ARN). For actions that don't support resource-level permissions, use a wildcard (*) to indicate that the statement applies to all resources.

```
"Resource": "*"
```

To see a list of Amazon Transcribe resource types and their ARNs, see Resources defined by Amazon Transcribe in the *Service Authorization Reference*. To learn with which actions you can specify the ARN of each resource, see Actions defined by Amazon Transcribe.

To view examples of Amazon Transcribe identity-based policies, see Amazon Transcribe identity-based policy examples.

Policy condition keys for Amazon Transcribe

Supports service-specific policy condition keys: Yes

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

The `Condition` element specifies when statements execute based on defined criteria. You can create conditional expressions that use condition operators, such as equals or less than, to match the condition in the policy with values in the request. To see all AWS global condition keys, see AWS global condition context keys in the *IAM User Guide*.

To see a list of Amazon Transcribe condition keys, see Condition keys for Amazon Transcribe in the *Service Authorization Reference*. To learn with which actions and resources you can use a condition key, see Actions defined by Amazon Transcribe.

To view examples of Amazon Transcribe identity-based policies, see Amazon Transcribe identity-based policy examples.

ACLs in Amazon Transcribe

Supports ACLs: No

Access control lists (ACLs) control which principals (account members, users, or roles) have permissions to access a resource. ACLs are similar to resource-based policies, although they do not use the JSON policy document format.

ABAC with Amazon Transcribe

Supports ABAC (tags in policies): Partial

Attribute-based access control (ABAC) is an authorization strategy that defines permissions based on attributes called tags. You can attach tags to IAM entities and AWS resources, then design ABAC policies to allow operations when the principal's tag matches the tag on the resource.

To control access based on tags, you provide tag information in the condition element of a policy using the `aws:ResourceTag/key-name`, `aws:RequestTag/key-name`, or `aws:TagKeys` condition keys.

If a service supports all three condition keys for every resource type, then the value is **Yes** for the service. If a service supports all three condition keys for only some resource types, then the value is **Partial**.

For more information about ABAC, see Define permissions with ABAC authorization in the *IAM User Guide*. To view a tutorial with steps for setting up ABAC, see Use attribute-based access control (ABAC) in the *IAM User Guide*.

For more information about tagging Amazon Transcribe resources, see Tagging resources. For more detailed information on tag-based access control, see Controlling access to AWS resources using tags.

Using temporary credentials with Amazon Transcribe

Supports temporary credentials: Yes

Temporary credentials provide short-term access to AWS resources and are automatically created when you use federation or switch roles. AWS recommends that you dynamically generate temporary credentials instead of using long-term access keys. For more information, see Temporary security credentials in IAM and AWS services that work with IAM in the *IAM User Guide*.

Cross-service principal permissions for Amazon Transcribe

Supports forward access sessions (FAS): Yes

Forward access sessions (FAS) use the permissions of the principal calling an AWS service, combined with the requesting AWS service to make requests to downstream services. For policy details when making FAS requests, see Forward access sessions.

Service roles for Amazon Transcribe

Supports service roles: Yes

A service role is an IAM role that a service assumes to perform actions on your behalf. An IAM administrator can create, modify, and delete a service role from within IAM. For more information, see Create a role to delegate permissions to an AWS service in the *IAM User Guide*.

> **Warning**
>
> Changing the permissions for a service role might break Amazon Transcribe functionality. Edit service roles only when Amazon Transcribe provides guidance to do so.

Service-linked roles for Amazon Transcribe

Supports service-linked roles: No

A service-linked role is a type of service role that is linked to an AWS service. The service can assume the role to perform an action on your behalf. Service-linked roles appear in your AWS account and are owned by the service. An IAM administrator can view, but not edit the permissions for service-linked roles.

Amazon Transcribe doesn't support service-linked roles.

For details about creating or managing service-linked roles for other services, see AWS services that work with IAM. Find a service in the table that includes a `Yes` in the **Service-linked role** column. Choose the **Yes** link to view the service-linked role documentation for that service.

Cross-service confused deputy prevention

A confused deputy is an entity (a service or an account) that is coerced by a different entity to perform an action. This type of impersonation can happen cross-account and cross-service.

To prevent confused deputies, AWS provides tools that help you protect your data for all services using service principals that have been given access to resources in your AWS account. This section focuses on cross-service confused deputy prevention specific to Amazon Transcribe; however, you can learn more about this topic in the confused deputy problem section of the *IAM User Guide*.

To limit the permissions IAM gives to Amazon Transcribe to access your resources, we recommend using the global condition context keys `aws:SourceArn` and `aws:SourceAccount` in your resource policies.

If you use both of these global condition context keys, and the `aws:SourceArn` value contains the AWS account ID, the `aws:SourceAccount` value and the AWS account in `aws:SourceArn` must use the same AWS account ID when used in the same policy statement.

If you want only one resource to be associated with the cross-service access, use `aws:SourceArn`. If you want to associate any resource in that AWS account with cross-service access, use `aws:SourceAccount`.

> **Note**
>
> The most effective way to protect against the confused deputy problem is to use the `aws:SourceArn` global condition context key with the **full ARN** of the resource. If you don't know the full ARN, or if you're specifying multiple resources, use the `aws:SourceArn` global context condition key with wildcards (*) for the unknown portions of the ARN. For example, `arn:aws:transcribe::123456789012:*`.

For an example of an assume role policy that shows how you can prevent a confused deputy issue, see Confused deputy prevention policy.

Amazon Transcribe identity-based policy examples

By default, users and roles don't have permission to create or modify Amazon Transcribe resources. To grant users permission to perform actions on the resources that they need, an IAM administrator can create IAM policies.

To learn how to create an IAM identity-based policy by using these example JSON policy documents, see Create IAM policies (console) in the *IAM User Guide*.

For details about actions and resource types defined by Amazon Transcribe, including the format of the ARNs for each of the resource types, see Actions, resources, and condition keys for Amazon Transcribe in the *Service Authorization Reference*.

Topics

- Policy best practices
- Using the AWS Management Console
- Permissions required for IAM roles
- Permissions required for Amazon S3 encryption keys
- Permissions required for customer-managed key encryption
- Allow users to view their own permissions
- AWS KMS encryption context policy
- Confused deputy prevention policy
- Viewing transcription jobs based on tags

Policy best practices

Identity-based policies determine whether someone can create, access, or delete Amazon Transcribe resources in your account. These actions can incur costs for your AWS account. When you create or edit identity-based policies, follow these guidelines and recommendations:

- **Get started with AWS managed policies and move toward least-privilege permissions** – To get started granting permissions to your users and workloads, use the *AWS managed policies* that grant permissions for many common use cases. They are available in your AWS account. We recommend that you reduce permissions further by defining AWS customer managed policies that are specific to your use cases. For more information, see AWS managed policies or AWS managed policies for job functions in the *IAM User Guide*.
- **Apply least-privilege permissions** – When you set permissions with IAM policies, grant only the permissions required to perform a task. You do this by defining the actions that can be taken on specific resources under specific conditions, also known as *least-privilege permissions*. For more information about using IAM to apply permissions, see Policies and permissions in IAM in the *IAM User Guide*.

- **Use conditions in IAM policies to further restrict access** – You can add a condition to your policies to limit access to actions and resources. For example, you can write a policy condition to specify that all requests must be sent using SSL. You can also use conditions to grant access to service actions if they are used through a specific AWS service, such as CloudFormation. For more information, see IAM JSON policy elements: Condition in the *IAM User Guide*.
- **Use IAM Access Analyzer to validate your IAM policies to ensure secure and functional permissions** – IAM Access Analyzer validates new and existing policies so that the policies adhere to the IAM policy language (JSON) and IAM best practices. IAM Access Analyzer provides more than 100 policy checks and actionable recommendations to help you author secure and functional policies. For more information, see Validate policies with IAM Access Analyzer in the *IAM User Guide*.
- **Require multi-factor authentication (MFA)** – If you have a scenario that requires IAM users or a root user in your AWS account, turn on MFA for additional security. To require MFA when API operations are called, add MFA conditions to your policies. For more information, see Secure API access with MFA in the *IAM User Guide*.

For more information about best practices in IAM, see Security best practices in IAM in the *IAM User Guide*.

Using the AWS Management Console

To access the Amazon Transcribe console, you must have a minimum set of permissions. These permissions must allow you to list and view details about the Amazon Transcribe resources in your AWS account. If you create an identity-based policy that is more restrictive than the minimum required permissions, the console won't function as intended for entities (users or roles) with that policy.

You don't need to allow minimum console permissions for users that are making calls only to the AWS CLI or the AWS API. Instead, allow access to only the actions that match the API operation that they're trying to perform.

To ensure that an entity (users and roles) can use the AWS Management Console, attach one of the following AWS-managed policies to them.

- `AmazonTranscribeFullAccess`: Grants full access to create, read, update, delete, and run all Amazon Transcribe resources. It also allows access to Amazon S3 buckets with `transcribe` in the bucket name.

- `AmazonTranscribeReadOnlyAccess`: Grants read-only access to Amazon Transcribe resources so that you can get and list transcription jobs and custom vocabularies.

> **Note**
>
> You can review the managed permission policies by signing in to the IAM AWS Management Console and searching by policy name. A search for "transcribe" returns both policies listed above (*AmazonTranscribeReadOnly* and *AmazonTranscribeFullAccess*).

You can also create your own custom IAM policies to allow permissions for Amazon Transcribe API actions. You can attach these custom policies to the entities that require those permissions.

Permissions required for IAM roles

If you create an IAM role to call Amazon Transcribe, it must have permission to access the Amazon S3 bucket. If applicable, the KMS key must also be used to encrypt the contents of the bucket. Refer to the following sections for example policies.

Trust policies

The IAM entity you use to make your transcription request must have a trust policy that enables Amazon Transcribe to assume that role. Use the following Amazon Transcribe trust policy. Note that if you're making a real-time Call Analytics request with post-call analytics enabled, you must use 'Trust policy for real-time Call Analytics'.

Trust policy for Amazon Transcribe

JSON

```
{
  "Version":"2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "Service": [
          "transcribe.amazonaws.com"
        ]
```

```
      },
      "Action": [
        "sts:AssumeRole"
      ],
      "Condition": {
        "StringEquals": {
          "aws:SourceAccount": "111122223333"
        },
        "ArnLike": {
          "aws:SourceArn": "arn:aws-eusc:transcribe:us-west-2:111122223333:*"
        }
      }
    }
  ]
}
```

Trust policy for real-time Call Analytics

JSON

```
{
  "Version":"2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "Service": [
          "transcribe.streaming.amazonaws.com"
        ]
      },
      "Action": [
        "sts:AssumeRole"
      ],
      "Condition": {
        "StringEquals": {
          "aws:SourceAccount": "111122223333"
        },
        "ArnLike": {
          "aws:SourceArn": "arn:aws-eusc:transcribe:us-west-2:111122223333:*"
        }
      }
    }
```

```
  ]
}
```

Amazon S3 input bucket policy

The following policy gives an IAM role permission to access files from the specified input bucket.

JSON

```
{
    "Version":"2012-10-17",
    "Statement": {
        "Effect": "Allow",
        "Action": [
            "s3:GetObject",
            "s3:ListBucket"
        ],
        "Resource": [
            "arn:aws-eusc:s3:::DOC-EXAMPLE-INPUT-BUCKET",
            "arn:aws-eusc:s3:::DOC-EXAMPLE-INPUT-BUCKET/*"
        ]
    }
}
```

Amazon S3 output bucket policy

The following policy gives an IAM role permission to write files to the specified output bucket.

JSON

```
{
    "Version":"2012-10-17",
    "Statement": {
        "Effect": "Allow",
        "Action": [
            "s3:PutObject"
        ],
        "Resource": [
            "arn:aws-eusc:s3:::DOC-EXAMPLE-OUTPUT-BUCKET/*"
        ]
```

```
    }
}
```

Permissions required for Amazon S3 encryption keys

If you're using a KMS key to encrypt an Amazon S3 bucket, include the following in the KMS key policy. This gives Amazon Transcribe access to the contents of the bucket. For more information about allowing access to KMS keys, see Allowing external AWS accounts to access an KMS key in the *AWS KMS Developer Guide*.

JSON

```
{
  "Version":"2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "AWS": "arn:aws-eusc:iam::111122223333:role/ExampleRole"
      },
      "Action": [
        "kms:Decrypt"
      ],
      "Resource": "arn:aws-eusc:kms:us-west-2:111122223333:key/KMS-Example-KeyId"
    }
  ]
}
```

Permissions required for customer-managed key encryption

If you use a customer-managed KMS key to encrypt your custom vocabularies, custom vocabulary filters, or custom language models, both the caller making the API request and the IAM role you specify in `DataAccessRoleArn` need access to the key. Both must have the following AWS KMS permissions on the key, and your key policy must grant permissions to both. For more information about encrypting resource artifacts with a customer-managed key, see Encrypting resource artifacts with a customer-managed key.

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "kms:Encrypt",
                "kms:Decrypt",
                "kms:DescribeKey",
                "kms:GenerateDataKey*"
            ],
            "Resource": "arn:aws-eusc:kms:us-west-2:111122223333:key/KMS-Example-KeyId"
        }
    ]
}
```

You must also add a key policy to your customer-managed key that grants the data access role permission to use the key. The following is an example key policy.

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Sid": "Allow use of the key",
            "Effect": "Allow",
            "Principal": {
                "AWS": "arn:aws-eusc:iam::111122223333:role/ExampleDataAccessRole"
            },
            "Action": [
                "kms:Encrypt",
                "kms:Decrypt",
                "kms:GenerateDataKey*",
                "kms:DescribeKey"
            ],
```

```
            "Resource": "*"
        }
    ]
}
```

The role you specify in `DataAccessRoleArn` must also have a trust policy that allows Amazon Transcribe to assume it. Use `transcribe.amazonaws.com` for batch transcription. If your streaming request uses a customer-managed key to encrypt a custom vocabulary, custom vocabulary filter, or custom language model, the role must also trust `transcribe.streaming.amazonaws.com`. The following is an example trust policy that allows both.

JSON

```
{
  "Version":"2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "Service": [
          "transcribe.amazonaws.com",
          "transcribe.streaming.amazonaws.com"
        ]
      },
      "Action": [
        "sts:AssumeRole"
      ],
      "Condition": {
        "StringEquals": {
          "aws:SourceAccount": "111122223333"
        },
        "ArnLike": {
          "aws:SourceArn": "arn:aws-eusc:transcribe:us-west-2:111122223333:*"
        }
      }
    }
  ]
}
```

Allow users to view their own permissions

This example shows how you might create a policy that allows IAM users to view the inline and managed policies that are attached to their user identity. This policy includes permissions to complete this action on the console or programmatically using the AWS CLI or AWS API.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "ViewOwnUserInfo",
            "Effect": "Allow",
            "Action": [
                "iam:GetUserPolicy",
                "iam:ListGroupsForUser",
                "iam:ListAttachedUserPolicies",
                "iam:ListUserPolicies",
                "iam:GetUser"
            ],
            "Resource": ["arn:aws-eusc:iam::*:user/${aws:username}"]
        },
        {
            "Sid": "NavigateInConsole",
            "Effect": "Allow",
            "Action": [
                "iam:GetGroupPolicy",
                "iam:GetPolicyVersion",
                "iam:GetPolicy",
                "iam:ListAttachedGroupPolicies",
                "iam:ListGroupPolicies",
                "iam:ListPolicyVersions",
                "iam:ListPolicies",
                "iam:ListUsers"
            ],
            "Resource": "*"
        }
    ]
}
```

AWS KMS encryption context policy

The following policy grants the IAM role "`ExampleRole`" permission to use the AWS KMS *Decrypt* and *Encrypt* operations for this particular KMS key. This policy works **only** for requests with at least one encryption context pair, in this case "`color:indigoBlue`". For more information on AWS KMS encryption context, see AWS KMS encryption context.

JSON

```
{
  "Version":"2012-10-17",
  "Statement": [
      {
          "Effect": "Allow",
          "Principal": {
              "AWS": "arn:aws-eusc:iam::111122223333:role/ExampleRole"
          },
          "Action": [
              "kms:Decrypt",
              "kms:Encrypt",
              "kms:GenerateDataKey*",
              "kms:ReEncrypt*"
          ],
          "Resource": "*",
          "Condition": {
              "StringEquals": {
                  "kms:EncryptionContext:color": "indigoBlue"
              }
           }
      },
      {
          "Effect": "Allow",
          "Principal": {
              "AWS": "arn:aws-eusc:iam::111122223333:role/ExampleRole"
          },
          "Action": "kms:DescribeKey",
          "Resource": "*"
      }
   ]
}
```

Confused deputy prevention policy

Here's an example of an assume role policy that shows how you can use `aws:SourceArn` and `aws:SourceAccount` with Amazon Transcribe to prevent a confused deputy issue. For more information on confused deputy prevention, see Cross-service confused deputy prevention.

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Principal": {
                "Service": "transcribe.amazonaws.com"
            },
            "Action": [
                "sts:AssumeRole"
            ],
            "Condition": {
                "StringEquals": {
                    "aws:SourceAccount": "111122223333"
                },
                "ArnLike": {
                    "aws:SourceArn": "arn:aws-eusc:transcribe:us-
west-2:111122223333:*"
                }
            }
        }
    ]
}
```

Viewing transcription jobs based on tags

You can use conditions in your identity-based policy to control access to Amazon Transcribe resources based on tags. This example shows how you might create a policy that allows viewing a transcription job. However, permission is granted only if the transcription job tag `Owner` has the value of that user's user name. This policy also grants the permissions necessary to complete this action using the AWS Management Console.

You can attach this policy to the IAM entities in your account. If a role named `test-role` attempts to view a transcription job, the transcription job must be tagged `Owner=test-role` or `owner=test-role` (condition key names are not case-sensitive), otherwise they are denied access. For more information, see IAM JSON policy elements: Condition in the *IAM User Guide*.

For more information on tagging in Amazon Transcribe, see Tagging resources.

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Sid": "ListTranscriptionJobsInConsole",
            "Effect": "Allow",
            "Action": "transcribe:ListTranscriptionJobs",
            "Resource": "*"
        },
        {
            "Sid": "ViewTranscriptionJobsIfOwner",
            "Effect": "Allow",
            "Action": "transcribe:GetTranscriptionJob",
            "Resource": "arn:aws-eusc:transcribe:us-
west-2:111122223333:transcription-job/*",
            "Condition": {
                "StringEquals": {"aws:ResourceTag/Owner": "${aws:username}"}
            }
        }
    ]
}
```

Troubleshooting Amazon Transcribe identity and access

Use the following information to diagnose and fix common issues that you might encounter when working with Amazon Transcribe and AWS Identity and Access Management (IAM).

Topics

- I am not authorized to perform an action in Amazon Transcribe
- I am not authorized to perform iam:PassRole

- I want to allow people outside of my AWS account to access my Amazon Transcribe resources

I am not authorized to perform an action in Amazon Transcribe

If you receive an error that you're not authorized to perform an action, your policies must be updated to allow you to perform the action.

The following example error occurs when the `mateojackson` IAM user tries to use the console to view details about a fictional *`my-example-widget`* resource but doesn't have the fictional `transcribe:`*`GetWidget`* permissions.

```
User: arn:aws-eusc:iam::123456789012:user/mateojackson is not authorized to perform:
 transcribe:GetWidget on resource: my-example-widget
```

In this case, the policy for the `mateojackson` user must be updated to allow access to the *`my-example-widget`* resource by using the `transcribe:`*`GetWidget`* action.

If you need help, contact your AWS administrator. Your administrator is the person who provided you with your sign-in credentials.

I am not authorized to perform iam:PassRole

If you receive an error that you're not authorized to perform the `iam:PassRole` action, your policies must be updated to allow you to pass a role to Amazon Transcribe.

Some AWS services allow you to pass an existing role to that service instead of creating a new service role or service-linked role. To do this, you must have permissions to pass the role to the service.

The following example error occurs when an IAM user named `marymajor` tries to use the console to perform an action in Amazon Transcribe. However, the action requires the service to have permissions that are granted by a service role. Mary does not have permissions to pass the role to the service.

```
User: arn:aws-eusc:iam::123456789012:user/marymajor is not authorized to perform:
 iam:PassRole
```

In this case, Mary's policies must be updated to allow her to perform the `iam:PassRole` action.

If you need help, contact your AWS administrator. Your administrator is the person who provided you with your sign-in credentials.

I want to allow people outside of my AWS account to access my Amazon Transcribe resources

You can create a role that users in other accounts or people outside of your organization can use to access your resources. You can specify who is trusted to assume the role. For services that support resource-based policies or access control lists (ACLs), you can use those policies to grant people access to your resources.

To learn more, consult the following:

- To learn whether Amazon Transcribe supports these features, see How Amazon Transcribe works with IAM.
- To learn how to provide access to your resources across AWS accounts that you own, see Providing access to an IAM user in another AWS account that you own in the *IAM User Guide*.
- To learn how to provide access to your resources to third-party AWS accounts, see Providing access to AWS accounts owned by third parties in the *IAM User Guide*.
- To learn how to provide access through identity federation, see Providing access to externally authenticated users (identity federation) in the *IAM User Guide*.
- To learn the difference between using roles and resource-based policies for cross-account access, see Cross account resource access in IAM in the *IAM User Guide*.

Data protection in Amazon Transcribe

The AWS shared responsibility model applies to data protection in Amazon Transcribe. As described in this model, AWS is responsible for protecting the global infrastructure that runs all of the AWS Cloud. You are responsible for maintaining control over your content that is hosted on this infrastructure. You are also responsible for the security configuration and management tasks for the AWS services that you use. For more information about data privacy, see Data Privacy FAQ. For information about data protection in Europe, see the General Data Protection Regulation (GDPR) Center.

For data protection purposes, we recommend that you protect AWS account credentials and set up individual users with AWS IAM Identity Center or AWS Identity and Access Management (IAM).

That way, each user is given only the permissions necessary to fulfill their job duties. We also recommend that you secure your data in the following ways:

- Use multi-factor authentication (MFA) with each account.
- Use SSL/TLS to communicate with AWS resources. We require TLS 1.2 and recommend TLS 1.3.
- Set up API and user activity logging with AWS CloudTrail. For information about using CloudTrail trails to capture AWS activities, see Working with CloudTrail trails in the *AWS CloudTrail User Guide*.
- Use AWS encryption solutions, along with all default security controls within AWS services.
- Use advanced managed security services such as Amazon Macie, which assists in discovering and securing sensitive data that is stored in Amazon S3.
- If you require FIPS 140-3 validated cryptographic modules when accessing AWS through a command line interface or an API, use a FIPS endpoint. For more information about the available FIPS endpoints, see Federal Information Processing Standard (FIPS) 140-3.

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

Inter-network traffic privacy

An Amazon Virtual Private Cloud (Amazon VPC) endpoint for Amazon Transcribe is a logical entity within a VPC that allows connectivity only to Amazon Transcribe. Amazon VPC routes requests to Amazon Transcribe and routes responses back to the VPC. For more information, see AWS PrivateLink concepts. For information about using Amazon VPC endpoints with Amazon Transcribe see Amazon Transcribe and interface VPC endpoints (AWS PrivateLink).

Data encryption

Data encryption refers to protecting data while in transit and at rest. You can protect your data by using Amazon S3-managed keys or KMS keys at rest, alongside standard Transport Layer Security (TLS) while in transit.

Encryption at rest

Amazon Transcribe uses the default Amazon S3 key (SSE-S3) for server-side encryption of transcripts placed in your Amazon S3 bucket.

When you use the `StartTranscriptionJob` operation, you can specify your own KMS key to encrypt the output from a transcription job.

When you create or update custom vocabularies, custom vocabulary filters, or custom language models, you can specify a customer-managed KMS key to encrypt the associated resource artifacts at rest. If you do not provide a KMS key, Amazon Transcribe encrypts these artifacts using an AWS-owned key. For more information about encrypting resource artifacts with a customer-managed key, see Encrypting resource artifacts with a customer-managed key.

Amazon Transcribe uses an Amazon EBS volume encrypted with the default key.

Encryption in transit

Amazon Transcribe uses TLS 1.2 with AWS certificates to encrypt data in transit. This includes streaming transcriptions.

Key management

Amazon Transcribe works with KMS keys to provide enhanced encryption for your data. With Amazon S3, you can encrypt your input media when creating a transcription job. Integration with AWS KMS allows encryption of the output from a `StartTranscriptionJob` request.

If you don't specify a KMS key, the output of the transcription job is encrypted with the default Amazon S3 key (SSE-S3).

For more information on AWS KMS, see the *AWS Key Management Service Developer Guide*.

Key management using the AWS Management Console

To encrypt the output of your transcription job, you can choose between using a KMS key for the AWS account that is making the request, or a KMS key from another AWS account.

If you don't specify a KMS key, the output of the transcription job is encrypted with the default Amazon S3 key (SSE-S3).

To enable output encryption:

1. Under **Output data** choose **Encryption**.



2. Choose whether the KMS key is from the AWS account you're currently using or from a different AWS account. If you want to use a key from the current AWS account, choose the key from **KMS key ID**. If you're using a key from a different AWS account, you must enter the key's ARN. To use a key from a different AWS account, the caller must have `kms:Encrypt` permissions for the KMS key. Refer to Creating a key policy for more information.

Key management using the API

To use output encryption with the API, you must specify your KMS key using the `OutputEncryptionKMSKeyId` parameter of the `StartCallAnalyticsJob`, `StartMedicalTranscriptionJob`, or `StartTranscriptionJob` operation.

Specify your KMS key using the Amazon Resource Name (ARN) of the key. KMS key ARNs have the format `arn:`*`partition`*`:kms:`*`region`*`:`*`account-ID`*`:key/`*`key-id`*. For example, `arn:aws:kms:us-west-2:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab`.

Note that the entity making the request must have permission to use the specified KMS key.

Encrypting resource artifacts with a customer-managed key

For additional control over encryption, you can use a customer-managed KMS key to encrypt the artifacts associated with your custom vocabularies, custom vocabulary filters, and custom language models.

> **Warning**
>
> Do not disable or delete a customer-managed KMS key, or revoke the permissions granted to the data access role, while your resources are in use. If Amazon Transcribe cannot access the key during a transcription job, the job fails. If key access is lost during a resource update, the resource can become unusable. To recover, restore access to the

key and call the corresponding Update operation (for example, `UpdateVocabulary`, `UpdateVocabularyFilter`, or `UpdateLanguageModel`).

To configure encryption, include the `EncryptionConfiguration` parameter when calling the following operations:

- `CreateVocabulary`, `UpdateVocabulary`
- `CreateVocabularyFilter`, `UpdateVocabularyFilter`
- `CreateLanguageModel`, `UpdateLanguageModel`

The `EncryptionConfiguration` object contains the following fields:

- `KMSKey` (required) – The Amazon Resource Name (ARN) of the KMS key to use for encryption. KMS key ARNs have the format `arn:partition:kms:region:account-ID:key/key-id`. For example, `arn:aws:kms:us-west-2:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab`.
- `KMSEncryptionContext` (optional) – A map of key:value pairs that provide an additional layer of security. For more information, see AWS KMS encryption context.

You must also provide a `DataAccessRoleArn` that references an IAM role with permissions to use the specified KMS key. For more information about the required AWS KMS permissions and an example policy, see Permissions required for customer-managed key encryption.

Note

You can change the KMS key used to encrypt a resource by calling the corresponding Update operation with a new `EncryptionConfiguration`. If you omit `EncryptionConfiguration` from an Update request, the resource artifacts are encrypted with an AWS-owned key.

Note

When a custom vocabulary or custom vocabulary filter is encrypted with a customer-managed key, the download URI returned by `GetVocabulary` or `GetVocabularyFilter`

points to encrypted data. To decrypt this data, use the AWS Encryption SDK with the same KMS key that was used to encrypt the resource. Direct AWS KMS decryption is not supported. If no customer-managed key was specified, these URIs return unencrypted data.

AWS KMS encryption context

AWS KMS encryption context is a map of plain text, non-secret key:value pairs. This map represents additional authenticated data, known as encryption context pairs, which provide an added layer of security for your data. Amazon Transcribe requires a symmetric encryption key to encrypt transcription output into a customer-specified Amazon S3 bucket. To learn more, see Asymmetric keys in AWS KMS.

When creating your encryption context pairs, **do not** include sensitive information. Encryption context is not secret—it's visible in plain text within your CloudTrail logs (so you can use it to identify and categorize your cryptographic operations).

Your encryption context pair can include special characters, such as underscores (_), dashes (-), slashes (/, \) and colons (:).

Tip

It can be useful to relate the values in your encryption context pair to the data being encrypted. Although not required, we recommend you use non-sensitive metadata related to your encrypted content, such as file names, header values, or unencrypted database fields.

To use output encryption with the API, set the `KMSEncryptionContext` parameter in the `StartTranscriptionJob` operation. In order to provide encryption context for the output encryption operation, the `OutputEncryptionKMSKeyId` parameter must reference a symmetric KMS key ID.

You can also specify encryption context when encrypting resource artifacts (custom vocabularies, custom vocabulary filters, and custom language models) by including the `KMSEncryptionContext` field in the `EncryptionConfiguration` parameter. For more information about encrypting resource artifacts with a customer-managed key, see Encrypting resource artifacts with a customer-managed key.

You can use AWS KMS condition keys with IAM policies to control access to a symmetric encryption KMS key based on the encryption context that was used in the request for a cryptographic operation. For an example encryption context policy, see AWS KMS encryption context policy.

Using encryption context is optional, but recommended. For more information, see Encryption context.

Opting out of using your data for service improvement

By default, Amazon Transcribe stores and uses voice inputs that it has processed to develop the service and continuously improve your experience. You can opt out of having your content used to develop and improve Amazon Transcribe by using an AWS Organizations opt-out policy. For information about how to opt out, see AI services opt-out policies.

Monitoring Amazon Transcribe

Monitoring is an important part of maintaining the reliability, availability, and performance of Amazon Transcribe and your other AWS solutions. AWS provides the following monitoring tools to watch Amazon Transcribe, report when something is wrong, and take automatic actions when appropriate:

- **Amazon CloudWatch** monitors your AWS resources and the applications that you run on AWS in real time. You can collect and track metrics, create customized dashboards, and set alarms that notify you or take actions when a specified metric reaches a threshold that you specify. For example, you can have CloudWatch track CPU usage or other metrics on your Amazon EC2 instances and automatically launch new instances when needed.
- **Amazon CloudWatch Logs** can monitor, store, and access your log files from Amazon EC2 instances, CloudTrail, and other sources. CloudWatch Logs can monitor information in the log files and notify you when certain thresholds are met. You can also archive your log data in highly durable storage.
- **AWS CloudTrail** captures API calls and related events made by or on behalf of your AWS account and delivers the log files to an Amazon S3 bucket that you specify. You can identify which users and accounts called AWS, the source IP address from which the calls were made, and when the calls occurred.

For more information, see the *Amazon CloudWatch User Guide*.

Amazon EventBridge is a serverless service that uses events to connect application components together, making it easier for you to build scalable event-driven applications. EventBridge delivers a stream of real-time data from your own applications, Software as a Service (SaaS) applications, and AWS services and routes that data to targets such as Lambda. You can monitor events that happen in services, and build event-driven architectures. For more information, see the *Amazon EventBridge User Guide*.

Topics

- Monitoring Amazon Transcribe with Amazon CloudWatch
- Monitoring Amazon Transcribe with AWS CloudTrail
- Using Amazon EventBridge with Amazon Transcribe

Monitoring Amazon Transcribe with Amazon CloudWatch

You can monitor Amazon Transcribe using CloudWatch, which collects raw data and processes it into readable, near real-time metrics. These statistics are kept for 15 months, so that you can access historical information and gain a better perspective on how your web application or service is performing. You can also set alarms that watch for certain thresholds, and send notifications or take actions when those thresholds are met. For more information, see the *CloudWatch User Guide*.

Using Amazon CloudWatch metrics and dimensions with Amazon Transcribe

Amazon Transcribe supports CloudWatch metrics and dimensions, which are data that can help you monitor performance. Supported metrics categories include traffic, errors, data transfer, and latency associated with your transcription jobs. Supported metrics are located through CloudWatch in the **AWS/Transcribe** namespace.

> **Note**
>
> CloudWatch monitoring metrics are free of charge and don't count against CloudWatch service quotas.

For more information on CloudWatch metrics, see Using Amazon CloudWatch metrics.

Monitoring Amazon Transcribe with AWS CloudTrail

Amazon Transcribe is integrated with AWS CloudTrail, a service that provides a record of actions taken in Amazon Transcribe by an AWS Identity and Access Management (IAM) user or role, or by an AWS service. CloudTrail captures all API calls for Amazon Transcribe. That includes calls from the AWS Management Console and code calls to the Amazon Transcribe APIs, as events. By creating a trail, you can enable continuous delivery of CloudTrail events, including events for Amazon Transcribe, to an Amazon S3 bucket. If you don't create a trail, you can still view the most recent events in the CloudTrail AWS Management Console in **Event history**. Using the information collected by CloudTrail, you can see each request that is made to Amazon Transcribe, the IP address from which the request is made, who made the request, when it is made, and additional details.

To learn more about CloudTrail, refer to the *AWS CloudTrail User Guide*.

Amazon Transcribe and CloudTrail

CloudTrail is enabled on your AWS account when you create the account. When activity occurs in Amazon Transcribe, that activity is recorded in a CloudTrail event along with other AWS service events in the CloudTrail **Event history**. You can view, search, and download recent events in your AWS account. For more information, see Viewing Events with CloudTrail Event History.

To get an ongoing record of events in your AWS account, including events for Amazon Transcribe, create a trail. A *trail* is a configuration that enables CloudTrail to deliver events as log files to a specified Amazon S3 bucket. CloudTrail log files contain one or more log entries. An *event* represents a single request from any source. It includes information about the requested action, the date and time of the action, request parameters, and so on. CloudTrail log files aren't an ordered stack trace of the public API calls, so they don't appear in any specific order.

By default, when you create a trail in the AWS Management Console, the trail applies to all AWS Regions. The trail logs events from all AWS Regions in the AWS partition and delivers the log files to the Amazon S3 bucket that you specify. Additionally, you can configure other AWS services to further analyze and act upon the event data collected in CloudTrail logs. For more information, see:

- Overview for Creating a Trail
- CloudTrail Supported Services and Integrations
- Configuring Amazon SNS Notifications for CloudTrail
- Receiving CloudTrail Log Files from Multiple Regions and Receiving CloudTrail Log Files from Multiple Accounts

CloudTrail logs all Amazon Transcribe actions, which are documented in the API Reference. For example, the `CreateVocabulary`, `GetTranscriptionJob`, and `StartTranscriptionJob` operations generate entries in the CloudTrail log files. When CloudTrail logs Amazon Transcribe API operations, the CloudTrail log entry uses empty strings for sensitive information in the request and response parameters, such as Amazon S3 URI values.

Every event or log entry contains information about who generated the request. This information helps you determine the following:

- Whether the request is made with root or IAM user credentials
- Whether the request is made with temporary security credentials for an IAM role or federated user
- Whether the request is made by another AWS service

For more information, see the CloudTrail userIdentity Element.

You can also aggregate Amazon Transcribe log files from multiple AWS Regions and multiple AWS accounts into a single Amazon S3 bucket. For more information, see Receiving CloudTrail Log Files from Multiple Regions and Receiving CloudTrail Log Files from Multiple Accounts.

Example: Amazon Transcribe log file entries

A *trail* is a configuration that enables delivery of events as log files to a specified Amazon S3 bucket. CloudTrail log files contain one or more log entries. An *event* represents a single request from any source. It includes such information about the requested action as the date and time of the action, and request parameters. CloudTrail log files aren't an ordered stack trace of the public API calls, so they don't appear in any specific order.

Calls to the `StartTranscriptionJob` and `GetTranscriptionJob` API operations create the following entry.

> **Note**
>
> When CloudTrail logs Amazon Transcribe API operations, the CloudTrail log entry uses empty strings for sensitive information in the request and response parameters, such as Amazon S3 URI values.

```
{
```

```
    "Records": [
        {
            "eventVersion": "1.05",
            "userIdentity": {
                "type": "IAMUser",
                "principalId": "111122223333",
                "arn": "arn:aws:iam:us-west-2:111122223333:user/my-user-name",
                "accountId": "111122223333",
                "accessKeyId": "AKIAIOSFODNN7EXAMPLE",
                "userName": "my-user-name"
            },
            "eventTime": "2022-03-07T15:03:45Z",
            "eventSource": "transcribe.amazonaws.com",
            "eventName": "StartTranscriptionJob",
            "awsRegion": "us-west-2",
            "sourceIPAddress": "127.0.0.1",
            "userAgent": "[]",
            "requestParameters": {
                "mediaFormat": "flac",
                "languageCode": "en-US",
                "transcriptionJobName": "my-first-transcription-job",
                "media": {
                    "mediaFileUri": ""
                }
            },
            "responseElements": {
                "transcriptionJob": {
                    "transcriptionJobStatus": "IN_PROGRESS",
                    "mediaFormat": "flac",
                    "creationTime": "2022-03-07T15:03:44.229000-08:00",
                    "transcriptionJobName": "my-first-transcription-job",
                    "languageCode": "en-US",
                    "media": {
                        "mediaFileUri": ""
                    }
                }
            },
            "requestID": "47B8E8D397DCE7A6",
            "eventID": "cdc4b7ed-e171-4cef-975a-ad829d4123e8",
            "eventType": "AwsApiCall",
            "recipientAccountId": "111122223333"
        },
        {
            "eventVersion": "1.05",
```

```
            "userIdentity": {
                "type": "IAMUser",
                "principalId": "111122223333",
                "arn": "arn:aws:iam:us-west-2:111122223333:user/my-user-name",
                "accountId": "111122223333",
                "accessKeyId": "AKIAIOSFODNN7EXAMPLE",
                "userName": "my-user-name"
            },
            "eventTime": "2022-03-07T15:07:11Z",
            "eventSource": "transcribe.amazonaws.com",
            "eventName": "GetTranscriptionJob",
            "awsRegion": "us-west-2",
            "sourceIPAddress": "127.0.0.1",
            "userAgent": "[]",
            "requestParameters": {
                "transcriptionJobName": "my-first-transcription-job"
            },
            "responseElements": {
                "transcriptionJob": {
                    "settings": {

                    },
                    "transcriptionJobStatus": "COMPLETED",
                    "mediaFormat": "flac",
                    "creationTime": "2022-03-07T15:03:44.229000-08:00",
                    "transcriptionJobName": "my-first-transcription-job",
                    "languageCode": "en-US",
                    "media": {
                        "mediaFileUri": ""
                    },
                    "transcript": {
                        "transcriptFileUri": ""
                    }
                }
            },
            "requestID": "BD8798EACDD16751",
            "eventID": "607b9532-1423-41c7-b048-ec2641693c47",
            "eventType": "AwsApiCall",
            "recipientAccountId": "111122223333"
        }
    ]
}
```

Using Amazon EventBridge with Amazon Transcribe

With Amazon EventBridge, you can respond to state changes in your Amazon Transcribe jobs by initiating events in other AWS services. When a transcription job changes state, EventBridge automatically sends an event to an event stream. You create rules that define the events that you want to monitor in the event stream and the action that EventBridge should take when those events occur. For example, routing the event to another service (or target), which can then take an action. You could, for example, configure a rule to route an event to an AWS Lambda function when a transcription job has completed successfully. To define EventBridge rules, refer to the following sections.

You can receive notifications for events through multiple channels, including email, Amazon Q Developer in chat applications chat notifications, or AWS Console Mobile Application push notifications. You can also see notifications in the Console Notifications Center. If you want to set up notifications, you can use AWS User Notifications. AWS User Notifications supports aggregation, which can reduce the number of notifications you receive during specific events.

Defining EventBridge rules

To define EventBridge rules, use the AWS Management Console. When you define a rule, use Amazon Transcribe as the service name. For an example of how to create an EventBridge rule, see Amazon EventBridge rules.

Before using EventBridge, note the following definitions:

- **Event**–An event indicates a change in the state of one of your transcription jobs. For example, when the `TranscriptionJobStatus` of a job changes from `IN_PROGRESS` to `COMPLETED`.
- **Target**–A target is another AWS service that processes an event. For example, AWS Lambda or Amazon Simple Notification Service (Amazon SNS). A target receives events in JSON format.
- **Rule**–A rule matches incoming events that you want EventBridge to watch for and routes them to a target or targets for processing. If a rule routes an event to multiple targets, all of the targets process the event in parallel. A rule can customize the JSON sent to the target.

Amazon EventBridge events are emitted on a best-effort basis. For more information about creating and managing events in EventBridge, see Amazon EventBridge events in the *Amazon EventBridge User Guide*.

The following is an example of an EventBridge rule for Amazon Transcribe that's initiated when a transcription job's status changes to `COMPLETED` or `FAILED`.

```
{
    "source": [
        "aws.transcribe"
    ],
    "detail-type": [
        "Transcribe Job State Change"
    ],
    "detail": {
        "TranscriptionJobStatus": [
            "COMPLETED",
            "FAILED"
        ]
    }
}
```

The rule contains the following fields:

- `source`–The source of the event. For Amazon Transcribe, this is always `aws.transcribe`.
- `detail-type`–An identifier for the details of the event. For Amazon Transcribe, this is always `Transcribe Job State Change`.
- `detail`–The new job status of the transcription job. In this example, the rule initiates an event when the job status changes to `COMPLETED` or `FAILED`.

Amazon Transcribe events

Amazon EventBridge logs several Amazon Transcribe events:

- Transcription job events
- Language identification events
- Call Analytics events
- Call Analytics post-call events
- Vocabulary events

These events all contain the following shared fields:

- `version`: The version of the event data. This value is always `0`.
- `id`: A unique identifier generated by EventBridge for the event.
- `detail-type`: An identifier for the details of the event. For example, `Transcribe Job State Change`.
- `source`: The source of the event. For Amazon Transcribe this is always `aws.transcribe`.
- `account`: The AWS account ID of the account that generated the API call.
- `time`: The date and time the event is delivered.
- `region`: The AWS Region in which the request is made.
- `resources`: The resources used by the API call. For Amazon Transcribe, this field is always empty.
- `detail`: Additional details about the event.
  - `FailureReason`: This field is present if the state or status changes to `FAILED`, and describes the reason for the `FAILED` state or status.
  - Each event type has additional unique fields that are displayed under `detail`. These unique fields are defined in the following sections after each event example.

Transcription job events

When a job's state changes from `IN_PROGRESS` to `COMPLETED` or `FAILED`, Amazon Transcribe generates an event. To identify the job that changed state and initiate the event in your target, use the event's `TranscriptionJobName` field. An Amazon Transcribe event contains the following information. A `FailureReason` field is added under `detail` if your transcription job status is `FAILED`.

Note that this event applies only to the `StartTranscriptionJob` API operation.

```
{
    "version": "0",
    "id": "event ID",
    "detail-type":"Transcribe Job State Change",
    "source": "aws.transcribe",
    "account": "111122223333",
    "time": "timestamp",
    "region": "us-west-2",
    "resources": [],
    "detail": {
          "TranscriptionJobName": "my-first-transcription-job",
```

```
        "TranscriptionJobStatus": "COMPLETED" (or "FAILED")
    }
}
```

- `TranscriptionJobName`: The unique name you chose for your transcription job.
- `TranscriptionJobStatus` : The status of the transcription job. This can be `COMPLETED` or `FAILED`.

Language identification events

When you enable automatic language identification, Amazon Transcribe generates an event when the language identification state is `COMPLETED` or `FAILED`. To identify the job that changed state and initiate the event in your target, use the event's `JobName` field. An Amazon Transcribe event contains the following information. A `FailureReason` field is added under `detail` if your language identification status is `FAILED`.

Note that this event applies only to the `StartTranscriptionJob` API operation when the `LanguageIdSettings` parameter is included.

```
{
    "version": "0",
    "id": "event ID",
    "detail-type": "Language Identification State Change",
    "source": "aws.transcribe",
    "account": "111122223333",
    "time": "timestamp",
    "region": "us-west-2",
    "resources": [],
    "detail": {
        "JobType": "TranscriptionJob",
        "JobName": "my-first-lang-id-job",
        "LanguageIdentificationStatus": "COMPLETED" (or "FAILED")
    }
}
```

- `JobType`: For transcription jobs, this value must be `TranscriptionJob`.
- `JobName`: The unique name of your transcription job.
- `LanguageIdentificationStatus`: The status of language identification in a transcription job. This can be `COMPLETED` or `FAILED`.

Call Analytics events

When a Call Analytics job state changes from `IN_PROGRESS` to `COMPLETED` or `FAILED`, Amazon Transcribe generates an event. To identify the Call Analytics job that changed state and initiates the event in your target, use the event's `JobName` field. An Amazon Transcribe event contains the following information. A `FailureReason` field is added under `detail` if your Call Analytics job status is `FAILED`.

Note that this event applies only to the `StartCallAnalyticsJob` API operation.

```
{
    "version": "0",
    "id": "event ID",
    "detail-type": "Call Analytics Job State Change",
    "source": "aws.transcribe",
    "account": "111122223333",
    "time": "timestamp",
    "region": "us-west-2",
    "resources": [],
    "detail": {
        "JobName": "my-first-analytics-job",
        "JobStatus": "COMPLETED" (or "FAILED"),
        "FailureReason": "failure reason", // only present when JobStatus is FAILED
        "AnalyticsJobDetails": { // only when you enable optional features such as
 Generative Call Summarization
            "Skipped": []
        }
    }
}
```

- `JobName`: The unique name of your Call Analytics transcription job.
- `JobStatus`: The status of your Call Analytics transcription job. This can be either `COMPLETED` or `FAILED`.
- `FailureReason`: This field is present only when the `JobStatus` is `FAILED`, and describes the reason for the failure.
- `AnalyticsJobDetails`: The details of your Call Analytics transcription job, including information about skipped analytics features.

Call Analytics post-call events

When a post-call analytics transcription changes state from `IN_PROGRESS` to `COMPLETED` or `FAILED`, Amazon Transcribe generates an event. To identify the Call Analytics post-call job that changed state and initiate the event in your target, use the event's `StreamingSessionId` field.

Note that this event applies only to the `StartCallAnalyticsStreamTranscription` API operation when the `PostCallAnalyticsSettings` parameter is included.

A `COMPLETED` event contains the following information:

```
{
    "version": "0",
    "id": "event ID",
    "detail-type": "Call Analytics Post Call Job State Change",
    "source": "aws.transcribe",
    "account": "111122223333",
    "time": "timestamp",
    "region": "us-west-2",
    "resources": [],
    "detail": {
        "StreamingSessionId": "session-id",
        "PostCallStatus": "COMPLETED",
        "Transcript": {
            "RedactedTranscriptFileUri": "s3://amzn-s3-demo-bucket/my-output-files/my-
redacted-file.JSON",
            "TranscriptFileUri": "s3://amzn-s3-demo-bucket/my-output-files/my-
file.JSON"
        },
        "Media": {
            "MediaFileUri": "s3://amzn-s3-demo-bucket/my-output-files/my-redacted-
file.WAV",
            "RedactedMediaFileUri": "s3://amzn-s3-demo-bucket/my-output-files/my-
redacted-file.WAV"
        }
    }
}
```

A `FAILED` event contains the following information:

```
{
    "version": "0",
    "id": "event ID",
```

```
    "detail-type": "Call Analytics Post Call Job State Change",
    "source": "aws.transcribe",
    "account": "111122223333",
    "time": "timestamp",
    "region": "us-west-2",
    "resources": [],
    "detail": {
        "StreamingSessionId": "session-id",
        "PostCallStatus": "FAILED"
    }
}
```

- `StreamingSessionId`: The identification number assigned to your real-time Call Analytics transcription request.
- `PostCallStatus`: The status of your post-call Call Analytics transcription. This can be either `COMPLETED` or `FAILED`.
- `Transcript`: The URI of your redacted and unredacted transcripts.
- `Media`: The URI of your redacted and unredacted audio files.

AWS HealthScribe post stream analytics events

When a state changes for a AWS HealthScribe post-stream analytics operation, such as a ClinicalNoteGenerationResult changing from `IN_PROGRESS` to `COMPLETED`, AWS HealthScribe generates an event with the following information:

```
{
   "version":"0",
   "id":"event ID",
   "detail-type":"MedicalScribe Post Stream Analytics Update",
   "source":"aws.transcribe",
   "account":"111122223333",
   "time":"timestamp",
   "region":"us-east-1",
   "resources":[],
   "detail":{
      "SessionId": <SessionID>,
      "UpdateType": "ClinicalNoteGenerationResult",
      "ClinicalNoteGenerationResult": {
          "ClinicalNoteOutputLocation": s3://amzn-s3-demo-bucket/clinical-note-output-
files/clinical-notes.JSON,
```

```
            "TranscriptOutputLocation": s3://amzn-s3-demo-bucket/my-output-files/my-
file.JSON,
            "Status": <IN_PROGRESS | COMPLETED | FAILED>,
            "FailureReason": <failure_reason>
        }
    }
}
```

- `UpdateType`: The type of post-stream analytics operation that generated the event. The content of the result object varies depending on the `UpdateType`.
- `SessionId`: The identification number for your AWS HealthScribe stream. Use this ID to identify originating streaming session and then find the post-stream analytics that generated the event.
- `Status`: The status of the post-stream analytics operation. This can be `IN_PROGRESS`, `COMPLETED`, or `FAILED`.
- `ClinicalNoteOutputLocation`: The URI of the output Amazon S3 bucket for the `ClinicalNoteGenerationResult`.
- `TranscriptOutputLocation`: The URI of your transcript.

Vocabulary events

When a custom vocabulary's state changes from `PENDING` to `READY` or `FAILED`, Amazon Transcribe generates an event. To identify the custom vocabulary that changed state and initiate the event in your target, use the event's `VocabularyName` field. An Amazon Transcribe event contains the following information. A `FailureReason` field is added under `detail` if your custom vocabulary state is `FAILED`.

> **Note**
>
> This event applies only to the `CreateVocabulary` API operation.

```
{
    "version": "0",
    "id": "event ID",
    "detail-type": "Vocabulary State Change",
    "source": "aws.transcribe",
    "account": "111122223333",
    "time": "timestamp",
```

```
    "region": "us-west-2",
    "resources": [],
    "detail": {
        "VocabularyName": "unique-vocabulary-name",
        "VocabularyState": "READY" (or "FAILED")
    }
}
```

- `VocabularyName`: The unique name of your custom vocabulary.
- `VocabularyState`: The processing state of your custom vocabulary. This can be `READY` or `FAILED`.

Compliance validation for Amazon Transcribe

To learn whether an AWS service is within the scope of specific compliance programs, see AWS services in Scope by Compliance Program and choose the compliance program that you are interested in. For general information, see AWS Compliance Programs.

You can download third-party audit reports using AWS Artifact. For more information, see Downloading Reports in AWS Artifact.

Your compliance responsibility when using AWS services is determined by the sensitivity of your data, your company's compliance objectives, and applicable laws and regulations. For more information about your compliance responsibility when using AWS services, see AWS Security Documentation.

Resilience in Amazon Transcribe

The AWS global infrastructure is built around AWS Regions and Availability Zones. AWS Regions provide multiple physically separated and isolated Availability Zones, which are connected with low-latency, high-throughput, and highly redundant networking. With Availability Zones, you can design and operate applications and databases that automatically fail over between zones without interruption. Availability Zones are more highly available, fault tolerant, and scalable than traditional single or multiple data center infrastructures.

For more information about AWS Regions and Availability Zones, see AWS Global Infrastructure.

Infrastructure security in Amazon Transcribe

As a managed service, Amazon Transcribe is protected by AWS global network security. For information about AWS security services and how AWS protects infrastructure, see AWS Cloud Security. To design your AWS environment using the best practices for infrastructure security, see Infrastructure Protection in *Security Pillar AWS Well-Architected Framework*.

You use AWS published API calls to access Amazon Transcribe through the network. Clients must support the following:

- Transport Layer Security (TLS). We require TLS 1.2 and recommend TLS 1.3.
- Cipher suites with perfect forward secrecy (PFS) such as DHE (Ephemeral Diffie-Hellman) or ECDHE (Elliptic Curve Ephemeral Diffie-Hellman). Most modern systems such as Java 7 and later support these modes.

Vulnerability analysis and management in Amazon Transcribe

Configuration and IT controls are a shared responsibility between AWS and you, our customer. For more information, see the AWS shared responsibility model.

Amazon Transcribe and interface VPC endpoints (AWS PrivateLink)

You can establish a private connection between your VPC and Amazon Transcribe by creating an *interface VPC endpoint*. Interface endpoints are powered by AWS PrivateLink, a technology that you can use to privately access Amazon Transcribe APIs without an internet gateway, NAT device, VPN connection, or Direct Connect connection. Instances in your VPC don't need public IP addresses to communicate with Amazon Transcribe APIs. Traffic between your VPC and Amazon Transcribe does not leave the Amazon network.

Each interface endpoint is represented by one or more Elastic Network Interfaces in your subnets.

For more information, see Interface VPC endpoints (AWS PrivateLink) in the *Amazon VPC User Guide*.

Considerations for Amazon Transcribe VPC endpoints

Before you set up an interface VPC endpoint for Amazon Transcribe, make sure that you review Interface endpoint properties and limitations in the *Amazon VPC User Guide*.

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

Creating an interface VPC endpoint for Amazon Transcribe

You can create a VPC endpoint for the Amazon Transcribe service using the Amazon VPC AWS Management Console or AWS CLI. For more information, see Creating an interface endpoint in the *Amazon VPC User Guide*.

For batch transcriptions in Amazon Transcribe, create a VPC endpoint using the following service name:

- com.amazonaws.*us-west-2*.transcribe

For streaming transcriptions in Amazon Transcribe, create a VPC endpoint using the following service name:

- com.amazonaws.*us-west-2*.transcribestreaming

If you enable private DNS for the endpoint, you can make API requests to Amazon Transcribe using its default DNS name for the AWS Region, for example, `transcribestreaming.us-east-2.amazonaws.com`.

For more information, see Accessing a service through an interface endpoint in the *Amazon VPC User Guide*.

Creating a VPC endpoint policy for Amazon Transcribe

You can attach an endpoint policy to your VPC endpoint that controls access to the streaming service or batch transcription service of Amazon Transcribe. The policy specifies the following information:

- The principal that can perform actions.
- The actions that can be performed.
- The resources on which actions can be performed.

For more information, see Controlling access to services with VPC endpoints in the *Amazon VPC User Guide*.

Example: VPC endpoint policy for Amazon Transcribe batch transcription actions

The following is an example of an endpoint policy for a batch transcription in Amazon Transcribe. When attached to an endpoint, this policy grants access to the listed Amazon Transcribe actions for all principals on all resources.

```
{
   "Statement":[
      {
         "Principal":"*",
         "Effect":"Allow",
         "Action":[
            "transcribe:StartTranscriptionJob",
            "transcribe:ListTranscriptionJobs"
         ],
         "Resource":"*"
      }
   ]
}
```

Example: VPC endpoint policy for Amazon Transcribe streaming transcription actions

The following is an example of an endpoint policy for a streaming transcription in Amazon Transcribe. When attached to an endpoint, this policy grants access to the listed Amazon Transcribe actions for all principals on all resources.

```
{
   "Statement":[
      {
         "Principal":"*",
         "Effect":"Allow",
         "Action":[
            "transcribe:StartStreamTranscription",
            "transcribe:StartStreamTranscriptionWebsocket"
         ],
         "Resource":"*"
      }
   ]
}
```

Shared subnets

You cannot create, describe, modify, or delete VPC endpoints in subnets that are shared with you. However, you can use the VPC endpoints in subnets that are shared with you. For information

about VPC sharing, see Share your VPC with other accounts in the Amazon Virtual Private Cloud guide.

Security best practices for Amazon Transcribe

The following best practices are general guidelines and don't represent a complete security solution. Because these best practices might not be appropriate or sufficient for your environment, use them as helpful considerations rather than prescriptions.

- **Use data encryption, such as AWS KMS encryption context**

  AWS KMS encryption context is a map of plain text, non-secret key:value pairs. This map represents additional authenticated data, known as encryption context pairs, which provide an added layer of security for your data.

  For more information, refer to AWS KMS encryption context.

- **Use temporary credentials whenever possible**

  Where possible, use temporary credentials instead of long-term credentials, such as access keys. For scenarios in which you need IAM users with programmatic access and long-term credentials, we recommend that you rotate access keys. Regularly rotating long-term credentials helps you familiarize yourself with the process. This is useful in case you are ever in a situation where you must rotate credentials, such as when an employee leaves your company. We recommend that you use *IAM access last used information* to rotate and remove access keys safely.

  For more information, see Rotating access keys and Security best practices in IAM.

- **Use IAM roles for applications and AWS services that require Amazon Transcribe access**

  Use an IAM role to manage temporary credentials for applications or services that need to access Amazon Transcribe. When you use a role, you don't have to distribute long-term credentials, such as passwords or access keys, to an Amazon EC2 instance or AWS service. IAM roles can supply temporary permissions that applications can use when they make requests to AWS resources.

  For more information, refer to IAM roles and Common scenarios for roles: Users, applications, and services.

- **Use tag-based access control**

You can use tags to control access within your AWS accounts. In Amazon Transcribe. tags can be added to: transcription jobs, custom vocabularies, custom vocabulary filters, and custom language models.

For more information, refer to Tag-based access control.

- **Use AWS monitoring tools**

  Monitoring is an important part of maintaining the reliability, security, availability, and performance of Amazon Transcribe and your AWS solutions. You can monitor Amazon Transcribe using CloudTrail.

  For more information, refer to Monitoring Amazon Transcribe with AWS CloudTrail.

- **Enable AWS Config**

  AWS Config can assess, audit, and evaluate the configurations of your AWS resources. Using AWS Config, you can review changes in configurations and relationships between AWS resources. You can also investigate detailed resource configuration histories and determine your overall compliance against the configurations specified in your internal guidelines. This can help you simplify compliance auditing, security analysis, change management, and operational troubleshooting.

  For more information, refer to What Is AWS Config?

Amazon Transcribe Medical

Amazon Transcribe Medical is an automatic speech recognition (ASR) service designed for medical professionals who want to transcribe medical-related speech, such as physician-dictated notes, drug safety monitoring, telemedicine appointments, or physician-patient conversations. Amazon Transcribe Medical is available through either real-time streaming (via microphone) or transcription of an uploaded file (batch).

> **Important**
>
> Amazon Transcribe Medical is not a substitute for professional medical advice, diagnosis, or treatment. Identify the right confidence threshold for your use case, and use high confidence thresholds in situations that require high accuracy. For certain use cases, results should be reviewed and verified by appropriately trained human reviewers. Amazon Transcribe Medical transcriptions should only be used in patient care scenarios after review for accuracy and sound medical judgment by trained medical professionals.

Amazon Transcribe Medical operates under a shared responsibility model, whereby AWS is responsible for protecting the infrastructure that runs Amazon Transcribe Medical and you are responsible for managing your data. For more information, see Shared Responsibility Model.

Amazon Transcribe Medical is available in US English (en-US).

For best results, use a lossless audio format, such as FLAC or WAV, with PCM 16-bit encoding. Amazon Transcribe Medical supports sample rates of 16,000 Hz or higher.

For analysis of your transcripts, you can use other AWS services, such as Amazon Comprehend Medical.

Supported specialties

Specialty	Sub-specialty	Audio input
Cardiology	none	streaming only
Neurology	none	streaming only
Oncology	none	streaming only

Specialty	Sub-specialty	Audio input
Primary Care	Family Medicine	batch, streaming
Primary Care	Internal Medicine	batch, streaming
Primary Care	Obstetrics and Gynecology (OB-GYN)	batch, streaming
Primary Care	Pediatrics	batch, streaming
Radiology	none	streaming only
Urology	none	streaming only

Region availability and quotas

Call Analytics is supported in the following AWS Regions:

Region	Transcription type
af-south-1 (Cape Town)	batch
ap-east-1 (Hong Kong)	batch
ap-northeast-1 (Tokyo)	batch, streaming
ap-northeast-2 (Seoul)	batch, streaming
ap-south-1 (Mumbai)	batch
ap-southeast-1 (Singapore)	batch
ap-southeast-2 (Sydney)	batch, streaming
ca-central-1 (Canada, Central)	batch, streaming
eu-central-1 (Frankfurt)	batch, streaming
eu-north-1 (Stockholm)	batch

Region	Transcription type
eu-west-1 (Ireland)	batch, streaming
eu-west-2 (London)	batch, streaming
eu-west-3 (Paris)	batch
me-south-1 (Bahrain)	batch
sa-east-1 (São Paulo)	batch, streaming
us-east-1 (N. Virginia)	batch, streaming
us-east-2 (Ohio)	batch, streaming
us-gov-east-1 (GovCloud, US-East)	batch, streaming
us-gov-west-1 (GovCloud, US-West)	batch, streaming
us-west-1 (San Francisco)	batch
us-west-2 (Oregon)	batch, streaming

Note that Region support differs for Amazon Transcribe, Amazon Transcribe Medical, and Call Analytics..

To get the endpoints for each supported Region, see Service endpoints in the *AWS General Reference*.

For a list of quotas that pertain to your transcriptions, refer to the Service quotas in the *AWS General Reference*. Some quotas can be changed upon request. If the **Adjustable** column contains '**Yes**', you can request an increase. To do so, select the provided link.

Medical specialties and terms

When creating a medical transcription job, you specify the language, the medical specialty, and the audio type of the source file. You input US English (en-US) as the language and `PRIMARYCARE` as the medical specialty. Entering primary care as the value enables you to generate transcriptions from source audio in the following medical specialties:

- Family Medicine
- Internal Medicine
- Obstetrics and Gynecology (OB-GYN)
- Pediatrics

You have the choice between dictation and conversation for your audio type. Choose dictation for audio files where the physician is giving a report about a patient visit or procedure. Choose conversation for audio files that involve a conversation between a physician and a patient or a conversation between physicians.

To store the output of your transcription job, select an Amazon S3 bucket that you've already created. For more information on Amazon S3 buckets see Getting Started with Amazon Simple Storage Service.

The following are the minimum number of request parameters to enter in the sample JSON.

```
{
   "MedicalTranscriptionJobName": "my-first-transcription-job",
   "LanguageCode": "en-US",
   "Media": {
       "MediaFileUri": "s3://path to your audio file"
   },
   "OutputBucketName": “your output bucket name",
   "Specialty": "PRIMARYCARE",
   "Type": "CONVERSATION"
}
```

Amazon Transcribe Medical enables you to generate alternative transcriptions. For more information, see Generating alternative transcriptions.

You can also enable speaker partitioning or identify channels in your audio. For more information, see Enabling speaker partitioning and Transcribing multi-channel audio.

Transcribing medical terms and measurements

Amazon Transcribe Medical can transcribe medical terms and measurements. Amazon Transcribe Medical outputs abbreviations for spoken terms. For example, "blood pressure" is transcribed as BP. You can find a list of conventions that Amazon Transcribe Medical uses for medical terms and

measurements in the table on this page. The *Spoken Term* column refers to the term spoken in the source audio. The *Output* column refers to the abbreviation you see in your transcription results.

You can see how the terms spoken in source audio correspond to the transcription output here.

Term spoken in source audio	Abbreviation used in output	Example output
Centigrade	C	The patient's temperature is 37.4 C.
Celsius	C	The patient's temperature is 37.4 C.
Fahrenheit	F	The patient's temperature is 101 F.
grams	g	A mass of 100 g was extracted from the patient.
meters	m	The patient is 1.8 m tall.
feet	ft	The patient is 6 ft tall.
kilos	kg	The patient weighs 80 kg.
kilograms	kg	The patient weighs 80 kg.
c c	cc	Patient received 100 cc of saline solution.
cubic centimeter	cc	Patient received 100 cc of saline solution.
milliliter	mL	Patient excreted 100 mL urine.
blood pressure	BP	Patient BP was elevated.
b p	BP	Patient BP was elevated.
X over Y	X/Y	Patient BP was 120/80.

Term spoken in source audio	Abbreviation used in output	Example output
beats per min	BPM	Patient had atrial fibrillation with heart rate of 160 BPM.
beats per minute	BPM	Patient had atrial fibrillation with heart rate of 160 BPM.
O 2	O2	Patient O2 saturation was 98%.
CO2	CO2	Patient required respiratory support for elevated CO2.
post operation	POSTOP	Patient came for POSTOP evaluation.
post op	POSTOP	Patient came for POSTOP evaluation.
cat scan	CT Scan	Patient indication of cerebral hemorrhage required use of CT Scan.
Pulse 80	P 80	Patient vitals were P 80, R 17,...
Respiration 17	R 17	Patient vitals were P 80, R 17,...
in and out	I/O	Patient was I/O sinus rhythm
L five	L5	Lumbar puncture was performed between L4 and L5

Transcribing numbers

Amazon Transcribe Medical transcribes digits as numbers instead of words. For example, the spoken number "one thousand two hundred forty-two" is transcribed as 1242.

Numbers are transcribed according to the following rules.

Rule	Description
Convert cardinal numbers greater than 10 to numbers.	• "Fifty five" > 55 • "a hundred" > 100 • "One thousand and thirty one" > 1031 • "One hundred twenty-three million four hundred fifty six thousand seven hundred eight nine" > 123,456,789
Convert cardinal numbers followed by "million" or "billion" to numbers followed by a word when "million" or "billion" is not followed by a number.	• "one hundred million" > 100 million • "one billion" > 1 billion • "two point three million" > 2.3 million
Convert ordinal numbers greater than 10 to numbers.	• "Forty third" > 43rd • "twenty sixth avenue" > 26th avenue
Convert fractions to their numeric format.	• "a quarter" > 1/4 • "three sixteenths" > 3/16 • "a half" > 1/2 • "a hundredth" > 1/100
Convert numbers less than 10 to digits if there are more than one in a row.	• "three four five" > 345 • "My phone number is four two five five five five one two one two" > 4255551212
Decimals are indicated by "dot" or "point."	• "three hundred and three dot five" > 303.5 • "three point twenty three" > 3.23 • "zero point four" > 0.4

Rule	Description
	• "point three" > 0.3
Convert the word "percent" after a number to a percent sign.	• "twenty three percent" > 23% • "twenty three point four five percent" > 23.45%
Convert the words "dollar," "US dollar," "Australian dollar, "AUD," or "USD" after a number to a dollar symbol before the number.	• "one dollar and fifteen cents" > $1.15 • "twenty three USD" > $23 • "twenty three Australian dollars" > $23
Convert the words "pounds," or "milligrams" to "lbs" or "mg".	• "twenty three pounds" > 23 lbs • "forty-five milligrams" > 45 mg
Convert the words "rupees," "Indian rupees," or "INR" after a number to rupee sign (₹) before the number.	• "twenty three rupees" > ₹23 • "fifty rupees thirty paise" > ₹50.30
Convert times to numbers.	• "seven a m eastern standard time" > 7 a.m. eastern standard time • "twelve thirty p m" > 12:30 p.m.
Combine years expressed as two digits into four. Only valid for the 20th, 21st, and 22nd centuries.	• "nineteen sixty two" > 1962 • "the year is twenty twelve" > the year is 2012 • "twenty nineteen" > 2019 • "twenty one thirty" > 2130
Convert dates to numbers.	• "May fifth twenty twelve" > May 5th 2012 • "May five twenty twelve" > May 5 2012 • "five May twenty twelve" > 5 May 2012
Separate spans of numbers by the word "to."	• "twenty three to thirty seven" > 23 to 37

Transcribing a medical conversation

You can use Amazon Transcribe Medical to transcribe a medical conversation between a clinician and a patient using either a batch transcription job or a real-time stream. Batch transcription jobs enable you to transcribe audio files. To ensure that Amazon Transcribe Medical produces transcription results with the highest possible accuracy, you must specify the medical specialty of the clinician in your transcription job or stream.

You can transcribe a clinician-patient visit in the following medical specialities:

- Cardiology – available in streaming transcription only
- Neurology – available in streaming transcription only
- Oncology – available in streaming transcription only
- Primary Care – includes the following types of medical practice:
  - Family medicine
  - Internal medicine
  - Obstetrics and Gynecology (OB-GYN)
  - Pediatrics
- Urology – available in streaming transcription only

You can improve transcription accuracy by using medical custom vocabularies. For information on how medical custom vocabularies work, see Improving transcription accuracy with medical custom vocabularies.

By default, Amazon Transcribe Medical returns the transcription with the highest confidence level. If you'd like to configure it to return alternative transcriptions, see Generating alternative transcriptions.

For information about how numbers and medical measurements appear in the transcription output, see Transcribing numbers and Transcribing medical terms and measurements.

Topics

- Transcribing an audio file of a medical conversation
- Transcribing a medical conversation in a real-time stream
- Enabling speaker partitioning
- Transcribing multi-channel audio

Transcribing an audio file of a medical conversation

Use a batch transcription job to transcribe audio files of medical conversations. You can use this to transcribe a clinician-patient dialogue. You can start a batch transcription job in either the `StartMedicalTranscriptionJob` API or the AWS Management Console.

When you start a medical transcription job with the `StartMedicalTranscriptionJob` API, you specify `PRIMARYCARE` as the value of the `Specialty` parameter.

AWS Management Console

To transcribe a clinician-patient dialogue (AWS Management Console)

To use the AWS Management Console to transcribe a clinician-patient dialogue, create a transcription job and choose **Conversation** for **Audio input type**.

1. Sign in to the AWS Management Console.
2. In the navigation pane, under Amazon Transcribe Medical, choose **Transcription jobs**.
3. Choose **Create job**.
4. On the **Specify job details** page, under **Job settings** , specify the following.
    a. **Name** – the name of the transcription job.
    b. **Audio input type** – **Conversation**
5. For the remaining fields, specify the Amazon S3 location of your audio file and where you want to store the output of your transcription job.
6. Choose **Next**.
7. Choose **Create**.

API

To transcribe a medical conversation using a batch transcription job (API)

- For the `StartMedicalTranscriptionJob` API, specify the following.
    a. For `MedicalTranscriptionJobName`, specify a name unique in your AWS account.
    b. For `LanguageCode`, specify the language code that corresponds to the language spoken in your audio file and the language of your vocabulary filter.

c. For the `MediaFileUri` parameter of the `Media` object, specify the name of the audio file that you want to transcribe.

d. For `Specialty`, specify the medical specialty of the clinician speaking in the audio file as `PRIMARYCARE`.

e. For `Type`, specify `CONVERSATION`.

f. For `OutputBucketName`, specify the Amazon S3 bucket to store the transcription results.

The following is an example request that uses the AWS SDK for Python (Boto3) to transcribe a medical conversation of a clinician in the `PRIMARYCARE` specialty and a patient.

```
from __future__ import print_function
import time
import boto3
transcribe = boto3.client('transcribe', 'us-west-2')
job_name = "my-first-med-transcription-job"
job_uri = "s3://amzn-s3-demo-bucket/my-input-files/my-audio-file.flac"
transcribe.start_medical_transcription_job(
      MedicalTranscriptionJobName = job_name,
      Media = {
        'MediaFileUri': job_uri
      },
      OutputBucketName = 'amzn-s3-demo-bucket',
      OutputKey = 'output-files/',
      LanguageCode = 'en-US',
      Specialty = 'PRIMARYCARE',
      Type = 'CONVERSATION'
  )

while True:
    status = transcribe.get_medical_transcription_job(MedicalTranscriptionJobName =
 job_name)
    if status['MedicalTranscriptionJob']['TranscriptionJobStatus'] in ['COMPLETED',
 'FAILED']:
        break
    print("Not ready yet...")
    time.sleep(5)
print(status)
```

The following example code shows the transcription results of a clinician-patient conversation.


```
{
    "jobName": "conversation-medical-transcription-job",
    "accountId": "111122223333",
    "results": {
        "transcripts": [
            {
                "transcript": "... come for a follow up visit today..."
            }
        ],
        "items": [
            {
            ...
                "start_time": "4.85",
                "end_time": "5.12",
                "alternatives": [
                    {
                        "confidence": "1.0",
                        "content": "come"
                    }
                ],
                "type": "pronunciation"
            },
            {
                "start_time": "5.12",
                "end_time": "5.29",
                "alternatives": [
                    {
                        "confidence": "1.0",
                        "content": "for"
                    }
                ],
                "type": "pronunciation"
            },
            {
                "start_time": "5.29",
                "end_time": "5.33",
                "alternatives": [
                    {
                        "confidence": "0.9955",
                        "content": "a"
                    }
```

```
                ],
                "type": "pronunciation"
            },
            {
                "start_time": "5.33",
                "end_time": "5.66",
                "alternatives": [
                    {
                        "confidence": "0.9754",
                        "content": "follow"
                    }
                ],
                "type": "pronunciation"
            },
            {
                "start_time": "5.66",
                "end_time": "5.75",
                "alternatives": [
                    {
                        "confidence": "0.9754",
                        "content": "up"
                    }
                ],
                "type": "pronunciation"
            },
            {
                "start_time": "5.75",
                "end_time": "6.02",
                "alternatives": [
                    {
                        "confidence": "1.0",
                        "content": "visit"
                    }
                ]
                ...
    },
    "status": "COMPLETED"
}
```

AWS CLI

To transcribe a medical conversation using a batch transcription job (AWS CLI)

- Run the following code.

```
aws transcribe start-medical-transcription-job \
--region us-west-2 \
--cli-input-json file://example-start-command.json
```

The following code shows the contents of `example-start-command.json`.

```
{
      "MedicalTranscriptionJobName": "my-first-med-transcription-job",
      "Media": {
          "MediaFileUri": "s3://amzn-s3-demo-bucket/my-input-files/my-audio-
file.flac"
      },
      "OutputBucketName": "amzn-s3-demo-bucket",
      "OutputKey": "my-output-files/",
      "LanguageCode": "en-US",
      "Specialty": "PRIMARYCARE",
      "Type": "CONVERSATION"
  }
```

Transcribing a medical conversation in a real-time stream

You can transcribe an audio stream of a medical conversation using either the HTTP/2 or WebSocket protocols. For information on how to start a stream using the WebSocket protocol, see Setting up a WebSocket stream. To start an HTTP/2 stream, use the `StartMedicalStreamTranscription` API.

You can transcribe streaming audio in the following medical specialties:

- Cardiology
- Neurology
- Oncology

- Primary Care
- Urology

Each medical specialty includes many types of procedures and appointments. Clinicians therefore dictate many different types of notes. Use the following examples as guidance to help you specify the value of the `specialty` URI parameter of the WebSocket request, or the `Specialty` parameter of the `StartMedicalStreamTranscription` API:

- For electrophysiology or echocardiography consultations, choose `CARDIOLOGY`.
- For medical oncology, surgical oncology, or radiation oncology consultations, choose `ONCOLOGY`.
- For a physician providing a consultation to a patient who had a stroke, either a transient ischemic attack or a cerebrovascular attack, choose `NEUROLOGY`.
- For a consultation around urinary incontinence, choose `UROLOGY`.
- For yearly checkup or urgent care visits, choose `PRIMARYCARE`.
- For inpatient hospitalist visits, choose `PRIMARYCARE`.
- For consultations regarding fertility, tubal ligation, IUD insertion, or abortion, choose `PRIMARYCARE`.

AWS Management Console

To transcribe a streaming medical conversation (AWS Management Console)

To use the AWS Management Console to transcribe a clinician-patient dialogue in real-time stream, choose the option to transcribe a medical conversation, start the stream, and begin speaking into the microphone.

1. Sign in to the AWS Management Console.
2. In the navigation pane, under Amazon Transcribe Medical, choose **Real-time transcription**.
3. Choose **Conversation**.
4. For **Medical specialty**, choose the clinician's specialty.
5. Choose **Start streaming**.
6. Speak into the microphone.

Transcribing a medical conversation in an HTTP/2 stream

The following is the syntax for the parameters of an HTTP/2 request.

To transcribe an HTTP/2 stream of a medical conversation, use the `StartMedicalStreamTranscription` API and specify the following:

- `LanguageCode` – The language code. The valid value is `en-US`
- `MediaEncoding` – The encoding used for the input audio. Valid values are `pcm`, `ogg-opus`, and `flac`.
- `Specialty` – The specialty of the medical professional.
- `Type` – `CONVERSATION`

To improve transcription accuracy of specific terms in a real-time stream, use a custom vocabulary. To enable a custom vocabulary, set the value of `VocabularyName` parameter to the name of the custom vocabulary that you want to use. For more information, see Improving transcription accuracy with medical custom vocabularies.

To label the speech from different speakers, set the `ShowSpeakerLabel` parameter to `true`. For more information, see Enabling speaker partitioning.

For more information on setting up an HTTP/2 stream to transcribe a medical conversation, see Setting up an HTTP/2 stream.

Transcribing a medical conversation in a WebSocket stream

You can use a WebSocket request to transcribe a medical conversation. When you make a WebSocket request, you create a presigned URI. This URI contains the information needed to set up the audio stream between your application and Amazon Transcribe Medical. For more information on creating WebSocket requests, see Setting up a WebSocket stream.

Use the following template to create your presigned URI.

```
GET wss://transcribestreaming.us-west-2.amazonaws.com:8443/medical-stream-
transcription-websocket
?language-code=languageCode
&X-Amz-Algorithm=AWS4-HMAC-SHA256
&X-Amz-Credential=AKIAIOSFODNN7EXAMPLE%2F20220208%2Fus-
west-2%2Ftranscribe%2Faws4_request
&X-Amz-Date=20220208T235959Z
```

```
&X-Amz-Expires=300
&X-Amz-Security-Token=security-token
&X-Amz-Signature=Signature Version 4 signature
&X-Amz-SignedHeaders=host
&media-encoding=flac
&sample-rate=16000
&session-id=sessionId
&specialty=medicalSpecialty
&type=CONVERSATION
&vocabulary-name=vocabularyName
&show-speaker-label=boolean
```

To improve transcription accuracy of specific terms in a real-time stream, use a custom vocabulary. To enable a custom vocabulary, set the value of `vocabulary-name` to the name of the custom vocabulary that you want to use. For more information, see Improving transcription accuracy with medical custom vocabularies.

To label the speech from different speakers, set the `show-speaker-label` parameter in to `true`. For more information, see Enabling speaker partitioning.

For more information on creating pre-signed URIs, see Setting up a WebSocket stream.

Enabling speaker partitioning

To enable speaker partitioning in Amazon Transcribe Medical, use *speaker diarization*. This enables you to see what the patient said and what the clinician said in the transcription output.

When you enable speaker diarization, Amazon Transcribe Medical labels each speaker *utterance* with a unique identifier for each speaker. An *utterance* is a unit of speech that is typically separated from other utterances by silence. In batch transcription, an utterance from the clinician could receive a label of `spk_0` and an utterance the patient could receive a label of `spk_1`.

If an utterance from one speaker overlaps with an utterance from another speaker, Amazon Transcribe Medical orders them in the transcription by their start times. Utterances that overlap in the input audio don't overlap in the transcription output.

You can enable speaker diarization when you transcribe an audio file using batch transcription job, or in a real-time stream.

Topics

- Enabling speaker partitioning in batch transcriptions
- Enabling speaker partitioning in real-time streams

Enabling speaker partitioning in batch transcriptions

You can enable speaker partitioning in a batch transcription job using either the `StartMedicalTranscriptionJob` API or the AWS Management Console. This enables you to partition the text per speaker in a clinician-patient conversation and determine who said what in the transcription output.

AWS Management Console

To use the AWS Management Console to enable speaker diarization in your transcription job, you enable audio identification and then speaker partitioning.

1. Sign in to the AWS Management Console.
2. In the navigation pane, under Amazon Transcribe Medical, choose **Transcription jobs**.
3. Choose **Create job**.
4. On the **Specify job details** page, provide information about your transcription job.
5. Choose **Next**.
6. Enable **Audio identification**.
7. For **Audio identification type**, choose **Speaker partitioning**.
8. For **Maximum number of speakers**, enter the maximum number of speakers that you think are speaking in your audio file.
9. Choose **Create**.

API

To enable speaker partitioning using a batch transcription job (API)

- For the `StartMedicalTranscriptionJob` API, specify the following.

    a. For `MedicalTranscriptionJobName`, specify a name that is unique in your AWS account.

    b. For `LanguageCode`, specify the language code that corresponds to the language spoken in the audio file.

c. For the `MediaFileUri` parameter of the `Media` object, specify the name of the audio file that you want to transcribe.

d. For `Specialty`, specify the medical specialty of the clinician speaking in the audio file.

e. For `Type`, specify `CONVERSATION`.

f. For `OutputBucketName`, specify the Amazon S3 bucket to store the transcription results.

g. For the `Settings` object, specify the following.

   i. `ShowSpeakerLabels` – `true`.

   ii. `MaxSpeakerLabels` – An integer between 2 and 10 to indicate the number of speakers that you think are speaking in your audio.

The following request uses the AWS SDK for Python (Boto3) to start a batch transcription job of a primary care clinician patient dialogue with speaker partitioning enabled.

```
from __future__ import print_function
import time
import boto3
transcribe = boto3.client('transcribe', 'us-west-2')
job_name = "my-first-transcription-job"
job_uri = "s3://amzn-s3-demo-bucket/my-input-files/my-media-file.flac"
transcribe.start_medical_transcription_job(
    MedicalTranscriptionJobName = job_name,
    Media={
        'MediaFileUri': job_uri
    },
    OutputBucketName = 'amzn-s3-demo-bucket',
    OutputKey = 'my-output-files/',
    LanguageCode = 'en-US',
    Specialty = 'PRIMARYCARE',
    Type = 'CONVERSATION',
    OutputBucketName = 'amzn-s3-demo-bucket',
Settings = {'ShowSpeakerLabels': True,
         'MaxSpeakerLabels': 2
        }
        )
while True:
    status = transcribe.get_medical_transcription_job(MedicalTranscriptionJobName =
 job_name)
```

```
    if status['MedicalTranscriptionJob']['TranscriptionJobStatus'] in ['COMPLETED',
 'FAILED']:
        break
    print("Not ready yet...")
    time.sleep(5)
print(status)
```

The following example code shows the transcription results of a transcription job with speaker partitioning enabled.

```
{
    "jobName": "job ID",
    "accountId": "111122223333",
    "results": {
        "transcripts": [
            {
                "transcript": "Professional answer."
            }
        ],
        "speaker_labels": {
            "speakers": 1,
            "segments": [
                {
                    "start_time": "0.000000",
                    "speaker_label": "spk_0",
                    "end_time": "1.430",
                    "items": [
                        {
                            "start_time": "0.100",
                            "speaker_label": "spk_0",
                            "end_time": "0.690"
                        },
                        {
                            "start_time": "0.690",
                            "speaker_label": "spk_0",
                            "end_time": "1.210"
                        }
                    ]
                }
            ]
```

```
        },
        "items": [
            {
                "start_time": "0.100",
                "end_time": "0.690",
                "alternatives": [
                    {
                        "confidence": "0.8162",
                        "content": "Professional"
                    }
                ],
                "type": "pronunciation"
            },
            {
                "start_time": "0.690",
                "end_time": "1.210",
                "alternatives": [
                    {
                        "confidence": "0.9939",
                        "content": "answer"
                    }
                ],
                "type": "pronunciation"
            },
            {
                "alternatives": [
                    {
                        "content": "."
                    }
                ],
                "type": "punctuation"
            }
        ]
    },
    "status": "COMPLETED"
}
```

AWS CLI

To transcribe an audio file of a conversation between a clinician practicing primary care and a patient (AWS CLI)

- Run the following code.

```
aws transcribe start-transcription-job \
--region us-west-2 \
--cli-input-json file://example-start-command.json
```

The following code shows the contents of `example-start-command.json`.

```
{
    "MedicalTranscriptionJobName": "my-first-med-transcription-job",
     "Media": {
          "MediaFileUri": "s3://amzn-s3-demo-bucket/my-input-files/my-audio-
file.flac"
      },
      "OutputBucketName": "amzn-s3-demo-bucket",
      "OutputKey": "my-output-files/",
      "LanguageCode": "en-US",
      "Specialty": "PRIMARYCARE",
      "Type": "CONVERSATION",
      "Settings":{
          "ShowSpeakerLabels": true,
          "MaxSpeakerLabels": 2
        }
}
```

Enabling speaker partitioning in real-time streams

To partition speakers and label their speech in a real-time stream, use the AWS Management Console or a streaming request. Speaker partitioning works best for between two and five speakers in a stream. Although Amazon Transcribe Medical can partition more than five speakers in a stream, the accuracy of the partitions decrease if you exceed that number.

To start an HTTP/2 request, use the `StartMedicalStreamTranscription` API. To start a WebSocket request, use a pre-signed URI. The URI contains the information required to set up bi-directional communication between your application and Amazon Transcribe Medical.

Enabling speaker partitioning in audio that is spoken into your microphone (AWS Management Console)

You can use the AWS Management Console to start a real-time stream of a clinician-patient conversation, or a dictation that is spoken into your microphone in real-time.

1. Sign in to the AWS Management Console.
2. In the navigation pane, for Amazon Transcribe Medical choose **Real-time transcription**.
3. For **Audio input type**, choose the type of medical speech that you want to transcribe.
4. For **Additional settings**, choose **Speaker partitioning**.
5. Choose **Start streaming** to start transcribing your real-time audio.
6. Speak into the microphone.

Enabling speaker partitioning in an HTTP/2 stream

To enable speaker partitioning in an HTTP/2 stream of a medical conversation, use the `StartMedicalStreamTranscription` API and specify the following:

- For `LanguageCode`, specify the language code that corresponds to the language in the stream. The valid value is `en-US`.
- For `MediaSampleHertz`, specify the sample rate of the audio.
- For `Specialty`, specify the medical specialty of the provider.
- `ShowSpeakerLabel` – `true`

For more information on setting up an HTTP/2 stream to transcribe a medical conversation, see Setting up an HTTP/2 stream.

Enabling speaker partitioning in a WebSocket request

To partition speakers in WebSocket streams with the API, use the following format to create a pre-signed URI to start a WebSocket request and set `show-speaker-label` to `true`.

```
GET wss://transcribestreaming.us-west-2.amazonaws.com:8443/medical-stream-
transcription-websocket
?language-code=languageCode
&X-Amz-Algorithm=AWS4-HMAC-SHA256
&X-Amz-Credential=AKIAIOSFODNN7EXAMPLE%2F20220208%2Fus-
west-2%2Ftranscribe%2Faws4_request
&X-Amz-Date=20220208T235959Z
&X-Amz-Expires=300
&X-Amz-Security-Token=security-token
&X-Amz-Signature=Signature Version 4 signature
&X-Amz-SignedHeaders=host
&media-encoding=flac
&sample-rate=16000
&session-id=sessionId
&specialty=medicalSpecialty
&type=CONVERSATION
&vocabulary-name=vocabularyName
&show-speaker-label=boolean
```

The following code shows the truncated example response of a streaming request.

```
{
  "Transcript": {
    "Results": [
      {
        "Alternatives": [
          {
            "Items": [
              {
                "Confidence": 0.97,
                "Content": "From",
                "EndTime": 18.98,
                "Speaker": "0",
                "StartTime": 18.74,
                "Type": "pronunciation",
                "VocabularyFilterMatch": false
              },
              {
                "Confidence": 1,
                "Content": "the",
                "EndTime": 19.31,
```

```
            "Speaker": "0",
            "StartTime": 19,
            "Type": "pronunciation",
            "VocabularyFilterMatch": false
          },
          {
            "Confidence": 1,
            "Content": "last",
            "EndTime": 19.86,
            "Speaker": "0",
            "StartTime": 19.32,
            "Type": "pronunciation",
            "VocabularyFilterMatch": false
          },
        ...
          {
            "Confidence": 1,
            "Content": "chronic",
            "EndTime": 22.55,
            "Speaker": "0",
            "StartTime": 21.97,
            "Type": "pronunciation",
            "VocabularyFilterMatch": false
          },
          ...
            "Confidence": 1,
            "Content": "fatigue",
            "EndTime": 24.42,
            "Speaker": "0",
            "StartTime": 23.95,
            "Type": "pronunciation",
            "VocabularyFilterMatch": false
          },
          {
            "EndTime": 25.22,
            "StartTime": 25.22,
            "Type": "speaker-change",
            "VocabularyFilterMatch": false
          },
          {
            "Confidence": 0.99,
            "Content": "True",
            "EndTime": 25.63,
            "Speaker": "1",
```

```
                "StartTime": 25.22,
                "Type": "pronunciation",
                "VocabularyFilterMatch": false
              },
              {
                "Content": ".",
                "EndTime": 25.63,
                "StartTime": 25.63,
                "Type": "punctuation",
                "VocabularyFilterMatch": false
              }
            ],
            "Transcript": "From the last note she still has mild sleep deprivation and
 chronic fatigue True."
          }
        ],
        "EndTime": 25.63,
        "IsPartial": false,
        "ResultId": "XXXXXXXXX-XXXX-XXXX-XXXX-XXXXXXXXXXX",
        "StartTime": 18.74
      }
    ]
  }
}
```


Amazon Transcribe Medical breaks your incoming audio stream based on natural speech segments, such as a change in speaker or a pause in the audio. The transcription is returned progressively to your application, with each response containing more transcribed speech until the entire segment is transcribed. The preceding code is a truncated example of a fully-transcribed speech segment. Speaker labels only appear for entirely transcribed segments.

The following list shows the organization of the objects and parameters in a streaming transcription output.

`Transcript`

Each speech segment has its own `Transcript` object.

`Results`

Each `Transcript` object has its own `Results` object. This object contains the `isPartial` field. When its value is `false`, the results returned are for an entire speech segment.

`Alternatives`

Each `Results` object has an `Alternatives` object.

`Items`

Each `Alternatives` object has its own `Items` object that contains information about each word and punctuation mark in the transcription output. When you enable speaker partitioning, each word has a `Speaker` label for fully-transcribed speech segments. Amazon Transcribe Medical uses this label to assign a unique integer to each speaker in the stream. The `Type` parameter having a value of `speaker-change` indicates that one person has stopped speaking and that another person is about to begin.

`Transcript`

Each Items object contains a transcribed speech segment as the value of the `Transcript` field.

For more information about WebSocket requests, see Setting up a WebSocket stream.

Transcribing multi-channel audio

If you have an audio file or stream that has multiple channels, you can use *channel identification* to transcribe the speech from each of those channels. Amazon Transcribe Medical transcribes the speech from each channel separately. It combines the separate transcriptions of each channel into a single transcription output.

Use channel identification to identify the separate channels in your audio and transcribe the speech from each of those channels. Enable this in situations such as a caller and agent scenario. Use this to distinguish a caller from an agent in recordings or streams from contact centers that perform drug safety monitoring.

You can enable channel identification for both batch processing and real-time streaming. The following list describes how to enable it for each method.

- Batch transcription – AWS Management Console and `StartMedicalTranscriptionJob` API
- Streaming transcription – WebSocket streaming and `StartMedicalStreamTranscription` API

Transcribing multi-channel audio files

When you transcribe an audio file, Amazon Transcribe Medical returns a list of *items* for each channel. An item is a transcribed word or punctuation mark. Each word has a start time and an end time. If a person on one channel speaks over a person on a separate channel, the start times and end times of the items for each channel overlap while the individuals are speaking over each other.

By default, you can transcribe audio files with two channels. You can request a quota increase if you need to transcribe files that have more than two channels. For information about requesting a quota increase, see AWS service quotas.

To transcribe multi-channel audio in a batch transcription job, use the AWS Management Console or the `StartMedicalTranscriptionJob` API.

AWS Management Console

To use the AWS Management Console to enable channel identification in your batch transcription job, you enable audio identification and then channel identification. Channel identification is a subset of audio identification in the AWS Management Console.

1. Sign in to the AWS Management Console.
2. In the navigation pane, under Amazon Transcribe Medical, choose **Transcription jobs**.
3. Choose **Create job**.
4. On the **Specify job details** page, provide information about your transcription job.
5. Choose **Next**.
6. Enable **Audio identification**.
7. For **Audio identification type**, choose **Channel identification**.
8. Choose **Create**.

API

To transcribe a multi-channel audio file (API)

- For the `StartMedicalTranscriptionJob` API, specify the following.

  a. For `TranscriptionJobName`, specify a name unique to your AWS account.

  b. For `LanguageCode`, specify the language code that corresponds to the language spoken in the audio file. The valid value is `en-US`.

c. For the `MediaFileUri` parameter of the `Media` object, specify the name of the media file that you want to transcribe.

d. For the `Settings` object, set `ChannelIdentification` to `true`.

The following is an example request using the AWS SDK for Python (Boto3).

```
from __future__ import print_function
import time
import boto3
transcribe = boto3.client('transcribe', 'us-west-2')
job_name = "my-first-transcription-job"
job_name = "my-first-med-transcription-job"
job_uri = "s3://amzn-s3-demo-bucket/my-input-files/my-media-file.flac"
transcribe.start_medical_transcription_job(
      MedicalTranscriptionJobName = job_name,
      Media = {
        'MediaFileUri': job_uri
      },
      OutputBucketName = 'amzn-s3-demo-bucket',
      OutputKey = 'output-files/',
      LanguageCode = 'en-US',
      Specialty = 'PRIMARYCARE',
      Type = 'CONVERSATION',
      Settings = {
        'ChannelIdentification': True
      }
)
while True:
    status = transcribe.get_transcription_job(MedicalTranscriptionJobName = job_name)
    if status['MedicalTranscriptionJob']['TranscriptionJobStatus'] in ['COMPLETED',
 'FAILED']:
        break
    print("Not ready yet...")
    time.sleep(5)
print(status)
```

AWS CLI

To transcribe a multi-channel audio file using a batch transcription job (AWS CLI)

- Run the following code.

```
aws transcribe start-medical-transcription-job \
--region us-west-2 \
--cli-input-json file://example-start-command.json
```

The following is the code of `example-start-command.json`.

```
{
      "MedicalTranscriptionJobName": "my-first-med-transcription-job",
      "Media": {
          "MediaFileUri": "s3://amzn-s3-demo-bucket/my-input-files/my-audio-
file.flac"
      },
      "OutputBucketName": "amzn-s3-demo-bucket",
      "OutputKey": "my-output-files/",
      "LanguageCode": "en-US",
      "Specialty": "PRIMARYCARE",
      "Type": "CONVERSATION",

        "Settings":{
          "ChannelIdentification": true
        }
}
```

The following code shows the transcription output for an audio file that has a conversation on two channels.

```
{
  "jobName": "job id",
  "accountId": "111122223333",
  "results": {
    "transcripts": [
      {
```

```
        "transcript": "When you try ... It seems to ..."
      }
    ],
    "channel_labels": {
      "channels": [
        {
          "channel_label": "ch_0",
          "items": [
            {
              "start_time": "12.282",
              "end_time": "12.592",
              "alternatives": [
                {
                  "confidence": "1.0000",
                  "content": "When"
                }
              ],
              "type": "pronunciation"
            },
            {
              "start_time": "12.592",
              "end_time": "12.692",
              "alternatives": [
                {
                  "confidence": "0.8787",
                  "content": "you"
                }
              ],
              "type": "pronunciation"
            },
            {
              "start_time": "12.702",
              "end_time": "13.252",
              "alternatives": [
                {
                  "confidence": "0.8318",
                  "content": "try"
                }
              ],
              "type": "pronunciation"
            },
            ...
         ]
      },
```

```
      {
          "channel_label": "ch_1",
          "items": [
            {
              "start_time": "12.379",
              "end_time": "12.589",
              "alternatives": [
                {
                  "confidence": "0.5645",
                  "content": "It"
                }
              ],
              "type": "pronunciation"
            },
            {
              "start_time": "12.599",
              "end_time": "12.659",
              "alternatives": [
                {
                  "confidence": "0.2907",
                  "content": "seems"
                }
              ],
              "type": "pronunciation"
            },
            {
              "start_time": "12.669",
              "end_time": "13.029",
              "alternatives": [
                {
                  "confidence": "0.2497",
                  "content": "to"
                }
              ],
              "type": "pronunciation"
            },
            ...
        ]
    }
}
```

Transcribing multi-channel audio streams

You can transcribe audio from separate channels in either HTTP/2 or WebSocket streams using the `StartMedicalStreamTranscription` API.

By default, you can transcribe streams with two channels. You can request a quota increase if you need to transcribe streams that have more than two channels. For information about requesting a quota increase, see AWS service quotas.

Transcribing multi-channel audio in an HTTP/2 stream

To transcribe multi-channel audio in an HTTP/2 stream, use the StartMedicalStreamTranscription API and specify the following:

- `LanguageCode` – The language code of the audio. The valid value is `en-US`.
- `MediaEncoding` – The encoding of the audio. Valid values are `ogg-opus`, `flac`, and `pcm`.
- `EnableChannelIdentification` – `true`
- `NumberOfChannels` – the number of channels in your streaming audio.

For more information on setting up an HTTP/2 stream to transcribe a medical conversation, see Setting up an HTTP/2 stream.

Transcribing multi-channel audio in a WebSocket stream

To partition speakers in WebSocket streams, use the following format to create a pre-signed URI and start a WebSocket request. Specify `enable-channel-identification` as `true` and the number of channels in your stream in `number-of-channels`. A pre-signed URI contains the information needed to set up bi-directional communication between your application and Amazon Transcribe Medical.

```
GET wss://transcribestreaming.us-west-2.amazonaws.com:8443/medical-stream-
transcription-websocket
?language-code=languageCode
&X-Amz-Algorithm=AWS4-HMAC-SHA256
&X-Amz-Credential=AKIAIOSFODNN7EXAMPLE%2F20220208%2Fus-
west-2%2Ftranscribe%2Faws4_request
&X-Amz-Date=20220208T235959Z
&X-Amz-Expires=300
&X-Amz-Security-Token=security-token
```

```
&X-Amz-Signature=Signature Version 4 signature
&X-Amz-SignedHeaders=host
&media-encoding=flac
&sample-rate=16000
&session-id=sessionId
&enable-channel-identification=true
&number-of-channels=2
```

Parameter definitions can be found in the API Reference; parameters common to all AWS API operations are listed in the Common Parameters section.

For more information about WebSocket requests, see Setting up a WebSocket stream.

Multi-channel streaming output

The output of a streaming transcription is the same for HTTP/2 and WebSocket requests. The following is an example output.

```
{
    "resultId": "XXXXXX-XXXX-XXXX-XXXX-XXXXXXXXXX",
    "startTime": 0.11,
    "endTime": 0.66,
    "isPartial": false,
    "alternatives": [
        {
            "transcript": "Left.",
            "items": [
                {
                    "startTime": 0.11,
                    "endTime": 0.45,
                    "type": "pronunciation",
                    "content": "Left",
                    "vocabularyFilterMatch": false
                },
                {
                    "startTime": 0.45,
                    "endTime": 0.45,
                    "type": "punctuation",
                    "content": ".",
                    "vocabularyFilterMatch": false
                }
            ]
```

```
        }
    ],
    "channelId": "ch_0"
}
```

For each speech segment, there is a `channelId` flag that indicates which channel the speech belongs to.

Transcribing a medical dictation

You can use Amazon Transcribe Medical to transcribe clinician-dictated medical notes using either a batch transcription job or a real-time stream. Batch transcription jobs enable you to transcribe audio files. You specify the medical specialty of the clinician in your transcription job or stream to ensure that Amazon Transcribe Medical produces transcription results with the highest possible accuracy.

You can transcribe a medical dictation in the following specialties:

- Cardiology – available in streaming transcription only
- Neurology – available in streaming transcription only
- Oncology – available in streaming transcription only
- Primary Care – includes the following types of medical practice:
  - Family medicine
  - Internal medicine
  - Obstetrics and Gynecology (OB-GYN)
  - Pediatrics
- Radiology – available in streaming transcription only
- Urology – available in streaming transcription only

You can improve transcription accuracy by using custom vocabularies. For information on how medical custom vocabularies work, see Improving transcription accuracy with medical custom vocabularies.

By default, Amazon Transcribe Medical returns the transcription with the highest confidence level. If you'd like to configure it to return alternative transcriptions, see Generating alternative transcriptions.

For information about how numbers and medical measurements appear in the transcription output, see Transcribing numbers and Transcribing medical terms and measurements.

Topics

- Transcribing an audio file of a medical dictation
- Transcribing a medical dictation in a real-time stream

Transcribing an audio file of a medical dictation

Use a batch transcription job to transcribe audio files of medical conversations. You can use this to transcribe a clinician-patient dialogue. You can start a batch transcription job in either the `StartMedicalTranscriptionJob` API or the AWS Management Console.

When you start a medical transcription job with the `StartMedicalTranscriptionJob` API, you specify `PRIMARYCARE` as the value of the `Specialty` parameter.

AWS Management Console

To transcribe a clinician-patient dialogue (AWS Management Console)

To use the AWS Management Console to transcribe a clinician-patient dialogue, create a transcription job and choose **Conversation** for **Audio input type**.

1. Sign in to the AWS Management Console.
2. In the navigation pane, under Amazon Transcribe Medical, choose **Transcription jobs**.
3. Choose **Create job**.
4. On the **Specify job details** page, under **Job settings** , specify the following.
    a. **Name** – the name of the transcription job.
    b. **Audio input type** – **Dictation**
5. For the remaining fields, specify the Amazon S3 location of your audio file and where you want to store the output of your transcription job.
6. Choose **Next**.
7. Choose **Create**.

API

To transcribe a medical conversation using a batch transcription job (API)

- For the StartMedicalTranscriptionJob API, specify the following.

  a. For MedicalTranscriptionJobName, specify a name unique in your AWS account.

  b. For LanguageCode, specify the language code that corresponds to the language spoken in your audio file and the language of your vocabulary filter.

  c. In the MediaFileUri parameter of the Media object, specify the name of the audio file that you want to transcribe.

  d. For Specialty, specify the medical specialty of the clinician speaking in the audio file.

  e. For Type, specify DICTATION.

  f. For OutputBucketName, specify the Amazon S3 bucket to store the transcription results.

  The following is an example request that uses the AWS SDK for Python (Boto3) to transcribe a medical dictation of a clinician in the PRIMARYCARE specialty.

```
from __future__ import print_function
import time
import boto3
transcribe = boto3.client('transcribe')
job_name = "my-first-med-transcription-job"
job_uri = "s3://amzn-s3-demo-bucket/my-input-files/my-audio-file.flac"
transcribe.start_medical_transcription_job(
    MedicalTranscriptionJobName = job_name,
    Media = {
       'MediaFileUri': job_uri
    },
    OutputBucketName = 'amzn-s3-demo-bucket',
    OutputKey = 'my-output-files/',
    LanguageCode = 'en-US',
    Specialty = 'PRIMARYCARE',
    Type = 'DICTATION'
 )
while True:
    status = transcribe.get_medical_transcription_job(MedicalTranscriptionJobName =
 job_name)
```

```
    if status['MedicalTranscriptionJob']['TranscriptionJobStatus'] in ['COMPLETED',
 'FAILED']:
        break
    print("Not ready yet...")
    time.sleep(5)
print(status)
```

The following example code shows the transcription results of a medical dictation.

```
{
    "jobName": "dictation-medical-transcription-job",
    "accountId": "111122223333",
    "results": {
        "transcripts": [
            {
                "transcript": "... came for a follow up visit today..."
            }
        ],
        "items": [
            {
            ...
                "start_time": "4.85",
                "end_time": "5.12",
                "alternatives": [
                    {
                        "confidence": "1.0",
                        "content": "came"
                    }
                ],
                "type": "pronunciation"
            },
            {
                "start_time": "5.12",
                "end_time": "5.29",
                "alternatives": [
                    {
                        "confidence": "1.0",
                        "content": "for"
                    }
                ],
```

```
                "type": "pronunciation"
            },
            {
                "start_time": "5.29",
                "end_time": "5.33",
                "alternatives": [
                    {
                        "confidence": "0.9955",
                        "content": "a"
                    }
                ],
                "type": "pronunciation"
            },
            {
                "start_time": "5.33",
                "end_time": "5.66",
                "alternatives": [
                    {
                        "confidence": "0.9754",
                        "content": "follow"
                    }
                ],
                "type": "pronunciation"
            },
            {
                "start_time": "5.66",
                "end_time": "5.75",
                "alternatives": [
                    {
                        "confidence": "0.9754",
                        "content": "up"
                    }
                ],
                "type": "pronunciation"
            },
            {
                "start_time": "5.75",
                "end_time": "6.02",
                "alternatives": [
                    {
                        "confidence": "1.0",
                        "content": "visit"
                    }
                ]
```

```
                ...
    },
    "status": "COMPLETED"
}
```

AWS CLI

To enable speaker partitioning in a batch transcription job (AWS CLI)

- Run the following code.

```
aws transcribe start-medical-transcription-job \
--region us-west-2 \
--cli-input-json file://example-start-command.json
```

The following code shows the contents of `example-start-command.json`.

```
{
      "MedicalTranscriptionJobName": "my-first-med-transcription-job",
      "Media": {
      "MediaFileUri": "s3://amzn-s3-demo-bucket/my-input-files/my-audio-file.flac"
      },
      "OutputBucketName": "amzn-s3-demo-bucket",
      "OutputKey": "my-output-files/",
      "LanguageCode": "en-US",
      "Specialty": "PRIMARYCARE",
      "Type": "DICTATION"
}
```

Transcribing a medical dictation in a real-time stream

Use a WebSocket stream to transcribe a medical dictation as an audio stream. You can also use the AWS Management Console to transcribe speech that you or others speak directly into a microphone.

For an HTTP/2 or a WebSocket stream, you can transcribe audio in the following medical specialties:

- Cardiology
- Oncology
- Neurology
- Primary Care
- Radiology
- Urology

Each medical specialty includes many types of procedures and appointments. Clinicians therefore dictate many different types of notes. Use the following examples as guidance to help you specify the value of the `specialty` URI parameter of the WebSocket request, or the `Specialty` parameter of the `StartMedicalStreamTranscription` API:

- For a dictation after electrophysiology or echocardiogram procedure, choose `CARDIOLOGY`.
- For a dictation after a surgical oncology or radiation oncology procedure, choose `ONCOLOGY`.
- For a physician dictating notes indicating a diagnosis of encephalitis, choose `NEUROLOGY`.
- For a dictation of procedure notes to break up a bladder stone, choose `UROLOGY`.
- For a dictation of clinician notes after an internal medicine consultation, choose `PRIMARYCARE`.
- For a dictation of a physician communicating the findings of a CT scan, PET scan, MRI, or radiograph, choose `RADIOLOGY`.
- For a dictation of physician notes after a gynecology consultation, choose `PRIMARYCARE`.

To improve transcription accuracy of specific terms in a real-time stream, use a custom vocabulary. To enable a custom vocabulary, set the value of `vocabulary-name` to the name of the custom vocabulary you want to use.

Transcribing a dictation spoken into your microphone with the AWS Management Console

To use the AWS Management Console to transcribe streaming audio of a medical dictation, choose the option to transcribe a medical dictation, start the stream, and begin speaking into the microphone.

To transcribe streaming audio of a medical dictation (AWS Management Console)

1. Sign in to the AWS Management Console.
2. In the navigation pane, under Amazon Transcribe Medical, choose **Real-time transcription**.
3. Choose **Dictation**.
4. For **Medical specialty**, choose the medical specialty of the clinician speaking in the stream.
5. Choose **Start streaming**.
6. Speak into the microphone.

Transcribing a dictation in an HTTP/2 stream

To transcribe an HTTP/2 stream of a medical dictation, use the `StartMedicalStreamTranscription` API and specify the following:

- `LanguageCode` – The language code. The valid value is `en-US`
- `MediaEncoding` – The encoding used for the input audio. Valid values are `pcm`, `ogg-opus`, and `flac`.
- `Specialty` – The specialty of the medical professional.
- `Type` – `DICTATION`

For more information on setting up an HTTP/2 stream to transcribe a medical dictation, see Setting up an HTTP/2 stream.

Using a WebSocket streaming request to transcribe a medical dictation

To transcribe a medical dictation in a real-time stream using a WebSocket request, you create a presigned URI. This URI contains the information needed to set up the audio stream between your application and Amazon Transcribe Medical. For more information on creating WebSocket requests, see Setting up a WebSocket stream.

Use the following template to create your presigned URI.

```
GET wss://transcribestreaming.us-west-2.amazonaws.com:8443/medical-stream-
transcription-websocket
?language-code=languageCode
&X-Amz-Algorithm=AWS4-HMAC-SHA256
```

```
&X-Amz-Credential=AKIAIOSFODNN7EXAMPLE%2F20220208%2Fus-
west-2%2Ftranscribe%2Faws4_request
&X-Amz-Date=20220208T235959Z
&X-Amz-Expires=300
&X-Amz-Security-Token=security-token
&X-Amz-Signature=Signature Version 4 signature
&X-Amz-SignedHeaders=host
&media-encoding=flac
&sample-rate=16000
&session-id=sessionId
&specialty=medicalSpecialty
&type=DICTATION
&vocabulary-name=vocabularyName
&show-speaker-label=boolean
```

For more information on creating pre-signed URIs, see Setting up a WebSocket stream.

Improving transcription accuracy with medical custom vocabularies

To improve transcription accuracy in Amazon Transcribe Medical, create and use one or more medical custom vocabularies. A *custom vocabulary* is a collection of words or phrases that are domain-specific. This collection helps improve the performance of Amazon Transcribe Medical in transcribing those words or phrases.

You are responsible for the integrity of your own data when you use Amazon Transcribe Medical. Do not enter confidential information, personal information (PII), or protected health information (PHI), into a custom vocabulary.

For best results, create separate small custom vocabularies that each help transcribe a specific audio recording. You receive greater improvements in transcription accuracy than if you created one large custom vocabulary to use with all of your recordings.

By default, you can have up to 100 custom vocabularies in your AWS account. A custom vocabulary can't exceed 50 KB in size. For information on requesting an increase to the number of custom vocabularies that you can have in your AWS account, see AWS service quotas.

Custom vocabularies are available in US English (en-US).

Topics

- Creating a text file for your medical custom vocabulary
- Using a text file to create a medical custom vocabulary
- Transcribing an audio file using a medical custom vocabulary
- Transcribing a real-time stream using a medical custom vocabulary
- Character set for Amazon Transcribe Medical

Creating a text file for your medical custom vocabulary

To create a custom vocabulary, you create a text file that is in UTF-8 format. In this file, you create a four column table, with each column specifying a field. Each field tells Amazon Transcribe Medical either how the domain-specific terms are pronounced or how to display these terms in your transcriptions. You store the text file containing these fields in an Amazon S3 bucket.

Understanding how to format your text file

To create a medical custom vocabulary, you enter the column names as a header row. You enter the values for each column beneath the header row.

The following are the names of the four columns of the table:

- `Phrase` – column required, values required
- `IPA` – column required, values can be optional
- `SoundsLike` – column required, values can be optional
- `DisplayAs` – column required, values can be optional

When you create a custom vocabulary, make sure that you:

- Separate each column with a single Tab character. Amazon Transcribe throws an error message if you try to separate the columns with spaces or multiple Tab characters.
- Make sure that there's no trailing spaces or white space after each value within a column.

Make sure that the values that you enter for each column:

- Have fewer than 256 characters, including hyphens
- Use only characters from the allowed character set, see Character set for Amazon Transcribe Medical.

Entering values for the columns of the table

The following information shows you how to specify values for the four columns of the table:

- **`Phrase`** – The word or phrase that should be recognized. You must enter values in this column.

  If the entry is a phrase, separate the words with a hyphen (-). For example, enter **`cerebral autosomal dominant arteriopathy with subcortical infarcts and leukoencephalopathy`** as **`cerebral-autosomal-dominant-arteriopathy-with-subcortical-infarcts-and-leukoencephalopathy`**.

  Enter acronyms or other words whose letters should be pronounced individually as single letters followed by dots, such as **`D.N.A.`** or **`S.T.E.M.I.`**. To enter the plural form of an acronym, such as "STEMIs," separate the "s" from the acronym with a hyphen: "**`S.T.E.M.I-s`**" You can use either uppercase or lowercase letters for acronyms.

  The `Phrase` column is required. You can use any of the allowed characters for the input language. For allowed characters, see Character set for Amazon Transcribe Medical. If you don't specify the `DisplayAs` column, Amazon Transcribe Medical uses the contents of the `Phrase` column in the output file.

- **`IPA`** (column required, values can be optional) – To specify the pronunciation of a word or phrase, you can include characters in the International Phonetic Alphabet (IPA) in this column. The `IPA` column can't contain leading or trailing spaces, and you must use a single space to separate each phoneme in the input. For example, in English you would enter the phrase **`acute-respiratory-distress-syndrome`** as **`ə k j u t # # s p # # ə t # # i d # s t # # s s # n d # o# m`**. You would enter the phrase **`A.L.L.`** as **`e# # l # l`**.

  Even if you don't specify the contents of the `IPA` column, you must include a blank `IPA` column. If you include values in the `IPA` column, you can't provide values for the `SoundsLike` column.

  For a list of allowed IPA characters for a specific language, see Character set for Amazon Transcribe Medical. US English is the only language available in Amazon Transcribe Medical.

- **`SoundsLike`** (column required, values can be optional) – You can break a word or phrase down into smaller segments and provide a pronunciation for each segment using the standard orthography of the language to mimic the way that the word sounds. For example, you can provide pronunciation hints for the phrase **`cerebral-autosomal-dominant-arteriopathy-with-subcortical-infarcts-and-leukoencephalopathy`** like this:

sir-e-brul-aut-o-som-ul-dah-mi-nant-ar-ter-ri-o-pa-thy-with-sub-cor-ti-cul-in-farcts-and-lewk-o-en-ce-phul-ah-pu-thy. The hint for the phrase **atrioventricular-nodal-reentrant-tachycardia** would look like this: **ay-tree-o-ven-trick-u-lar-node-al-re-entr-ant-tack-ih-card-ia**. You separate each part of the hint with a hyphen (-).

Even if you don't provide values for the `SoundsLike` column, you must include a blank `SoundsLike` column. If you include values in the `SoundsLike` column, you can't provide values for the `IPA` column.

You can use any of the allowed characters for the input language. For the list of allowed characters, see Character set for Amazon Transcribe Medical.

- **`DisplayAs`** (column required, values can be optional) – Defines how the word or phrase looks when it's output. For example, if the word or phrase is **cerebral-autosomal-dominant-arteriopathy-with-subcortical-infarcts-and-leukoencephalopathy**, you can specify the display form as `cerebral autosomal dominant arteriopathy with subcortical infarcts and leukoencephalopathy`, so that the hyphen is not present. You can also specify `DisplayAs` as `CADASIL` if you'd like to show the acronym instead of the full term in the output.

  If you don't specify the `DisplayAs` column, Amazon Transcribe Medical uses the `Phrase` column from the input file in the output.

  You can use any UTF-8 character in the `DisplayAs` column.

You can include spaces only for the values in the `IPA` and `DisplayAs` columns.

To create the text file of your custom vocabulary, place each word or phrase in your text file on a separate line. Separate the columns with Tab characters. Include spaces only for values in the `IPA` and `DisplayAs` columns. Save the file with the extension `.txt` in an Amazon S3 bucket in the same AWS Region where you use Amazon Transcribe Medical to create your custom vocabulary.

Your custom vocabulary file supports both `LF` and `CRLF` line endings.

The following examples show text that you can use to create custom vocabularies. To create a custom vocabulary from these examples, copy an example into a text editor, replace `[TAB]` with a Tab character, and upload the saved text file to Amazon S3.

```
Phrase[TAB]IPA[TAB]SoundsLike[TAB]DisplayAs
```

```
acute-respiratory-distress-syndrome[TAB][TAB][TAB]acute respiratory distress syndrome
A.L.L.[TAB]e# # l # l[TAB][TAB]ALL
atrioventricular-nodal-reentrant-tachycardia[TAB][TAB]ay-tree-o-ven-trick-u-lar-node-
al-re-entr-ant-tack-ih-card-ia[TAB]
```

You can enter columns in any order. The following examples show other valid structures for the custom vocabulary input file.

```
Phrase[TAB]SoundsLike[TAB]IPA[TAB]DisplayAs
acute-respiratory-distress-syndrome[TAB][TAB][TAB]acute respiratory distress syndrome
A.L.L.[TAB][TAB]e# # l # l[TAB]ALL
atrioventricular-nodal-reentrant-tachycardia[TAB]ay-tree-o-ven-trick-u-lar-node-al-re-
entr-ant-tack-ih-card-ia[TAB][TAB]
```

```
DisplayAs[TAB]SoundsLike[TAB]IPA[TAB]Phrase
acute respiratory distress syndrome[TAB][TAB][TAB]acute-respiratory-distress-syndrome
ALL[TAB][TAB]e# # l # l[TAB]A.L.L.
[TAB]ay-tree-o-ven-trick-u-lar-node-al-re-entr-ant-tack-ih-card-ia[TAB]
[TAB]atrioventricular-nodal-reentrant-tachycardia
```

For reading ease, the following tables show the preceding examples more clearly in html format. They are meant only to illustrate the examples.

Phrase	IPA	SoundsLike	DisplayAs
acute-respiratory-distress-syndrome			acute respiratory distress syndrome
A.L.L.	eɪ ɛ l ɛ l		ALL
atrioventricular-nodal-reentrant-tachycardia		ay-tree-o-ven-trick-u-lar-node-al-re-entr-ant-tack-ih-card-ia	

Phrase	SoundsLike	IPA	DisplayAs
acute-respiratory-distress-syndrome			acute respiratory distress syndrome
atrioventricular-nodal-reentrant-tachycardia	ay-tree-o-ven-trick-u-lar-node-al-re-entr-ant-tack-ih-card-ia		
A.L.L.		eɪ ɛ l ɛ l	ALL

DisplayAs	SoundsLike	IPA	Phrase
acute respiratory distress syndrome			acute-respiratory-distress-syndrome
ALL		eɪ ɛ l ɛ l	A.L.L.
	ay-tree-o-ven-trick-u-lar-node-al-re-entr-ant-tack-ih-card-ia		atrioventricular-nodal-reentrant-tachycardia

Using a text file to create a medical custom vocabulary

To create a custom vocabulary, you must have prepared a text file that contains a collection a words or phrases. Amazon Transcribe Medical uses this text file to create a custom vocabulary that you can use to improve the transcription accuracy of those words or phrases. You can create a custom vocabulary using the `CreateMedicalVocabulary` API or the Amazon Transcribe Medical console.

AWS Management Console

To use the AWS Management Console to create a custom vocabulary, you provide the Amazon S3 URI of the text file containing your words or phrases.

1. Sign in to the AWS Management Console.
2. In the navigation pane, under Amazon Transcribe Medical, choose **Custom vocabulary**.

3. For **Name**, under **Vocabulary settings**, choose a name for your custom vocabulary.
4. Specify the location of your audio file or video file in Amazon S3:
   - For **Vocabulary input file location on S3** under **Vocabulary settings**, specify the Amazon S3 URI that identifies the text file you will use to create your custom vocabulary.
   - For **Vocabulary input file location in S3**, choose **Browse S3** to browse for the text file and choose it.
5. Choose **Create vocabulary**.

You can see the processing status of your custom vocabulary in the AWS Management Console.

API

To create a medical custom vocabulary (API)

- For the `StartTranscriptionJob` API, specify the following.
  a. For `LanguageCode`, specify `en-US`.
  b. For `VocabularyFileUri`, specify the Amazon S3 location of the text file that you use to define your custom vocabulary.
  c. For `VocabularyName`, specify a name for your custom vocabulary. The name you specify must be unique within your AWS account.

To see the processing status of your custom vocabulary, use the `GetMedicalVocabulary` API.

The following is an example request using the AWS SDK for Python (Boto3) to create a custom vocabulary.

```
from __future__ import print_function
import time
import boto3
transcribe = boto3.client('transcribe', 'us-west-2')
vocab_name = "my-first-vocabulary"
response = transcribe.create_medical_vocabulary(
    VocabularyName = job_name,
    VocabularyFileUri = 's3://amzn-s3-demo-bucket/my-vocabularies/my-vocabulary-
table.txt'
    LanguageCode = 'en-US',
```

```
  )

while True:
    status = transcribe.get_medical_vocabulary(VocabularyName = vocab_name)
    if status['VocabularyState'] in ['READY', 'FAILED']:
        break
    print("Not ready yet...")
    time.sleep(5)
print(status)
```

AWS CLI

To enable speaker partitioning in a batch transcription job (AWS CLI)

- Run the following code.

```
aws transcribe create-medical-vocabulary \
--vocabulary-name my-first-vocabulary \
--vocabulary-file-uri s3://amzn-s3-demo-bucket/my-vocabularies/my-vocabulary-
file.txt \
--language-code en-US
```

Transcribing an audio file using a medical custom vocabulary

Use the StartMedicalTranscriptionJob or the AWS Management Console to start a transcription job that uses a custom vocabulary to improve transcription accuracy.

AWS Management Console

1. Sign in to the AWS Management Console.
2. In the navigation pane, under Amazon Transcribe Medical, choose **Transcription jobs**.
3. Choose **Create job**.
4. On the **Specify job details** page, provide information about your transcription job.
5. Choose **Next**.
6. Under **Customization**, enable **Custom vocabulary**.
7. Under **Vocabulary selection**, choose a custom vocabulary.
8. Choose **Create**.

API

To enable speaker partitioning in an audio file using a batch transcription job (API)

- For the `StartMedicalTranscriptionJob` API, specify the following.

  a. For `MedicalTranscriptionJobName`, specify a name that is unique in your AWS account.

  b. For `LanguageCode`, specify the language code that corresponds to the language spoken in your audio file and the language of your vocabulary filter.

  c. For the `MediaFileUri` parameter of the `Media` object, specify the name of the audio file that you want to transcribe.

  d. For `Specialty`, specify the medical specialty of the clinician speaking in the audio file.

  e. For `Type`, specify whether the audio file is a conversation or a dictation.

  f. For `OutputBucketName`, specify the Amazon S3 bucket to store the transcription results.

  g. For the `Settings` object, specify the following.

  - `VocabularyName` – the name of your custom vocabulary.

The following request uses the AWS SDK for Python (Boto3) to start a batch transcription job with a custom vocabulary.

```
from __future__ import print_function
import time
import boto3
transcribe = boto3.client('transcribe', 'us-west-2')
job_name = "my-first-med-transcription-job"
job_uri = "s3://amzn-s3-demo-bucket/my-input-files/my-media-file.flac"
transcribe.start_medical_transcription_job(
   MedicalTranscriptionJobName = job_name,
   Media = {
       'MediaFileUri': job_uri
   },
   OutputBucketName = 'amzn-s3-demo-bucket',
   OutputKey = 'my-output-files/',
   LanguageCode = 'en-US',
   Specialty = 'PRIMARYCARE',
   Type = 'CONVERSATION',
```

```
   Settings = {
       'VocabularyName': 'example-med-custom-vocab'
       }
 )

while True:
   status = transcribe.get_medical_transcription_job(MedicalTranscriptionJobName =
 job_name)
   if status['MedicalTranscriptionJob']['TranscriptionJobStatus'] in ['COMPLETED',
 'FAILED']:
        break
    print("Not ready yet...")
    time.sleep(5)
print(status)
```

Transcribing a real-time stream using a medical custom vocabulary

To improve transcription accuracy in a real-time stream, you can use a custom vocabulary using either HTTP/2 or WebSocket streams. To start an HTTP/2 request, use the `StartMedicalStreamTranscription` API. You can use a custom vocabulary in real-time using either the AWS Management Console, the `StartMedicalStreamTranscription` API, or by using the WebSocket protocol.

Transcribing a dictation that is spoken into your Microphone (AWS Management Console)

To use the AWS Management Console to transcribe streaming audio of a medical dictation, choose the option to transcribe a medical dictation, start the stream, and begin speaking into the microphone.

To transcribe streaming audio of a medical dictation (AWS Management Console)

1. Sign in to the AWS Management Console.
2. In the navigation pane, under Amazon Transcribe Medical, choose **Real-time transcription**.
3. For **Medical specialty**, choose the medical specialty of the clinician speaking in the stream.
4. For **Audio input type**, choose either **Conversation** or **Dictation**.
5. For **Additional settings**, choose **Custom vocabulary**.

    - For **Vocabulary selection**, choose the custom vocabulary.

6. Choose **Start streaming**.
7. Speak into the microphone.

Enabling speaker partitioning in an HTTP/2 stream

The following is the syntax for the parameters of an HTTP/2 request.

```
POST /medical-stream-transcription HTTP/2
host: transcribestreaming.us-west-2.amazonaws.com
authorization: Generated value
x-amz-target: com.amazonaws.transcribe.Transcribe.StartMedicalStreamTranscription
x-amz-content-sha256: STREAMING-MED-AWS4-HMAC-SHA256-EVENTS
x-amz-date: 20220208T235959Z
x-amzn-transcribe-session-id: my-first-http2-med-stream
x-amzn-transcribe-language-code: en-US
x-amzn-transcribe-media-encoding: flac
x-amzn-transcribe-sample-rate: 16000
x-amzn-transcribe-vocabulary-name: my-first-med-vocab
x-amzn-transcribe-specialty: PRIMARYCARE
x-amzn-transcribe-type: CONVERSATION
x-amzn-transcribe-show-speaker-label: true
Content-type: application/vnd.amazon.eventstream
transfer-encoding: chunked
```

Parameter descriptions:

- **host**: Update the AWS Region ('us-west-2' in the preceding example) with the AWS Region you are calling. For a list of valid AWS Regions, see AWS Regions and Endpoints.
- **authorization**: This is a generated field. To learn more about creating a signature, see Signing AWS requests with Signature Version 4.
- **x-amz-target**: Don't alter this field; use the content shown in the preceding example.
- **x-amz-content-sha256**: This is a generated field. To learn more about calculating a signature, see Signing AWS requests with Signature Version 4.
- **x-amz-date**: The date and time the signature is created. The format is YYYYMMDDTHHMMSSZ, where YYYY=year, MM=month, DD=day, HH=hour, MM=minute, SS=seconds, and 'T' and 'Z' are fixed characters. For more information, refer to Handling Dates in Signature Version 4.
- **x-amzn-transcribe-session-id**: The name for your streaming session.

- **x-amzn-transcribe-language-code**: The encoding used for your input audio. Refer to `StartMedicalStreamTranscription` or Supported languages and language-specific features for a list of valid values.
- **x-amzn-transcribe-media-encoding**: The encoding used for your input audio. Valid values are `pcm`, `ogg-opus`, and `flac`.
- **x-amzn-transcribe-sample-rate**: The sample rate of the input audio (in Hertz). Amazon Transcribe supports a range from 8,000 Hz to 48,000 Hz. Low-quality audio, such as telephone audio, is typically around 8,000 Hz. High-quality audio typically ranges from 16,000 Hz to 48,000 Hz. Note that the sample rate you specify **must** match that of your audio.
- **x-amzn-transcribe-vocabulary-name**: The name of the vocabulary you want to use with your transcription.
- **x-amzn-transcribe-specialty**: The medical specialty being transcribed.
- **x-amzn-transcribe-type**: Choose whether this is a dictation or a conversation.
- **x-amzn-transcribe-show-speaker-label**: To enable diarization, this value must be `true`.
- **content-type**: Don't alter this field; use the content shown in the preceding example.

Enabling speaker partitioning in a WebSocket request

To partition speakers in WebSocket streams with the API, use the following format to create a pre-signed URI to start a WebSocket request and set `vocabulary-name` to the name of the custom vocabulary.

```
GET wss://transcribestreaming.us-west-2.amazonaws.com:8443/medical-stream-
transcription-websocket
?language-code=en-US
&X-Amz-Algorithm=AWS4-HMAC-SHA256
&X-Amz-Credential=AKIAIOSFODNN7EXAMPLE%2F20220208%2Fus-
west-2%2Ftranscribe%2Faws4_request
&X-Amz-Date=20220208T235959Z
&X-Amz-Expires=300
&X-Amz-Security-Token=security-token
&X-Amz-Signature=Signature Version 4 signature
&X-Amz-SignedHeaders=host
&media-encoding=flac
&sample-rate=16000
&session-id=sessionId
&specialty=medicalSpecialty
&type=CONVERSATION
```

```
&vocabulary-name=vocabularyName
&show-speaker-label=boolean
```

Character set for Amazon Transcribe Medical

To use custom vocabularies in Amazon Transcribe Medical, use the following character set.

English character set

For English custom vocabularies, you can use the following characters in the `Phrase` and `SoundsLike` columns:

- a - z
- A - Z
- ' (apostrophe)
- - (hyphen)
- . (period)

You can use the following International Phonetic Alphabet (IPA) characters in the `IPA` column of the vocabulary input file.

Character	Code	Character	Code
aʊ	0061 028A	w	0077
aɪ	0061 026A	z	007A
b	0062	æ	00E6
d	0064	ð	00F0
eɪ	0065 026A	ŋ	014B
f	0066	ɑ	0251
g	0067	ɔ	0254
h	0068	ɔɪ	0254 026A

Character	Code	Character	Code
i	0069	ə	0259
j	006A	ɛ	025B
k	006B	ɝ	025D
l	006C	g	0261
l̩	006C 0329	ɪ	026A
m	006D	ɹ	0279
n	006E	ʃ	0283
n̩	006E 0329	ʊ	028A
oʊ	006F 028A	ʌ	028C
p	0070	ʍ	028D
s	0073	ʒ	0292
t	0074	ʤ	02A4
u	0075	ʧ	02A7
v	0076	θ	03B8

Identifying personal health information (PHI) in a transcription

Use *Personal Health Information Identification* to label personal health information (PHI) in your transcription results. By reviewing labels, you can find PHI that could be used to identify a patient.

You can identify PHI using either a real-time stream or batch transcription job.

You can use your own post-processing to redact the PHI identified in the transcription output.

Use Personal Health Information Identification to identify the following types of PHI:

- Personal PHI:
  - Names – Full name or last name and initial
  - Gender
  - Age
  - Phone numbers
  - Dates (not including the year) that directly relate to the patient
  - Email addresses
- Geographic PHI:
  - Physical address
  - Zip code
  - Name of medical center or practice
- Account PHI:
  - Fax numbers
  - Social security numbers (SSNs)
  - Health insurance beneficiary numbers
  - Account numbers
  - Certificate or license numbers
- Vehicle PHI:
  - Vehicle identification number (VIN)
  - License plate number
- Other PHI:
  - Web Uniform Resource Location (URL)
  - Internet Protocol (IP) address numbers

Amazon Transcribe Medical is a Health Insurance Portability and Accountability Act of 1996 (HIPAA) eligible service. For more information, see Amazon Transcribe Medical. For information about identifying PHI in an audio file, see Identifying PHI in an audio file. For information about identifying PHI in a stream, see Identifying PHI in a real-time stream.

Topics

- Identifying PHI in an audio file
- Identifying PHI in a real-time stream

Identifying PHI in an audio file

Use a batch transcription job to transcribe audio files and identify the personal health information (PHI) within them. When you activate Personal Health Information (PHI) Identification, Amazon Transcribe Medical labels the PHI that it identified in the transcription results. For information about the PHI that Amazon Transcribe Medical can identify, see Identifying personal health information (PHI) in a transcription.

You can start a batch transcription job using either the `StartMedicalTranscriptionJob` API or the AWS Management Console.

AWS Management Console

To use the AWS Management Console to transcribe a clinician-patient dialogue, create a transcription job and choose **Conversation** for **Audio input type**.

To transcribe an audio file and identify its PHI (AWS Management Console)

1. Sign in to the AWS Management Console.
2. In the navigation pane, under Amazon Transcribe Medical, choose **Transcription jobs**.
3. Choose **Create job**.
4. On the **Specify job details** page, under **Job settings** , specify the following.
    a. **Name** – The name of the transcription job that is unique to your AWS account.
    b. **Audio input type** – **Conversation** or **Dictation**.
5. For the remaining fields, specify the Amazon S3 location of your audio file and where you want to store the output of your transcription job.
6. Choose **Next**.
7. Under **Audio settings**, choose **PHI Identification**.
8. Choose **Create**.

API

To transcribe an audio file and identify its PHI using a batch transcription job (API)

- For the `StartMedicalTranscriptionJob` API, specify the following.

a. For `MedicalTranscriptionJobName`, specify a name that is unique to your AWS account.

b. For `LanguageCode`, specify the language code that corresponds to the language spoken in your audio file.

c. For the `MediaFileUri` parameter of the `Media` object, specify the name of the audio file that you want to transcribe.

d. For `Specialty`, specify the medical specialty of the clinician speaking in the audio file as `PRIMARYCARE`.

e. For `Type`, specify either `CONVERSATION` or `DICTATION`.

f. For `OutputBucketName`, specify the Amazon S3 bucket where you want to store the transcription results.

The following is an example request that uses the AWS SDK for Python (Boto3) to transcribe an audio file and identify the PHI of a patient.

```
from __future__ import print_function
import time
import boto3
transcribe = boto3.client('transcribe')
job_name = "my-first-transcription-job"
job_uri = "s3://amzn-s3-demo-bucket/my-input-files/my-audio-file.flac"
transcribe.start_medical_transcription_job(
      MedicalTranscriptionJobName = job_name,
      Media = {'MediaFileUri': job_uri},
      LanguageCode = 'en-US',
      ContentIdentificationType = 'PHI',
      Specialty = 'PRIMARYCARE',
      Type = 'type', # Specify 'CONVERSATION' for a medical conversation. Specify
 'DICTATION' for a medical dictation.
      OutputBucketName = 'amzn-s3-demo-bucket'
  )
while True:
    status = transcribe.get_medical_transcription_job(MedicalTranscriptionJobName =
 job_name)
    if status['MedicalTranscriptionJob']['TranscriptionJobStatus'] in ['COMPLETED',
 'FAILED']:
        break
```

```
    print("Not ready yet...")
    time.sleep(5)
print(status)
```

The following example code shows the transcription results with patient PHI identified.

```
{
    "jobName": "my-medical-transcription-job-name",
    "accountId": "111122223333",
    "results": {
        "transcripts": [{
            "transcript": "The patient's name is Bertrand."
        }],
        "items": [{
                "id": 0,
            "start_time": "0.0",
            "end_time": "0.37",
            "alternatives": [{
                "confidence": "0.9993",
                "content": "The"
            }],
            "type": "pronunciation"
        }, {
                "id": 1,
            "start_time": "0.37",
            "end_time": "0.44",
            "alternatives": [{
                "confidence": "0.9981",
                "content": "patient's"
            }],
            "type": "pronunciation"
        }, {
                "id": 2,
            "start_time": "0.44",
            "end_time": "0.52",
            "alternatives": [{
                "confidence": "1.0",
                "content": "name"
            }],
            "type": "pronunciation"
```

```
        }, {
                "id": 3,
            "start_time": "0.52",
            "end_time": "0.92",
            "alternatives": [{
                "confidence": "1.0",
                "content": "is"
            }],
            "type": "pronunciation"
        }, {
                "id": 4,
            "start_time": "0.92",
            "end_time": "0.9989",
            "alternatives": [{
                "confidence": "1.0",
                "content": "Bertrand"
            }],
            "type": "pronunciation"
        }, {
                "id": 5,
            "alternatives": [{
                "confidence": "0.0",
                "content": "."
            }],
            "type": "punctuation"
        }],
        "entities": [{
            "content": "Bertrand",
            "category": "PHI*-Personal*",
            "startTime": 0.92,
            "endTime": 1.2,
            "confidence": 0.9989
        }],
        "audio_segments": [
            {
                "id": 0,
                "transcript": "The patient's name is Bertrand.",
                "start_time": "0.0",
                "end_time": "0.9989",
                "items": [
                    0,
                    1,
                    2,
                    3,
```

```
                    4,
                    5
                ]
            }
        ]
    },
    "status": "COMPLETED"
}
```

AWS CLI

To transcribe an audio file and identify PHI using a batch transcription job (AWS CLI)

- Run the following code.

```
aws transcribe start-medical-transcription-job \
--medical-transcription-job-name my-medical-transcription-job-name\
--language-code en-US \
--media MediaFileUri="s3://amzn-s3-demo-bucket/my-input-files/my-audio-file.flac" \
--output-bucket-name amzn-s3-demo-bucket \
--specialty PRIMARYCARE \
--type type \ # Choose CONVERSATION to transcribe a medical conversation.
 Choose DICTATION to transcribe a medical dictation.
--content-identification-type PHI
```

Identifying PHI in a real-time stream

You can identify Personal Health Information (PHI) in either HTTP/2 or WebSocket streams. When you activate PHI Identification, Amazon Transcribe Medical labels the PHI that it identifies in the transcription results. For information about the PHI that Amazon Transcribe Medical can identify, see Identifying personal health information (PHI) in a transcription.

Identifying PHI in a dictation that is spoken into your microphone

To use the AWS Management Console to transcribe the speech picked up by your microphone and identify any PHI, choose **Dictation** as the audio input type, start the stream, and begin speaking into the microphone on your computer.

To identify PHI in a dictation using the AWS Management Console

1. Sign in to the AWS Management Console.
2. In the navigation pane, choose **Real-time transcription**.
3. For **Audio input type**, choose **Dictation**.
4. For **Additional settings**, choose **PHI identification**.
5. Choose **Start streaming** and speak into the microphone.
6. Choose **Stop streaming** to end the dictation.

Identifying PHI in an HTTP/2 stream

To start an HTTP/2 stream with PHI Identification activated, use the `StartMedicalStreamTranscription` API and specify the following:

- For `LanguageCode`, specify the language code for the language spoken in the stream. For US English, specify `en-US`.
- For `MediaSampleHertz`, specify the sample rate of the audio.
- For `content-identification-type`, specify `PHI`.

Identifying PHI in a WebSocket stream

To a start a WebSocket stream with PHI Identification activated, use the following format to create a presigned URL.

```
GET wss://transcribestreaming.us-west-2.amazonaws.com:8443/medical-stream-
transcription-websocket?
&X-Amz-Algorithm=AWS4-HMAC-SHA256
&X-Amz-Credential=AKIAIOSFODNN7EXAMPLE%2F20220208%2Fus-
west-2%2Ftranscribe%2Faws4_request
&X-Amz-Date=20220208T235959Z
&X-Amz-Expires=300
```

```
&X-Amz-Security-Token=security-token
&X-Amz-Signature=Signature Version 4 signature
&X-Amz-SignedHeaders=host
&language-code=en-US
&media-encoding=flac
&sample-rate=16000
&specialty=medical-specialty
&content-identification-type=PHI
```

Parameter definitions can be found in the API Reference; parameters common to all AWS API operations are listed in the Common Parameters section.

Generating alternative transcriptions

When you use Amazon Transcribe Medical, you get the transcription that has the highest confidence level. However, you can configure Amazon Transcribe Medical to return additional transcriptions with lower confidence levels.

Use alternative transcriptions to see different interpretations of the transcribed audio. For example, in an application that enables a person to review the transcription, you can present the alternative transcriptions for the person to choose from.

You can generate alternative transcriptions with the AWS Management Console or the `StartMedicalTranscriptionJob` API.

AWS Management Console

To use the AWS Management Console to generate alternative transcriptions, you enable alternative results when you configure your job.

1. Sign in to the AWS Management Console.
2. In the navigation pane, under Amazon Transcribe Medical, choose **Transcription jobs**.
3. Choose **Create job**.
4. On the **Specify job details** page, provide information about your transcription job.
5. Choose **Next**.
6. Enable **Alternative results**.
7. For **Maximum alternatives**, enter an integer value between 2 and 10, for the maximum number of alternative transcriptions you want in the output.

8. Choose **Create**.

API

To separate text per speaker in an audio file using a batch transcription job (API)

- For the `StartMedicalTranscriptionJob` API, specify the following.

  a. For `MedicalTranscriptionJobName`, specify a name that is unique in your AWS account.

  b. For `LanguageCode`, specify the language code that corresponds to the language spoken in your audio file and the language of your vocabulary filter.

  c. In the `MediaFileUri` parameter of the `Media` object, specify the location of the audio file you want to transcribe.

  d. For `Specialty`, specify the medical specialty of the clinician speaking in the audio file.

  e. For `Type`, specify whether you're transcribing a medical conversation or a dictation.

  f. For `OutputBucketName`, specify the Amazon S3 bucket to store the transcription results.

  g. For the `Settings` object, specify the following.

     i. `ShowAlternatives` – `true`.

     ii. `MaxAlternatives` - An integer between 2 and 10 to indicate the number of alternative transcriptions you want in the transcription output.

The following request uses the AWS SDK for Python (Boto3) to start a transcription job that generates up to two alternative transcriptions.

```
from __future__ import print_function
import time
import boto3
transcribe = boto3.client('transcribe', 'us-west-2')
job_name = "my-first-transcription-job"
job_uri = s3://amzn-s3-demo-bucket/my-input-files/my-audio-file.flac
transcribe.start_medical_transcription_job(
    MedicalTranscriptionJobName = job_name,
    Media = {
        'MediaFileUri': job_uri
    },
```

```
    OutputBucketName = 'amzn-s3-demo-bucket',
    OutputKey = 'my-output-files/',
    LanguageCode = 'en-US',
    Specialty = 'PRIMARYCARE',
    Type = 'CONVERSATION',
    Settings = {
        'ShowAlternatives': True,
        'MaxAlternatives': 2
    }
)

while True:
   status = transcribe.get_medical_transcription_job(MedicalTranscriptionJobName =
 job_name)
   if status['MedicalTranscriptionJob']['TranscriptionJobStatus'] in ['COMPLETED',
 'FAILED']:
       break
   print("Not ready yet...")
   time.sleep(5)
print(status)
```

AWS CLI

To transcribe an audio file of a conversation between a primary care clinician and a patient in an audio file (AWS CLI)

- Run the following code.

```
aws transcribe start-transcription-job \
--cli-input-json file://filepath/example-start-command.json
```

The following code shows the contents of `example-start-command.json`.

```
{
      "MedicalTranscriptionJobName": "my-first-transcription-job",
      "LanguageCode": "en-US",
      "Specialty": "PRIMARYCARE",
```

```
      "Type": "CONVERSATION",
      "OutputBucketName":"amzn-s3-demo-bucket",
      "Media": {
          "MediaFileUri": "s3://amzn-s3-demo-bucket/my-input-files/my-audio-
file.flac"
        },
      "Settings":{
          "ShowAlternatives": true,
          "MaxAlternatives": 2
        }
}
```

Amazon Transcribe Medical and interface VPC endpoints (AWS PrivateLink)

You can establish a private connection between your VPC and Amazon Transcribe Medical by creating an *interface VPC endpoint*. Interface endpoints are powered by AWS PrivateLink, a technology that enables you to privately access Amazon Transcribe Medical APIs without an internet gateway, NAT device, VPN connection, or Direct Connect connection. Instances in your VPC don't need public IP addresses to communicate with Amazon Transcribe Medical APIs. Traffic between your VPC and Amazon Transcribe Medical does not leave the Amazon network.

Each interface endpoint is represented by one or more Elastic Network Interfaces in your subnets.

For more information, see Interface VPC endpoints (AWS PrivateLink) in the *Amazon VPC User Guide*.

Considerations for Amazon Transcribe Medical VPC endpoints

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

Amazon Transcribe Medical supports making calls to all of its API actions from your VPC.

Creating an interface VPC endpoint for Amazon Transcribe Medical

You can create a VPC endpoint for the Amazon Transcribe Medical service using either the AWS Management Console or the AWS CLI. For more information, see Creating an interface endpoint in the *Amazon VPC User Guide*.

For batch transcription in Amazon Transcribe Medical, create a VPC endpoint using the following service name:

- com.amazonaws.*us-west-2*.transcribe

For streaming transcription in Amazon Transcribe Medical, create a VPC endpoint using the following service name:

- com.amazonaws.*us-west-2*.transcribestreaming

If you enable private DNS for the endpoint, you can make API requests to Amazon Transcribe Medical using its default DNS name for the AWS Region, for example, `transcribestreaming.us-east-2.amazonaws.com`.

For more information, see Accessing a service through an interface endpoint in the *Amazon VPC User Guide*.

Creating a VPC endpoint policy for Amazon Transcribe Medical streaming

You can attach an endpoint policy to your VPC endpoint that controls access to Amazon Transcribe Medical. The policy specifies the following information:

- The principal that can perform actions.
- The actions that can be performed.
- The resources on which actions can be performed.

For more information, see Controlling access to services with VPC endpoints in the *Amazon VPC User Guide*.

Example: VPC endpoint policy for Amazon Transcribe Medical streaming transcription actions

The following is an example of an endpoint policy for streaming transcription in Amazon Transcribe Medical. When attached to an endpoint, this policy grants access to the listed Amazon Transcribe Medical actions for all principals on all resources.

```
{
   "Statement":[
```

```
      {
         "Principal":"*",
         "Effect":"Allow",
         "Action":[
            "transcribe:StartMedicalStreamTranscription"
         ],
         "Resource":"*"
      }
   ]
}
```

Example: VPC endpoint policy for Amazon Transcribe Medical batch transcription actions

The following is an example of an endpoint policy for batch transcription in Amazon Transcribe Medical. When attached to an endpoint, this policy grants access to the listed Amazon Transcribe Medical actions for all principals on all resources.

```
{
   "Statement":[
      {
         "Principal":"*",
         "Effect":"Allow",
         "Action":[
            "transcribe:StartMedicalTranscriptionJob"
         ],
         "Resource":"*"
      }
   ]
}
```

Shared subnets

You cannot create, describe, modify, or delete VPC endpoints in subnets that are shared with you. However, you can use the VPC endpoints in subnets that are shared with you. For information about VPC sharing, see Share your VPC with other accounts in the Amazon Virtual Private Cloud guide.

AWS HealthScribe

Introducing Amazon Connect Health Ambient agent

You can now use the ambient agent to power your healthcare workflow. For more details, see Amazon Connect Health Ambient documentation.

AWS HealthScribe is a HIPAA-eligible machine learning (ML) capability that combines speech recognition and generative AI to transcribe patient-clinician conversations and generate easy-to-review clinical notes. AWS HealthScribe helps healthcare software vendors build clinical applications that reduce the documentation burden and improve consultation experience. The service automatically provides rich conversation transcripts, identifies speaker roles, classifies dialogues, extracts medical terms, and generates preliminary clinical notes. AWS HealthScribe combines these capabilities to remove the need to integrate and optimize separate AI services, enabling you to expedite implementation.

Common use cases:

- **Reduce documentation time** — Enable clinicians to quickly complete clinical documentation with AI-generated clinical notes that are easy to review, adjust, and finalize in your application.
- **Boost medical scribe efficiency** — Equip medical scribes with AI-generated transcript and clinical notes, along with the consultation audio, to expedite documentation turn-around time.
- **Efficient patient visit recap** — Create an experience that enables users to quickly recollect key highlights of their conversation in your application.

Important

The results produced by AWS HealthScribe are probabilistic and may not always be accurate due to various factors, including audio quality, background noise, speaker clarity, the complexity of medical terminology, context-specific language nuances, and the nature of machine learning and generative AI. AWS HealthScribe is designed to be used in an assistive role for clinicians and medical scribes. AWS HealthScribe output should only be used in patient care scenarios, including, but not limited to as part of Electronic Health Records, after review for accuracy and imposition of sound medical judgment by trained medical professionals. AWS HealthScribe output is not a substitute for professional medical advice,

diagnosis, or treatment, and is not intended to cure, treat, mitigate, prevent, or diagnose any disease or health condition.

Topics

- Security
- Service availability
- Technical requirements
- Supported Medical Specialties
- Workflows
- AWS HealthScribe Transcript file
- AWS HealthScribe Clinical Documentation file
- AWS HealthScribe transcription jobs
- AWS HealthScribe streaming
- Data Encryption at rest for AWS HealthScribe

Security

AWS HealthScribe operates under a shared responsibility model, whereby AWS is responsible for protecting the infrastructure that runs AWS HealthScribe and you are responsible for managing your data. For more information, see Shared Responsibility Model.

By default, AWS HealthScribe provides encryption at rest to protect sensitive customer data using Amazon S3-managed keys. When you create AWS HealthScribe transcription job or start a stream, you can specify a customer managed key. This adds a second layer of encryption. For more information, see Data Encryption at rest for AWS HealthScribe.

Service availability

AWS HealthScribe is available in the US East (N. Virginia) region.

Technical requirements

- **Supported Language:** US English (en-US)

- **Recommended Audio Format:** Lossless audio (such as FLAC or WAV)
- **Encoding:** PCM 16-bit
- **Sample Rate:** 16,000 Hz or higher

Supported Medical Specialties

AWS HealthScribe currently supports the following specialties:

- Allergy Immunology
- Cardiology
- Dermatology
- Endocrinology
- Gastroenterology
- Hematology/Oncology
- Infectious Disease
- Nephrology
- Neurology
- OBGYN
- Oncology
- Ophthalmology
- Orthopedics
- Otolaryngology
- Pain Medicine
- Pediatrics
- Primary Care
- Psychiatry
- Pulmonology
- Rheumatology
- Surgery
- Urology

Workflows

AWS HealthScribe workflows include transcription jobs and streaming. After you run a transcription job or complete a stream, AWS HealthScribe generates a transcript file, with turn-by-turn transcription output and insights for each conversation turn. Also it generates a clinical documentation file, with summaries and evidence links. For more information see AWS HealthScribe Transcript file and AWS HealthScribe Clinical Documentation file.

- **Transcription jobs** – With transcription jobs, AWS HealthScribe analyzes completed medical consultation media files from an Amazon S3 bucket. The following are API operations specific to AWS HealthScribe transcription jobs.
  - StartMedicalScribeJob
  - ListMedicalScribeJobs
  - GetMedicalScribeJob
  - DeleteMedicalScribeJob

  For more information, including code examples, see AWS HealthScribe transcription jobs.
- **Streaming** – AWS HealthScribe streaming is a real-time HTTP2 based bi-directional service that accepts audio stream on one channel and vends an audio transcription on the other channel.

  The following are API operations specific to AWS HealthScribe streaming:
  - StartMedicalScribeStream
  - GetMedicalScribeStream

  For more information, including code examples, see AWS HealthScribe streaming.

AWS HealthScribe Transcript file

In the transcript file, in addition to standard turn-by-turn transcription output with word level timestamps, AWS HealthScribe provides you with:

- **Participant role detection** so you can distinguish the patients from the clinicians in the conversation transcript.

- **Transcript sectioning**, which categorizes transcript dialogues based on their clinical relevance like small talk, subjective, objective, etc. This can be used to show specific portions of the transcript.
- **Clinical entities**, which includes structured information like medications, medical conditions, and treatments mentioned in the conversation.

In addition, the following insights are provided for each conversation turn:

- **Participant role** — Each participant is labeled as either a clinician or a patient. If a conversation has more than one participant in each category, each participant is assigned a number. For example, `CLINICIAN_0`, `CLINICIAN_1` and `PATIENT_0`, `PATIENT_1`.
- **Section** — Each dialogue turn is assigned to one of four possible sections based on the content identified.
  - **Subjective** — Information provided by the patient about their health concerns.
  - **Objective** — Information observed by the clinician through physical exam, lab, imaging, or diagnostic tests.
  - **Assessment and Plan** — Information that relates to the doctor's assessment and treatment plan.
  - **Visit Flow Management** — Information related to small talk or transitions.
- **Insights** — Extract clinically relevant entities (`ClinicalEntity`) present in the conversation. AWS HealthScribe detects all clinical entities supported by Amazon Comprehend Medical.

For an example of a transcript from a transcription job, see the transcript output in Transcription job output examples. For an example of a transcript from streaming, see the transcript output in Streaming transcription output examples.

AWS HealthScribe Clinical Documentation file

AWS HealthScribe can use one of the following templates for the clinical note summary. The default is `HISTORY_AND_PHYSICAL`.

- HISTORY_AND_PHYSICAL: Provides summaries for key sections of the clinical documentation. Examples of sections include Chief Complaint, History of Present Illness, Review of Systems, Past Medical History, Assessment, and Plan.

- GIRPP: Provides summaries based on the patients progress toward goals. Examples of sections include Goal, Intervention, Response, Progress, and Plan.
- BIRP: Focuses on the patient's behavioral patterns and responses. Examples of sections include Behavior, Intervention, Response, and Plan.
- SIRP: Emphasizes the situational context of therapy. Examples of sections include Situation, Intervention, Response, and Plan.
- DAP: Provides a simplified format for clinical documentation. Examples of sections include Data, Assessment, and Plan.
- BH_SOAP: Behavioral health focused documentation format. Examples of sections include Subjective, Objective, Assessment, and Plan.
- PH_SOAP: Physical health focused documentation format. Examples of sections include Subjective, Objective, Assessment, and Plan.

To specify what template to use, do the following:

- For transcription jobs, specify the template to use in the `NoteTemplate` of the ClinicalNoteGenerationSettings of the `Settings` of your StartMedicalScribeJob API operation.
- For streaming, you specify the template to use in the `NoteTemplate` of the ClinicalNoteGenerationSettings of the `PostStreamAnalyticsSettings` of your MedicalScribeConfigurationEvent.

Medical Scribe context

The `MedicalScribeContext` object contains contextual information used to customize clinical note generation with patient-specific details.

The `MedicalScribeContext` object includes the following components:

- **PatientContext** - Contains patient-specific information used to customize clinical note generation. This includes:
  - **Pronouns** - The patient's preferred pronouns that you want to provide as context for clinical note generation. These pronouns are used when referring to the patient in the generated clinical output.

For information on how to set up the medical scribe context, see MedicalScribeContext.

Topics

- HISTORY_AND_PHYSICAL template sections
- GIRPP template sections
- BIRP template sections
- SIRP template sections
- DAP template sections
- BH_SOAP template sections
- PH_SOAP template sections

HISTORY_AND_PHYSICAL template sections

The HISTORY_AND_PHYSICAL insights template includes the following sections.

Section	Description
CHIEF COMPLAINT	Brief description for the patient's reason for visiting the clinician.
HISTORY OF PRESENT ILLNESS	Notes that provide information on patient's illness, including reference to severity, onset, timing of symptoms, current treatments, and the affected areas.
REVIEW OF SYSTEMS	Patient-reported evaluation of symptoms across different body systems.
PAST MEDICAL HISTORY	Details a patient's previous medical conditions, surgeries, and treatments.
ASSESSMENT	Notes that provide information on clinician's assessment of patient's health.
PLAN	Notes that reference any medical treatments, lifestyle adjustments, and further appointme nts.

Section	Description
PHYSICAL_EXAMINATION	Documentation of clinician's findings from physical examination of patient's body systems and vital signs.
PAST_FAMILY_HISTORY	Information about health conditions that run in the patient's family.
PAST_SOCIAL_HISTORY	Details about patient's social life, habits, occupation, and environmental factors affecting health.
DIAGNOSTIC_TESTING	Results and interpretation of laboratory tests, imaging studies, and other diagnostic procedures.

Each sentence present in the `Summary` includes `EvidenceLinks` that provide the `SegmentId` for the relevant dialogues in the transcript that were summarized. This helps users validate accuracy of the summary in your application. Like explainability, providing traceability and transparency for AI-generated insights is consistent with Responsible AI principles. Providing these references along with the summary notes to clinicians or medical scribes helps foster trust and encourage the safe use of AI in the clinical settings.

For an example of a Clinical Documentation file from a transcription job, see the a Clinical Documentation file example in Transcription job output examples. For an example of a Clinical Documentation file from streaming, see the a Clinical Documentation file example in Streaming transcription output examples.

GIRPP template sections

The GIRPP insights template includes the following sections.

Section	Description
Goal	The identified problem, challenge, or behavior that needs to be addressed through treatment .

Section	Description
Intervention	The specific treatment, method, or technique used by the clinician to help the patient address the identified goal.
Response	How the patient responded to the intervent ion, including their participation level, reactions, and feedback.
Progress	The clinician's assessment of movement toward treatment goals, including observati ons of patient improvement or barriers.
Plan	The next steps in treatment, including future interventions, homework assignments, and referrals.

BIRP template sections

The BIRP insights template includes the following sections.

Section	Description
Behavior	The problems that the patient presents and their response to the treatment.
Intervention	The specific treatment, method, or technique used by the clinician to help the patient address the identified goal.
Response	How the patient responded to the intervent ion, including their participation level, reactions, and feedback.
Plan	The next steps in treatment, including future interventions, homework assignments, and referrals.

SIRP template sections

The SIRP insights template includes the following sections.

Section	Description
Situation	The problem that the patient presents and their goal for seeking therapy.
Intervention	The specific treatment, method, or technique used by the clinician to help the patient address the identified goal.
Response	How the patient responded to the intervent ion, including their participation level, reactions, and feedback.
Plan	The next steps in treatment, including future interventions, homework assignments, and referrals.

DAP template sections

The DAP insights template includes the following sections.

Section	Description
Data	The patient's reasons for seeking treatment and information about the patient.
Assessment	The clinician's diagnosis of the patient's situation.
Plan	The next steps in treatment, including future interventions, homework assignments, and referrals.

BH_SOAP template sections

The BH_SOAP insights template includes the following sections.

Section	Description
Subjective	The client's goals for treatment and their experiences, as well as existing and past issues.
Objective	Data and facts about the client.
Assessment	The clinician's diagnosis of the patient's situation.
Plan	The next steps in treatment, including future interventions, homework assignments, and referrals.

PH_SOAP template sections

The PH_SOAP insights template includes the following sections.

Section	Description
Subjective	The client's goals for treatment and their experiences, as well as existing and past issues.
Objective	Data and facts about the client.
Assessment	The clinician's diagnosis of the patient's situation.
Plan	The next steps in treatment, including future interventions, homework assignments, and referrals.

AWS HealthScribe transcription jobs

An AWS HealthScribe transcription job processes media files from an Amazon S3 bucket. When it processes a media file, it transcribes patient-clinician conversations and analyzes medical consultation to produces two JSON output files: a transcript file and a clinical documentation file.

The following are API operations specific to AWS HealthScribe transcription jobs:

- StartMedicalScribeJob
- ListMedicalScribeJobs
- GetMedicalScribeJob
- DeleteMedicalScribeJob

Starting an AWS HealthScribe transcription job

You can start an AWS HealthScribe job using the AWS CLI or AWS SDKs.

AWS CLI

This example uses the start-medical-scribe-job command. For more information, see StartMedicalScribeJob.

```
aws transcribe start-medical-scribe-job \
--region us-west-2 \
--medical-scribe-job-name my-first-medical-scribe-job \
--media MediaFileUri=s3://amzn-s3-demo-bucket/my-input-files/my-media-file.flac \
--output-bucket-name amzn-s3-demo-bucket \
--DataAccessRoleArn=arn:aws:iam::111122223333:role/ExampleRole \
--settings ShowSpeakerLabels=false,ChannelIdentification=true \
--channel-definitions ChannelId=0,ParticipantRole=CLINICIAN
 ChannelId=1,ParticipantRole=PATIENT
```

Here is another example using the start-medical-scribe-job command, and a request body with additional settings.

```
aws transcribe start-medical-scribe-job \
--region us-west-2 \
```

```
--cli-input-json file://filepath/my-first-medical-scribe-job.json
```

The file `my-first-medical-scribe-job.json` contains the following request body.

```
{
  "MedicalScribeJobName": "my-first-medical-scribe-job",
  "Media": {
    "MediaFileUri": "s3://amzn-s3-demo-bucket/my-input-files/my-media-file.flac"
   },
  "OutputBucketName": "amzn-s3-demo-bucket",
  "DataAccessRoleArn": "arn:aws:iam::111122223333:role/ExampleRole",
  "Settings": {
    "ShowSpeakerLabels": false,
    "ChannelIdentification": true
  },
  "ChannelDefinitions": [
    {
      "ChannelId": 0,
      "ParticipantRole":"CLINICIAN"
    }, {
      "ChannelId": 1,
      "ParticipantRole":"PATIENT"
    }
  ]
}
```

AWS SDK for Python (Boto3)

The following example uses the AWS SDK for Python (Boto3) to make a start_medical_scribe_job request. For more information, see StartMedicalScribeJob.

```
from __future__ import print_functionimport timeimport boto3
transcribe = boto3.client('transcribe', 'us-west-2')
job_name = "my-first-medical-scribe-job"
job_uri = "s3://amzn-s3-demo-bucket/my-input-files/my-media-file.flac"
transcribe.start_medical_scribe_job(
    MedicalScribeJobName = job_name,
    Media = {
      'MediaFileUri': job_uri
```

```
    },
    OutputBucketName = 'amzn-s3-demo-bucket',
    DataAccessRoleArn = 'arn:aws:iam::111122223333:role/ExampleRole',
    Settings = {
      'ShowSpeakerLabels': false,
      'ChannelIdentification': true
    },
    ChannelDefinitions = [
      {
        'ChannelId': 0,
        'ParticipantRole': 'CLINICIAN'
      }, {
        'ChannelId': 1,
        'ParticipantRole': 'PATIENT'
      }
    ]
)
while True:
    status = transcribe.get_medical_scribe_job(MedicalScribeJobName = job_name)
    if status['MedicalScribeJob']['MedicalScribeJobStatus'] in ['COMPLETED', 'FAILED']:
        break
    print("Not ready yet...")
    time.sleep(5)
print(status)
```

Note

The AWS Management Console does not currently support AWS HealthScribe jobs.

Transcription job output examples

In addition to a transcript, `StartMedicalScribeJob` requests generate a separate clinical documentation file. Both files are in JSON format and are stored in the output location you specify in your request. Here are examples of each output type:

Example transcript output

An AWS HealthScribe transcript file (from a `StartMedicalScribeJob` request) has the following format:

```
{
  "Conversation": {
    "ConversationId": "sampleConversationUUID",
    "JobName": "sampleJobName",
    "JobType": "ASYNC",
    "LanguageCode": "en-US",
    "ClinicalInsights": [
      {
        "Attributes": [],
        "Category": "MEDICAL_CONDITION",
        "InsightId": "insightUUID1",
        "InsightType": "ClinicalEntity",
        "Spans": [
          {
            "BeginCharacterOffset": 12,
            "Content": "pain",
            "EndCharacterOffset": 15,
            "SegmentId": "uuid1"
          }
        ],
        "Type": "DX_NAME"
      },
      {
        "Attributes": [],
        "Category": "TEST_TREATMENT_PROCEDURE",
        "InsightId": "insightUUID2",
        "InsightType": "ClinicalEntity",
        "Spans": [
          {
            "BeginCharacterOffset": 4,
            "Content": "mammogram",
            "EndCharacterOffset": 12,
            "SegmentId": "uuid2"
          }
        ],
        "Type": "TEST_NAME"
      },
      {
        "Attributes": [],
        "Category": "TEST_TREATMENT_PROCEDURE",
        "InsightId": "insightUUID3",
        "InsightType": "ClinicalEntity",
```

```
        "Spans": [
          {
            "BeginCharacterOffset": 15,
            "Content": "pap smear",
            "EndCharacterOffset": 23,
            "SegmentId": "uuid3"
          }
        ],
        "Type": "TEST_NAME"
      },
      {
        "Attributes": [],
        "Category": "MEDICATION",
        "InsightId": "insightUUID4",
        "InsightType": "ClinicalEntity",
        "Spans": [
          {
            "BeginCharacterOffset": 28,
            "Content": "phentermine",
            "EndCharacterOffset": 38,
            "SegmentId": "uuid4"
          }
        ],
        "Type": "GENERIC_NAME"
      },
      {
        "Attributes": [
          {
            "AttributeId": "attributeUUID1",
            "Spans": [
              {
                "BeginCharacterOffset": 38,
                "Content": "high",
                "EndCharacterOffset": 41,
                "SegmentId": "uuid5"
              }
            ],
            "Type": "TEST_VALUE"
          }
        ],
        "Category": "TEST_TREATMENT_PROCEDURE",
        "InsightId": "insightUUID5",
        "InsightType": "ClinicalEntity",
        "Spans": [
```

```
          {
            "BeginCharacterOffset": 14,
            "Content": "weight",
            "EndCharacterOffset": 19,
            "SegmentId": "uuid6"
          }
        ],
        "Type": "TEST_NAME"
      },
      {
        "Attributes": [],
        "Category": "ANATOMY",
        "InsightId": "insightUUID6",
        "InsightType": "ClinicalEntity",
        "Spans": [
          {
            "BeginCharacterOffset": 60,
            "Content": "heart",
            "EndCharacterOffset": 64,
            "SegmentId": "uuid7"
          }
        ],
        "Type": "SYSTEM_ORGAN_SITE"
      }
    ],
    "TranscriptItems": [
      {
        "Alternatives": [
          {
            "Confidence": 0.7925,
            "Content": "Okay"
          }
        ],
        "BeginAudioTime": 0.16,
        "EndAudioTime": 0.6,
        "Type": "PRONUNCIATION"
      },
      {
        "Alternatives": [
          {
            "Confidence": 0,
            "Content": "."
          }
        ],
```

```
      "BeginAudioTime": 0.17,
      "EndAudioTime": 0.9,
      "Type": "PUNCTUATION"
    },
    {
      "Alternatives": [
        {
          "Confidence": 1,
          "Content": "Good"
        }
      ],
      "BeginAudioTime": 0.61,
      "EndAudioTime": 0.92,
      "Type": "PRONUNCIATION"
    },
    {
      "Alternatives": [
        {
          "Confidence": 1,
          "Content": "afternoon"
        }
      ],
      "BeginAudioTime": 0.92,
      "EndAudioTime": 1.54,
      "Type": "PRONUNCIATION"
    },
    {
      "Alternatives": [
        {
          "Confidence": 0,
          "Content": "."
        }
      ],
      "BeginAudioTime": 0,
      "EndAudioTime": 0,
      "Type": "PUNCTUATION"
    },
    {
      "Alternatives": [
        {
          "Confidence": 0.9924,
          "Content": "You"
        }
      ],
```

```
        "BeginAudioTime": 1.55,
        "EndAudioTime": 1.88,
        "Type": "PRONUNCIATION"
      },
      {
        "Alternatives": [
          {
            "Confidence": 1,
            "Content": "lost"
          }
        ],
        "BeginAudioTime": 1.88,
        "EndAudioTime": 2.19,
        "Type": "PRONUNCIATION"
      },
      {
        "Alternatives": [
          {
            "Confidence": 1,
            "Content": "one"
          }
        ],
        "BeginAudioTime": 2.19,
        "EndAudioTime": 2.4,
        "Type": "PRONUNCIATION"
      },
      {
        "Alternatives": [
          {
            "Confidence": 1,
            "Content": "lb"
          }
        ],
        "BeginAudioTime": 2.4,
        "EndAudioTime": 2.97,
        "Type": "PRONUNCIATION"
      }
    ],
    "TranscriptSegments": [
      {
        "BeginAudioTime": 0.16,
        "Content": "Okay.",
        "EndAudioTime": 0.6,
        "ParticipantDetails": {
```

```
        "ParticipantRole": "CLINICIAN_0"
      },
      "SectionDetails": {
        "SectionName": "SUBJECTIVE"
      },
      "SegmentId": "uuid1"
    },
    {
      "BeginAudioTime": 0.61,
      "Content": "Good afternoon.",
      "EndAudioTime": 1.54,
      "ParticipantDetails": {
        "ParticipantRole": "CLINICIAN_0"
      },
      "SectionDetails": {
        "SectionName": "OTHER"
      },
      "SegmentId": "uuid2"
    },
    {
      "BeginAudioTime": 1.55,
      "Content": "You lost one lb.",
      "EndAudioTime": 2.97,
      "ParticipantDetails": {
        "ParticipantRole": "CLINICIAN_0"
      },
      "SectionDetails": {
        "SectionName": "SUBJECTIVE"
      },
      "SegmentId": "uuid3"
    },
    {
      "BeginAudioTime": 2.98,
      "Content": "Yeah, I think it, uh, do you feel more energy?",
      "EndAudioTime": 6.95,
      "ParticipantDetails": {
        "ParticipantRole": "CLINICIAN_0"
      },
      "SectionDetails": {
        "SectionName": "SUBJECTIVE"
      },
      "SegmentId": "uuid5"
    },
    {
```

```
        "BeginAudioTime": 6.96,
        "Content": "Yes.",
        "EndAudioTime": 7.88,
        "ParticipantDetails": {
          "ParticipantRole": "CLINICIAN_0"
        },
        "SectionDetails": {
          "SectionName": "SUBJECTIVE"
        },
        "SegmentId": "uuid6"
      },
      {
        "BeginAudioTime": 7.89,
        "Content": "Uh, how about craving for the carbohydrate or sugar or fat or
 anything?",
        "EndAudioTime": 17.93,
        "ParticipantDetails": {
          "ParticipantRole": "CLINICIAN_0"
        },
        "SectionDetails": {
          "SectionName": "SUBJECTIVE"
        },
        "SegmentId": "uuid7"
      }
    ]
  }
}
```

Here is another example using the start-medical-scribe-job command, and a request body with additional settings.

```
aws transcribe start-medical-scribe-job \
--region us-west-2 \
--cli-input-json file://filepath/my-first-medical-scribe-job.json
```

The file `my-first-medical-scribe-job.json` contains the following request body.

```
{
  "MedicalScribeJobName": "my-first-medical-scribe-job",
```

```
  "Media": {
    "MediaFileUri": "s3://amzn-s3-demo-bucket/my-input-files/my-media-file.flac"
   },
  "OutputBucketName": "amzn-s3-demo-bucket",
  "DataAccessRoleArn": "arn:aws:iam::111122223333:role/ExampleRole",
  "Settings": {
    "ShowSpeakerLabels": false,
    "ChannelIdentification": true
  },
  "ChannelDefinitions": [
    {
      "ChannelId": 0,
      "ParticipantRole":"CLINICIAN"
    }, {
      "ChannelId": 1,
      "ParticipantRole":"PATIENT"
    }
  ]
}
```

Example clinical documentation output

A documentation insights file (from a `StartMedicalScribeJob` request) has the following format:

```
{
  "ClinicalDocumentation": {
    "Sections": [
      {
        "SectionName": "CHIEF_COMPLAINT",
        "Summary": [
          {
            "EvidenceLinks": [
              {
                "SegmentId": "uuid1"
              },
              {
                "SegmentId": "uuid2"
              },
              {
                "SegmentId": "uuid3"
```

```
            },
            {
              "SegmentId": "uuid4"
            },
            {
              "SegmentId": "uuid5"
            },
            {
              "SegmentId": "uuid6"
            }
          ],
          "SummarizedSegment": "Weight loss."
        }
      ]
    },
    {
      "SectionName": "HISTORY_OF_PRESENT_ILLNESS",
      "Summary": [
        {
          "EvidenceLinks": [
            {
              "SegmentId": "uuid7"
            },
            {
              "SegmentId": "uuid8"
            },
            {
              "SegmentId": "uuid9"
            },
            {
              "SegmentId": "uuid10"
            }
          ],
          "SummarizedSegment": "The patient is seen today for a follow-up of weight
loss."
        },
        {
          "EvidenceLinks": [
            {
              "SegmentId": "uuid11"
            },
            {
              "SegmentId": "uuid12"
            },
```

```
            {
              "SegmentId": "uuid13"
            }
          ],
          "SummarizedSegment": "They report feeling more energy and craving
carbohydrates, sugar, and fat."
        },
        {
          "EvidenceLinks": [
            {
              "SegmentId": "uuid14"
            },
            {
              "SegmentId": "uuid15"
            },
            {
              "SegmentId": "uuid16"
            }
          ],
          "SummarizedSegment": "The patient is up to date on their mammogram and pap
smear."
        },
        {
          "EvidenceLinks": [
            {
              "SegmentId": "uuid17"
            },
            {
              "SegmentId": "uuid18"
            },
            {
              "SegmentId": "uuid19"
            },
            {
              "SegmentId": "uuid20"
            }
          ],
          "SummarizedSegment": "The patient is taking phentermine and would like to
continue."
        }
      ]
    },
    {
      "SectionName": "REVIEW_OF_SYSTEMS",
```

```
        "Summary": [
          {
            "EvidenceLinks": [
              {
                "SegmentId": "uuid21"
              },
              {
                "SegmentId": "uuid22"
              }
            ],
            "SummarizedSegment": "Patient reports intermittent headaches, occasional
chest pains but denies any recent fevers or chills."
          },
          {
            "EvidenceLinks": [
              {
                "SegmentId": "uuid23"
              },
              {
                "SegmentId": "uuid24"
              }
            ],
            "SummarizedSegment": "No recent changes in vision, hearing, or any
respiratory complaints."
          }
        ]
      },
      {
        "SectionName": "PAST_MEDICAL_HISTORY",
        "Summary": [
          {
            "EvidenceLinks": [
              {
                "SegmentId": "uuid25"
              },
              {
                "SegmentId": "uuid26"
              }
            ],
            "SummarizedSegment": "Patient has a history of hypertension and was
diagnosed with Type II diabetes 5 years ago."
          },
          {
            "EvidenceLinks": [
```

```
            {
              "SegmentId": "uuid27"
            },
            {
              "SegmentId": "uuid28"
            }
          ],
          "SummarizedSegment": "Underwent an appendectomy in the early '90s and had a
fracture in the left arm during childhood."
        }
      ]
    },
    {
      "SectionName": "ASSESSMENT",
      "Summary": [
        {
          "EvidenceLinks": [
            {
              "SegmentId": "uuid29"
            },
            {
              "SegmentId": "uuid30"
            }
          ],
          "SummarizedSegment": "Weight loss"
        }
      ]
    },
    {
      "SectionName": "PLAN",
      "Summary": [
        {
          "EvidenceLinks": [
            {
              "SegmentId": "uuid31"
            },
            {
              "SegmentId": "uuid32"
            },
            {
              "SegmentId": "uuid33"
            },
            {
              "SegmentId": "uuid34"
```

```
            }
          ],
          "SummarizedSegment": "For the condition of Weight loss: The patient was
 given a 30-day supply of phentermine and was advised to follow up in 30 days."
        }
      ]
    }
  ]
 }
}
```

AWS HealthScribe streaming

With AWS HealthScribe streaming, you can transcribe medical conversations in real-time. AWS HealthScribe streaming is a real-time HTTP2 based bi-directional service that accepts audio stream on one channel and vends an audio transcription on the other channel. After streaming is complete, AWS HealthScribe analyzes the stream contents and produces a transcript JSON file and a clinical note JSON file.

To start streaming, use the StartMedicalScribeStream API operation. This API starts an HTTP2 based bi-directional channel that you use to stream audio events.

When you start a stream, first specify the stream configuration in a `MedicalScribeConfigurationEvent`. This event includes channel definitions, encryption settings, and post-stream analytics settings, such as the output configuration for aggregated transcript and clinical note generation.

After you start streaming audio, you manage the stream as follows:

- When you are finished, to start processing the results with post-stream analytics, send a `MedicalScribeSessionControlEvent` with a `Type` of `END_OF_SESSION` and AWS HealthScribe starts the analytics.
- To pause streaming, complete the input stream without sending the `MedicalScribeSessionControlEvent`.
- To resume a paused stream, use the `StartMedicalScribeStream` API operation and specify the same `SessionId`. This is the `SessionId` you used when you originally started the stream.

Topics

- Guidelines and requirements
- ResourceAccessRoleArn role permissions
- Starting AWS HealthScribe streaming transcription

Guidelines and requirements

The following are guidelines and requirements for AWS HealthScribe streaming:

- Before you send audio events, you must first specify the stream configuration in a `MedicalScribeConfigurationEvent`.
- To run post-stream analytics, the `ResourceAccessRoleArn` in your `MedicalScribeConfigurationEvent` must have the correct permissions. For more information, see ResourceAccessRoleArn role permissions.
- You can resume a session any number of times within 5 hours from the initial stream creation.
- You can stream at most 2 hours of audio over a session across all streaming requests.
- By default, AWS HealthScribe provides encryption at rest to protect sensitive customer data using Amazon S3-managed keys. When you start a stream, you can specify a AWS KMS key for a second layer of encryption. Your `ResourceAccessRoleArn` must have permission to use your AWS KMS key. For more information, see Data Encryption at rest for AWS HealthScribe.
- You can use AWS HealthScribe streaming with the AWS SDKs, excluding the AWS SDK for Python (Boto3) and SDK for PHP.
- If a `LimitExceededException` exception occurs after you end a stream, you can restart the session and still generate post-stream analytics. To restart the stream, use the StartMedicalScribeStream API and use same `SessionID`. Then send a `MedicalScribeSessionControlEvent` with a `Type` of `END_OF_SESSION` and AWS HealthScribe starts the analytics.

ResourceAccessRoleArn role permissions

To run post-stream analytics, the `ResourceAccessRoleArn` in your `MedicalScribeConfigurationEvent` must be able to access your Amazon S3 output bucket and, if you provide it, your AWS KMS key. Also, the role's trust policy must grant the `transcribe.streaming.amazonaws.com` service permission to assume the role.

The following is an example IAM policy that grants Amazon S3 bucket permissions and AWS KMS key permissions. For more information, see Data Encryption at rest for AWS HealthScribe.

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Action": [
                "s3:PutObject"
            ],
            "Resource": [
                "arn:aws-eusc:s3:::amzn-s3-demo-bucket",
                "arn:aws-eusc:s3:::amzn-s3-demo-bucket/*"
            ],
            "Effect": "Allow"
        },
        {
            "Action": [
                "kms:DescribeKey",
                "kms:Decrypt",
                "kms:Encrypt",
                "kms:GenerateDataKey*"
            ],
            "Resource": "arn:aws-eusc:kms:us-
west-2:123456789012:key/1234abcd-12ab-34cd-56ef-123456SAMPLE",
            "Effect": "Allow"
        }
    ]
}
```

The following is an example trust policy.

JSON

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

```
            "Effect": "Allow",
            "Principal": {
                "Service": [
                    "transcribe.streaming.amazonaws.com"
                ]
            },
            "Action": "sts:AssumeRole"
        }
    ]
}
```

Starting AWS HealthScribe streaming transcription

The following code example shows how to set up a AWS HealthScribe streaming transcription using the AWS SDKs.

Topics

- SDK for Java 2.x
- Streaming transcription output examples

SDK for Java 2.x

The following example uses the SDK for Java 2.x to set up streaming and make a StartMedicalScribeStream request.

```
package org.example;

import io.reactivex.rxjava3.core.BackpressureStrategy;
import io.reactivex.rxjava3.core.Flowable;
import org.reactivestreams.Publisher;
import org.reactivestreams.Subscriber;
import software.amazon.awssdk.auth.credentials.AwsCredentialsProvider;
import software.amazon.awssdk.auth.credentials.DefaultCredentialsProvider;
import software.amazon.awssdk.core.SdkBytes;
import software.amazon.awssdk.regions.Region;
import
 software.amazon.awssdk.services.transcribestreaming.TranscribeStreamingAsyncClient;
import
 software.amazon.awssdk.services.transcribestreaming.model.ClinicalNoteGenerationSettings;
import software.amazon.awssdk.services.transcribestreaming.model.LanguageCode;
```

```
import software.amazon.awssdk.services.transcribestreaming.model.MediaEncoding;

import
 software.amazon.awssdk.services.transcribestreaming.model.MedicalScribeInputStream;
import
 software.amazon.awssdk.services.transcribestreaming.model.MedicalScribePostStreamAnalyticsSett
import
 software.amazon.awssdk.services.transcribestreaming.model.MedicalScribeSessionControlEventType
import
 software.amazon.awssdk.services.transcribestreaming.model.MedicalScribeTranscriptEvent;
import
 software.amazon.awssdk.services.transcribestreaming.model.MedicalScribeTranscriptSegment;
import
 software.amazon.awssdk.services.transcribestreaming.model.StartMedicalScribeStreamRequest;
import
 software.amazon.awssdk.services.transcribestreaming.model.StartMedicalScribeStreamResponseHand
import
 software.amazon.awssdk.services.transcribestreaming.model.medicalscribeinputstream.DefaultConf

import software.amazon.awssdk.http.nio.netty.NettyNioAsyncHttpClient;

import javax.sound.sampled.AudioFormat;
import javax.sound.sampled.AudioInputStream;
import javax.sound.sampled.AudioSystem;
import javax.sound.sampled.DataLine;
import javax.sound.sampled.LineUnavailableException;
import javax.sound.sampled.TargetDataLine;
import java.io.BufferedInputStream;
import java.io.IOException;
import java.io.InputStream;
import java.io.UncheckedIOException;
import java.util.Arrays;
import java.util.concurrent.CompletableFuture;


public class HealthScribeStreamingDemoApp {
    private static final int CHUNK_SIZE_IN_BYTES = 6400;
    private static final int SAMPLE_RATE = 16000;
    private static final Region REGION = Region.US_EAST_1;
    private static final String sessionId = "1234abcd-12ab-34cd-56ef-123456SAMPLE";
    private static final String bucketName = "amzn-s3-demo-bucket";
    private static final String resourceAccessRoleArn =
 "arn:aws:iam::123456789012:role/resource-access-role";
    private static TranscribeStreamingAsyncClient client;
```

```
   public static void main(String args[]) {

       client = TranscribeStreamingAsyncClient.builder()
               .credentialsProvider(getCredentials())
               .httpClientBuilder(NettyNioAsyncHttpClient.builder())
               .region(REGION)
               .build();
       try {
           StartMedicalScribeStreamRequest request =
StartMedicalScribeStreamRequest.builder()
                   .languageCode(LanguageCode.EN_US.toString())
                   .mediaSampleRateHertz(SAMPLE_RATE)
                   .mediaEncoding(MediaEncoding.PCM.toString())
                   .sessionId(sessionId)
                   .build();

           MedicalScribeInputStream endSessionEvent =
MedicalScribeInputStream.sessionControlEventBuilder()
                   .type(MedicalScribeSessionControlEventType.END_OF_SESSION)
                   .build();

           CompletableFuture<Void> result = client.startMedicalScribeStream(
                   request,
                   new AudioStreamPublisher(getStreamFromMic(),
getConfigurationEvent(),endSessionEvent),
                   getMedicalScribeResponseHandler());
           result.get();
           client.close();
       } catch (Exception e) {
           System.err.println("Error occurred: " + e.getMessage());
           e.printStackTrace();
       }
   }

   private static AudioInputStream getStreamFromMic() throws LineUnavailableException
{
       // Signed PCM AudioFormat with 16kHz, 16 bit sample size, mono
       AudioFormat format = new AudioFormat(SAMPLE_RATE, 16, 1, true, false);
       DataLine.Info info = new DataLine.Info(TargetDataLine.class, format);

       if (!AudioSystem.isLineSupported(info)) {
           System.out.println("Line not supported");
```

```
            throw new LineUnavailableException("The audio system microphone line is not
supported.");
        }
        TargetDataLine line = (TargetDataLine) AudioSystem.getLine(info);
        int bufferSize = (CHUNK_SIZE_IN_BYTES / format.getFrameSize()) *
format.getFrameSize();
        line.open(format);
        line.start();

        // Create a wrapper class that can be closed when Enter is pressed
        AudioInputStream audioStream = new AudioInputStream(line);

        // Start a thread to monitor for Enter key
        System.out.println("Recording... Press Enter to stop");
        Thread monitorThread = new Thread(() -> {
            try {
                System.in.read();
                line.stop();
                line.close();
            } catch (IOException e) {
                e.printStackTrace();
            }
        });
        monitorThread.setDaemon(true);  // Set as daemon thread so it doesn't prevent
JVM shutdown
        monitorThread.start();

        return new AudioInputStream(
            new BufferedInputStream(new AudioInputStream(line)),
            format,
            AudioSystem.NOT_SPECIFIED
        );
    }

    private static AwsCredentialsProvider getCredentials() {
        return DefaultCredentialsProvider.create();
    }

    private static StartMedicalScribeStreamResponseHandler
getMedicalScribeResponseHandler() {

        return StartMedicalScribeStreamResponseHandler.builder()
            .onResponse(r -> {
                System.out.println("Received Initial response");
```

```
            })
            .onError(Throwable::printStackTrace)
            .onComplete(() -> {
                System.out.println("=== All records streamed successfully ===");
            })
            .subscriber(event -> {
                if (event instanceof MedicalScribeTranscriptEvent) {
                    MedicalScribeTranscriptSegment segment =
 ((MedicalScribeTranscriptEvent) event).transcriptSegment();
                    if (segment != null && segment.content() != null && !
segment.content().isEmpty()) {
                        System.out.println(segment.content());
                    }
                }
            })
            .build();
    }

    private static DefaultConfigurationEvent getConfigurationEvent() {
        MedicalScribePostStreamAnalyticsSettings postStreamSettings =
 MedicalScribePostStreamAnalyticsSettings
                .builder()
                .clinicalNoteGenerationSettings(
                        ClinicalNoteGenerationSettings.builder()
                                .outputBucketName(bucketName)
                                .build()
                )
                .build();
        return (DefaultConfigurationEvent)
 MedicalScribeInputStream.configurationEventBuilder()
                .resourceAccessRoleArn(resourceAccessRoleArn)
                .postStreamAnalyticsSettings(postStreamSettings)
                .build();
    }

    private static class AudioStreamPublisher implements
 Publisher<MedicalScribeInputStream> {
        private final InputStream audioInputStream;
        private final MedicalScribeInputStream configEvent;
        private final MedicalScribeInputStream endSessionEvent;

        private AudioStreamPublisher(AudioInputStream audioInputStream,
                                     MedicalScribeInputStream configEvent,
                                     MedicalScribeInputStream endSessionEvent) {
```

```
            this.audioInputStream = audioInputStream;
            this.configEvent = configEvent;
            this.endSessionEvent = endSessionEvent;
        }

        @Override
        public void subscribe(Subscriber<? super MedicalScribeInputStream> subscriber)
{
            createAudioFlowable()
                    .doOnComplete(() -> {
                        try {
                            audioInputStream.close();
                        } catch (IOException e) {
                            throw new UncheckedIOException(e);
                        }
                    })
                    .subscribe(subscriber);
        }

        private Flowable<MedicalScribeInputStream> createAudioFlowable() {
            // Start with config event
            Flowable<MedicalScribeInputStream> configFlow = Flowable.just(configEvent);

            // Create audio chunk flowable
            Flowable<MedicalScribeInputStream> audioFlow = Flowable.create(emitter -> {
                byte[] buffer = new byte[CHUNK_SIZE_IN_BYTES];
                int bytesRead;

                try {
                    while (!emitter.isCancelled() && (bytesRead =
audioInputStream.read(buffer)) > 0) {
                        byte[] audioData = bytesRead < buffer.length
                                ? Arrays.copyOfRange(buffer, 0, bytesRead)
                                : buffer;

                        MedicalScribeInputStream audioEvent =
MedicalScribeInputStream.audioEventBuilder()
                                .audioChunk(SdkBytes.fromByteArray(audioData))
                                .build();

                        emitter.onNext(audioEvent);
                    }
                    emitter.onComplete();
                } catch (IOException e) {
```

```
                    emitter.onError(e);
                }
            }, BackpressureStrategy.BUFFER);

            // End with session end event
            Flowable<MedicalScribeInputStream> endFlow =
 Flowable.just(endSessionEvent);

            // Concatenate all flows
            return Flowable.concat(configFlow, audioFlow, endFlow);
        }
    }
}
```

Streaming transcription output examples

After streaming is complete, AWS HealthScribe analyzes the stream contents and produces a transcript JSON file and a clinical note JSON file. Here are examples of each output type:

Example transcript output

The following is an example of a AWS HealthScribe transcript file from a streaming session.

```
{
    "Conversation": {
        "ClinicalInsights": [{
            "Attributes": [],
            "Category": "MEDICAL_CONDITION",
            "InsightId": "insightUUID1",
            "InsightType": "ClinicalEntity",
            "Spans": [{
                "BeginCharacterOffset": 12,
                "Content": "pain",
                "EndCharacterOffset": 15,
                "SegmentId": "uuid1"
            }],
            "Type": "DX_NAME"
        }, {
            "Attributes": [],
            "Category": "TEST_TREATMENT_PROCEDURE",
            "InsightId": "insightUUID2",
            "InsightType": "ClinicalEntity",
            "Spans": [{
```

```
                "BeginCharacterOffset": 4,
                "Content": "mammogram",
                "EndCharacterOffset": 12,
                "SegmentId": "uuid2"
            }],
            "Type": "TEST_NAME"
        }, {
            "Attributes": [],
            "Category": "TEST_TREATMENT_PROCEDURE",
            "InsightId": "insightUUID3",
            "InsightType": "ClinicalEntity",
            "Spans": [{
                "BeginCharacterOffset": 15,
                "Content": "pap smear",
                "EndCharacterOffset": 23,
                "SegmentId": "uuid3"
            }],
            "Type": "TEST_NAME"
        }, {
            "Attributes": [],
            "Category": "MEDICATION",
            "InsightId": "insightUUID4",
            "InsightType": "ClinicalEntity",
            "Spans": [{
                "BeginCharacterOffset": 28,
                "Content": "phentermine",
                "EndCharacterOffset": 38,
                "SegmentId": "uuid4"
            }],
            "Type": "GENERIC_NAME"
        }, {
            "Attributes": [{
                "AttributeId": "attributeUUID1",
                "Spans": [{
                    "BeginCharacterOffset": 38,
                    "Content": "high",
                    "EndCharacterOffset": 41,
                    "SegmentId": "uuid5"
                }],
                "Type": "TEST_VALUE"
            }],
            "Category": "TEST_TREATMENT_PROCEDURE",
            "InsightId": "insightUUID5",
            "InsightType": "ClinicalEntity",
```

```
            "Spans": [{
                "BeginCharacterOffset": 14,
                "Content": "weight",
                "EndCharacterOffset": 19,
                "SegmentId": "uuid6"
            }],
            "Type": "TEST_NAME"
        }, {
            "Attributes": [],
            "Category": "ANATOMY",
            "InsightId": "insightUUID6",
            "InsightType": "ClinicalEntity",
            "Spans": [{
                "BeginCharacterOffset": 60,
                "Content": "heart",
                "EndCharacterOffset": 64,
                "SegmentId": "uuid7"
            }],
            "Type": "SYSTEM_ORGAN_SITE"
        }],
        "ConversationId": "sampleConversationUUID",
        "LanguageCode": "en-US",
        "SessionId": "sampleSessionUUID",
        "TranscriptItems": [{
            "Alternatives": [{
                "Confidence": 0.7925,
                "Content": "Okay"
            }],
            "BeginAudioTime": 0.16,
            "EndAudioTime": 0.6,
            "Type": "PRONUNCIATION"
        },
        {
            "Alternatives": [{
                "Confidence": 0,
                "Content": "."
            }],
            "BeginAudioTime": 0,
            "EndAudioTime": 0,
            "Type": "PUNCTUATION"
        },
        {
            "Alternatives": [{
                "Confidence": 1,
```

```
                "Content": "Good"
            }],
            "BeginAudioTime": 0.61,
            "EndAudioTime": 0.92,
            "Type": "PRONUNCIATION"
        },
        {
            "Alternatives": [{
                "Confidence": 1,
                "Content": "afternoon"
            }],
            "BeginAudioTime": 0.92,
            "EndAudioTime": 1.54,
            "Type": "PRONUNCIATION"
        },
        {
            "Alternatives": [{
                "Confidence": 0,
                "Content": "."
            }],
            "BeginAudioTime": 0,
            "EndAudioTime": 0,
            "Type": "PUNCTUATION"
        },
        {
            "Alternatives": [{
                "Confidence": 0.9924,
                "Content": "You"
            }],
            "BeginAudioTime": 1.55,
            "EndAudioTime": 1.88,
            "Type": "PRONUNCIATION"
        },
        {
            "Alternatives": [{
                "Confidence": 1,
                "Content": "lost"
            }],
            "BeginAudioTime": 1.88,
            "EndAudioTime": 2.19,
            "Type": "PRONUNCIATION"
        },
        {
            "Alternatives": [{
```

```
                "Confidence": 1,
                "Content": "one"
            }],
            "BeginAudioTime": 2.19,
            "EndAudioTime": 2.4,
            "Type": "PRONUNCIATION"
        },
        {
            "Alternatives": [{
                "Confidence": 1,
                "Content": "lb"
            }],
            "BeginAudioTime": 2.4,
            "EndAudioTime": 2.97,
            "Type": "PRONUNCIATION"
        }
        ],
        "TranscriptSegments": [{
            "BeginAudioTime": 0.16,
            "Content": "Okay.",
            "EndAudioTime": 0.6,
            "ParticipantDetails": {
                "ParticipantRole": "CLINICIAN_0"
            },
            "SectionDetails": {
                "SectionName": "SUBJECTIVE"
            },
            "SegmentId": "uuid1"
        }, {
            "BeginAudioTime": 0.61,
            "Content": "Good afternoon.",
            "EndAudioTime": 1.54,
            "ParticipantDetails": {
                "ParticipantRole": "CLINICIAN_0"
            },
            "SectionDetails": {
                "SectionName": "OTHER"
            },
            "SegmentId": "uuid2"
        }, {
            "BeginAudioTime": 1.55,
            "Content": "You lost one lb.",
            "EndAudioTime": 2.97,
            "ParticipantDetails": {
```

```
                "ParticipantRole": "CLINICIAN_0"
            },
            "SectionDetails": {
                "SectionName": "SUBJECTIVE"
            },
            "SegmentId": "uuid3"
        }, {
            "BeginAudioTime": 2.98,
            "Content": "Yeah, I think it, uh, do you feel more energy?",
            "EndAudioTime": 6.95,
            "ParticipantDetails": {
                "ParticipantRole": "CLINICIAN_0"
            },
            "SectionDetails": {
                "SectionName": "SUBJECTIVE"
            },
            "SegmentId": "uuid4"
        }, {
            "BeginAudioTime": 6.96,
            "Content": "Yes.",
            "EndAudioTime": 7.88,
            "ParticipantDetails": {
                "ParticipantRole": "CLINICIAN_0"
            },
            "SectionDetails": {
                "SectionName": "SUBJECTIVE"
            },
            "SegmentId": "uuid5"
        }, {
            "BeginAudioTime": 7.89,
            "Content": "Uh, how about craving for the carbohydrate or sugar or fat or
 anything?",
            "EndAudioTime": 17.93,
            "ParticipantDetails": {
                "ParticipantRole": "CLINICIAN_0"
            },
            "SectionDetails": {
                "SectionName": "SUBJECTIVE"
            },
            "SegmentId": "uuid6"
        }]
    }
}
```

Example Clinical Documentation output

The following is an example of a AWS HealthScribe clinical documentation insights file from a streaming session.


```
{
  "ClinicalDocumentation": {
    "Sections": [
      {
        "SectionName": "CHIEF_COMPLAINT",
        "Summary": [
          {
            "EvidenceLinks": [
              {
                "SegmentId": "uuid1"
              },
              {
                "SegmentId": "uuid2"
              },
              {
                "SegmentId": "uuid3"
              },
              {
                "SegmentId": "uuid4"
              },
              {
                "SegmentId": "uuid5"
              },
              {
                "SegmentId": "uuid6"
              }
            ],
            "SummarizedSegment": "Weight loss."
          }
        ]
      },
      {
        "SectionName": "HISTORY_OF_PRESENT_ILLNESS",
        "Summary": [
          {
            "EvidenceLinks": [
              {
                "SegmentId": "uuid7"
              },
```

```
            {
              "SegmentId": "uuid8"
            },
            {
              "SegmentId": "uuid9"
            },
            {
              "SegmentId": "uuid10"
            }
          ],
          "SummarizedSegment": "The patient is seen today for a follow-up of weight
loss."
        },
        {
          "EvidenceLinks": [
            {
              "SegmentId": "uuid11"
            },
            {
              "SegmentId": "uuid12"
            },
            {
              "SegmentId": "uuid13"
            }
          ],
          "SummarizedSegment": "They report feeling more energy and craving
carbohydrates, sugar, and fat."
        },
        {
          "EvidenceLinks": [
            {
              "SegmentId": "uuid14"
            },
            {
              "SegmentId": "uuid15"
            },
            {
              "SegmentId": "uuid16"
            }
          ],
          "SummarizedSegment": "The patient is up to date on their mammogram and pap
smear."
        },
        {
```

```
            "EvidenceLinks": [
              {
                "SegmentId": "uuid17"
              },
              {
                "SegmentId": "uuid18"
              },
              {
                "SegmentId": "uuid19"
              },
              {
                "SegmentId": "uuid20"
              }
            ],
            "SummarizedSegment": "The patient is taking phentermine and would like to
continue."
          }
        ]
      },
      {
        "SectionName": "REVIEW_OF_SYSTEMS",
        "Summary": [
          {
            "EvidenceLinks": [
              {
                "SegmentId": "uuid21"
              },
              {
                "SegmentId": "uuid22"
              }
            ],
            "SummarizedSegment": "Patient reports intermittent headaches, occasional
chest pains but denies any recent fevers or chills."
          },
          {
            "EvidenceLinks": [
              {
                "SegmentId": "uuid23"
              },
              {
                "SegmentId": "uuid24"
              }
            ],
```

```
            "SummarizedSegment": "No recent changes in vision, hearing, or any
respiratory complaints."
          }
        ]
      },
      {
        "SectionName": "PAST_MEDICAL_HISTORY",
        "Summary": [
          {
            "EvidenceLinks": [
              {
                "SegmentId": "uuid25"
              },
              {
                "SegmentId": "uuid26"
              }
            ],
            "SummarizedSegment": "Patient has a history of hypertension and was
diagnosed with Type II diabetes 5 years ago."
          },
          {
            "EvidenceLinks": [
              {
                "SegmentId": "uuid27"
              },
              {
                "SegmentId": "uuid28"
              }
            ],
            "SummarizedSegment": "Underwent an appendectomy in the early '90s and had a
fracture in the left arm during childhood."
          }
        ]
      },
      {
        "SectionName": "ASSESSMENT",
        "Summary": [
          {
            "EvidenceLinks": [
              {
                "SegmentId": "uuid29"
              },
              {
                "SegmentId": "uuid30"
```

```
              }
            ],
            "SummarizedSegment": "Weight loss"
          }
        ]
      },
      {
        "SectionName": "PLAN",
        "Summary": [
          {
            "EvidenceLinks": [
              {
                "SegmentId": "uuid31"
              },
              {
                "SegmentId": "uuid32"
              },
              {
                "SegmentId": "uuid33"
              },
              {
                "SegmentId": "uuid34"
              }
            ],
            "SummarizedSegment": "For the condition of Weight loss: The patient was
 given a 30-day supply of phentermine and was advised to follow up in 30 days."
          }
        ]
      }
    ],
    "SessionId": "sampleSessionUUID"
  }
}
```

Data Encryption at rest for AWS HealthScribe

By default, AWS HealthScribe provides encryption at rest to protect sensitive customer data using AWS HealthScribe managed AWS Key Management Service (AWS KMS) keys. Encryption of data at rest by default helps reduce the operational overhead and complexity involved in protecting sensitive data. Also, it enables you to build secure applications that meet strict encryption compliance and regulatory requirements. When you create an AWS HealthScribe transcription job or start a stream, you can specify a customer managed key. This adds a second layer of encryption.

- **AWS HealthScribe managed AWS KMS keys** — AWS HealthScribe uses AWS HealthScribe managed AWS Key Management Service (AWS KMS) keys by default to automatically encrypt intermediate files. You can't disable this layer of encryption or choose an alternate encryption type. You can't view, manage, or use the keys, or audit their use. However, you don't have to take any action or change any programs to protect the keys that encrypt your data.
- **Customer managed keys** — AWS HealthScribe supports the use of a symmetric customer managed key that you create, own, and manage to add a second layer of encryption over the existing AWS owned encryption. Because you have full control of this layer of encryption, you can perform such tasks as:
  - Establishing and maintaining key policies
  - Establishing and maintaining IAM policies and grants
  - Enabling and disabling key policies
  - Rotating key cryptographic material
  - Adding tags
  - Creating key aliases
  - Scheduling keys for deletion

  For more information, see customer managed key in the AWS Key Management Service Developer Guide.

> **Note**
>
> AWS HealthScribe automatically enables encryption at rest using AWS-owned keys to protect personally identifiable data at no charge. However, AWS KMS charges apply for using a customer managed key. For more information about pricing, see AWS Key Management Service pricing.
> For more information on AWS KMS, see What is AWS Key Management Service.

Topics

- Creating a customer managed key
- Specifying a customer managed key for AWS HealthScribe
- AWS KMS encryption context
- Monitoring your encryption keys for AWS HealthScribe

Creating a customer managed key

You can create a symmetric customer managed key by using the AWS Management Console, or the AWS KMS APIs. To create a symmetric customer managed key, follow the steps for Creating symmetric customer managed key in the AWS Key Management Service Developer Guide.

Key policies control access to your customer managed key. Every customer managed key must have exactly one key policy, which contains statements that determine who can use the key and how they can use it. When you create your customer managed key, you can specify a key policy. For more information, see Managing access to customer managed keys in the AWS Key Management Service Developer Guide.

AWS KMS key policies for AWS HealthScribe

If you are using a key in the same account as the IAM role you specify as the `DataAccessRole` in your StartMedicalScribeJob or `ResourceAccessRole` in your StartMedicalScribeStream request, you don't need to update the Key Policy. To use your customer managed key in a different account as your DataAccessRole (for transcription jobs) or ResourceAccessRole (for streaming), you must trust the respective role in the Key Policy for the following actions:

- `kms:Encrypt` — Allows encryption using the customer managed key
- `kms:Decrypt` — Allows decryption using the customer managed key
- `kms:DescribeKey` — Provides the customer managed key details to allow AWS HealthScribe to validate the key

The following is an example key policy you can use to grant your ResourceAccessRole cross account permissions to use your customer managed key for AWS HealthScribe streaming. To use this policy for transcription jobs, update the `Principal` to use the DataAccessRole ARN, and remove or modify the encryption context.

JSON

```
{
   "Version":"2012-10-17",
   "Statement":[
      {
         "Sid": "AllowAccessForKeyAdministrators",
         "Effect": "Allow",
```

```
            "Principal": {
               "AWS": "arn:aws-eusc:iam::111122223333:root"
            },
            "Action": [
              "kms:*"
            ],
            "Resource": "*"
         },
         {
            "Sid": "AllowAccessToResourceAccessRoleForMedicalScribe",
            "Effect": "Allow",
            "Principal": {
               "AWS": "arn:aws-eusc:iam::111122223333:role/ResourceAccessRole"
            },
            "Action": [
               "kms:Encrypt",
               "kms:Decrypt",
               "kms:GenerateDataKey*"
            ],
            "Resource": "*",
            "Condition": {
               "StringEquals": {
                   "kms:EncryptionContext:aws:eusc-de-east-1:transcribe:medical-
scribe:session-id": "1234abcd-12ab-34cd-56ef-123456SAMPLE"
               }
            }
         },
         {
            "Sid": "AllowAccessToResourceAccessRoleForDescribeKey",
            "Effect": "Allow",
            "Principal": {
                "AWS": "arn:aws-eusc:iam::111122223333:role/ResourceAccessRole"
            },
            "Action": "kms:DescribeKey",
            "Resource": "*"
        }
      ]
}
```

IAM policy permissions for access roles

The IAM policy attached to your DataAccessRole or ResourceAccessRole must grant permissions to perform the necessary AWS KMS actions, regardless of whether the customer-managed key and role are in the same or different accounts. Also, the role's trust policy must grant AWS HealthScribe permission to assume the role.

The following IAM policy example shows how to grant a ResourceAccessRole permissions for AWS HealthScribe streaming. To use this policy for transcription jobs, remove or modify the encryption context. The `kms:ViaService` condition value is the same for streaming and transcription jobs.

JSON

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "kms:Encrypt",
                "kms:Decrypt",
                "kms:GenerateDataKey*"
            ],
            "Resource": "arn:aws-eusc:kms:us-west-2:111122223333:key/KMS-Example-
KeyId",
            "Condition": {
                "StringEquals": {
                    "kms:ViaService": "transcribe.us-west-2.amazonaws.com",
                    "kms:EncryptionContext:aws:eusc-de-east-1:transcribe:medical-
scribe:session-id": "1234abcd-12ab-34cd-56ef-123456SAMPLE"
                }
            }
        },
        {
            "Effect": "Allow",
            "Action": [
                "kms:DescribeKey"
            ],
            "Resource": "arn:aws-eusc:kms:us-west-2:111122223333:key/KMS-Example-
KeyId",
            "Condition": {
                "StringEquals": {
                    "kms:ViaService": "transcribe.us-west-2.amazonaws.com"
```

```
                }
            }
        }
    ]
}
```

The following is trust policy example for the ResourceAccessRole. For DataAccessRole, replace `transcribe.streaming.amazonaws.com` with `transcribe.amazonaws.com`.

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Principal": {
                "Service": "transcribe.streaming.amazonaws.com"
            },
            "Action": "sts:AssumeRole",
            "Condition": {
                "StringEquals": {
                    "aws:SourceAccount": "111122223333"
                },
                "ArnLike": {
                    "aws:SourceArn": "arn:aws-eusc:transcribe:us-
west-2:111122223333:*"
                }
            }
        }
    ]
}
```

For more information about specifying permissions in a policy or troubleshooting key access, see the AWS Key Management Service Developer Guide.

Specifying a customer managed key for AWS HealthScribe

You can specify a customer managed key as a second layer encryption for transcription jobs or streaming.

- For transcription jobs, you specify your key in the OutputEncryptionKMSKeyId of your StartMedicalScribeJob API operation.
- For streaming, you specify the key in the MedicalScribeEncryptionSettings in your MedicalScribeConfigurationEvent.

AWS KMS encryption context

AWS KMS encryption context is a map of plain text, non-secret key:value pairs. This map represents additional authenticated data, known as encryption context pairs, which provide an added layer of security for your data. AWS HealthScribe requires a symmetric encryption key to encrypt AWS HealthScribe output into a customer-specified Amazon S3 bucket. To learn more, see Asymmetric keys in AWS KMS.

When creating your encryption context pairs, *do not* include sensitive information. Encryption context is not secret — it is visible in plain text within your CloudTrail logs (so you can use it to identify and categorize your cryptographic operations). Your encryption context pair can include special characters, such as underscores (_), dashes (-), slashes (/, \) and colons (:).

Tip

It can be useful to relate the values in your encryption context pair to the data being encrypted. Although not required, we recommend you use non-sensitive metadata related to your encrypted content, such as file names, header values, or unencrypted database fields.

To use output encryption with the API, set the KMSEncryptionContext parameter in the StartMedicalScribeJob operation. In order to provide encryption context for the output encryption operation, the OutputEncryptionKMSKeyId parameter must reference a symmetric AWS KMS key ID.

For streaming, you specify the key value pairs for the `KmsEncryptionContext` in the MedicalScribeEncryptionSettings in your MedicalScribeConfigurationEvent.

You can use AWS KMS condition keys with IAM policies to control access to a symmetric encryption AWS KMS key based on the encryption context that was used in the request for a cryptographic operation. For an example encryption context policy, see AWS KMS encryption context policy.

Using encryption context is optional, but recommended. For more information, see Encryption context.

AWS HealthScribe encryption context

AWS HealthScribe uses the same encryption context in all AWS Key Management Service cryptographic operations. The encryption context is a map of String to String that can be customized to anything you want.

```
"encryptionContext": {
   "ECKey": "ECValue"
   ...
}
```

For AWS HealthScribe streams, the following is the default service generated encryption context. It applies this context on top of any encryption context that you provide.

```
"encryptionContext": {
  "aws:<region>:transcribe:medical-scribe:session-id":
 "1234abcd-12ab-34cd-56ef-123456SAMPLE"
}
```

For AWS HealthScribe transcription jobs, the following is the default service generated encryption context. It applies this context on top of any encryption context that you provide.

```
"encryptionContext": {
  "aws:<region>:transcribe:medical-scribe:job-name": "<job-name>",
  "aws:<region>:transcribe:medical-scribe:start-time-epoch-ms": "<job-start-time>"
}
```

If you don't provide any encryption context, only service generated encryption context will be used for all AWS KMS cryptographic operations.

Monitoring AWS HealthScribe with encryption context

When you use a symmetric customer managed key to encrypt your data at rest in AWS HealthScribe, you can also use the encryption context in audit records and logs to identify how the customer managed key is being used. The encryption context also appears in logs generated by AWS CloudTrail or CloudWatch Logs.

Using encryption context to control access to your customer managed key

You can use the encryption context in key policies and IAM policies as conditions to control access to your symmetric customer managed key.

The following are example key policy statements to grant access to a customer managed key for a specific encryption context. The condition in this policy statement requires that the KMS key usages have an encryption context constraint that specifies the encryption context.

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Sid": "AllowAccessToResourceAccessRoleForMedicalScribe",
            "Effect": "Allow",
            "Principal": {
                "AWS": "arn:aws-eusc:iam::111122223333:role/ResourceAccessRole"
            },
            "Action": [
                "kms:Encrypt",
                "kms:Decrypt",
                "kms:GenerateDataKey*"
            ],
            "Resource": "arn:aws-eusc:kms:us-west-2:111122223333:key/KMS-Example-
KeyId",
            "Condition": {
                "StringEquals": {
                    "kms:EncryptionContext:aws:eusc-de-east-1:transcribe:medical-
scribe:session-id": "1234abcd-12ab-34cd-56ef-123456SAMPLE",
                    "kms:EncryptionContext:ECKey": "ECValue"
                }
            }
        },
        {
            "Sid": "AllowAccessToResourceAccessRoleForDescribeKey",
            "Effect": "Allow",
            "Principal": {
                "AWS": "arn:aws-eusc:iam::111122223333:role/ResourceAccessRole"
            },
            "Action": "kms:DescribeKey",
            "Resource": "arn:aws-eusc:kms:us-west-2:111122223333:key/KMS-Example-
KeyId"
        }
    ]
}
```

Monitoring your encryption keys for AWS HealthScribe

When you use an AWS Key Management Service customer managed key with AWS HealthScribe, you can use AWS CloudTrail or CloudWatch logs to track requests that AWS HealthScribe sends to AWS KMS.

The following examples are CloudTrail Encrypt and Decrypt events you can use that allow you to monitor how AWS HealthScribe uses of your customer managed key.

Encrypt


```
{
   "eventVersion":"1.09",
   "userIdentity":{
      "type":"AssumedRole",
      "principalId":"AROAIGDTESTANDEXAMPLE:Sampleuser01",
      "arn":"arn:aws:sts::123456789012:assumed-role/Admin/Sampleuser01",
      "accountId":"123456789012",
      "accessKeyId":"AKIAIOSFODNN7EXAMPLE3",
      "sessionContext":{
         "sessionIssuer":{
            "type":"Role",
            "principalId":"AROAIGDTESTANDEXAMPLE:Sampleuser01",
            "arn":"arn:aws:sts::123456789012:assumed-role/Admin/Sampleuser01",
            "accountId":"123456789012",
            "userName":"Admin"
         },
         "attributes":{
            "creationDate":"2024-08-16T01:10:05Z",
            "mfaAuthenticated":"false"
         }
      },
      "invokedBy":"transcribe.streaming.amazonaws.com"
   },
   "eventTime":"2024-08-16T01:10:05Z",
   "eventSource":"kms.amazonaws.com",
   "eventName":"Encrypt",
   "awsRegion":"us-east-1",
   "sourceIPAddress":"transcribe.streaming.amazonaws.com",
   "userAgent":"transcribe.streaming.amazonaws.com",
   "requestParameters":{
      "encryptionContext":{
```

```
         "aws:us-east-1:transcribe:medical-scribe:session-
id":"1234abcd-12ab-34cd-56ef-123456SAMPLE"
      },
      "encryptionAlgorithm":"SYMMETRIC_DEFAULT",
      "keyId":"1234abcd-12ab-34cd-56ef-1234567890ab"
   },
   "responseElements":null,
   "requestID":"cbe0ac33-8cca-49e5-9bb5-dc2b8dfcb389",
   "eventID":"1b9fedde-aa96-48cc-9dd9-a2cce2964b3c",
   "readOnly":true,
   "resources":[
      {
         "accountId":"123456789012",
         "type":"AWS::KMS::Key",
         "ARN":"arn:aws:kms:us-
west-2:123456789012:key/1234abcd-12ab-34cd-56ef-123456SAMPLE"
      }
   ],
   "eventType":"AwsApiCall",
   "managementEvent":true,
   "recipientAccountId":"123456789012",
   "eventCategory":"Management"
}
```

Decrypt

```
{
   "eventVersion":"1.09",
   "userIdentity":{
      "type":"AssumedRole",
      "principalId":"AROAIGDTESTANDEXAMPLE:Sampleuser01",
      "arn":"arn:aws:sts::123456789012:assumed-role/Admin/Sampleuser01",
      "accountId":"123456789012",
      "accessKeyId":"AKIAIOSFODNN7EXAMPLE3",
      "sessionContext":{
         "sessionIssuer":{
            "type":"Role",
            "principalId":"AROAIGDTESTANDEXAMPLE:Sampleuser01",
            "arn":"arn:aws:sts::123456789012:assumed-role/Admin/Sampleuser01",
            "accountId":"123456789012",
            "userName":"Admin"
```

```
         },
         "attributes":{
            "creationDate":"2024-08-16T20:47:04Z",
            "mfaAuthenticated":"false"
         }
      },
      "invokedBy":"transcribe.streaming.amazonaws.com"
   },
   "eventTime":"2024-08-16T20:47:04Z",
   "eventSource":"kms.amazonaws.com",
   "eventName":"Decrypt",
   "awsRegion":"us-east-1",
   "sourceIPAddress":"transcribe.streaming.amazonaws.com",
   "userAgent":"transcribe.streaming.amazonaws.com",
   "requestParameters":{
      "keyId":"mrk-de27f019178f4fbf86512ab03ba860be",
      "encryptionAlgorithm":"SYMMETRIC_DEFAULT",
      "encryptionContext":{
         "aws:us-east-1:transcribe:medical-scribe:session-
id":"1234abcd-12ab-34cd-56ef-123456SAMPLE"
      }
   },
   "responseElements":null,
   "requestID":"8b7fb865-48be-4e03-ac3d-e7bee3ba30a1",
   "eventID":"68b7a263-d410-4701-9e2b-20c196628966",
   "readOnly":true,
   "resources":[
      {
         "accountId":"123456789012",
         "type":"AWS::KMS::Key",
         "ARN":"arn:aws:kms:us-
west-2:123456789012:key/1234abcd-12ab-34cd-56ef-123456SAMPLE"
      }
   ],
   "eventType":"AwsApiCall",
   "managementEvent":true,
   "recipientAccountId":"123456789012",
   "eventCategory":"Management"
}
```

FAQs on supported protocols

The Amazon Transcribe service is adding SDK support for the Smithy RPC v2 CBOR protocol in addition to the existing AWS JSON protocol. The Smithy RPC v2 CBOR protocol is more performant than AWS JSON for most workloads. AWS SDKs will prioritize the protocol that is the most performant for each language.

What are the AWS JSON and Smithy RPC v2 CBOR protocols, and how do they differ?

A wire protocol acts as a mediator between a client and the service. The AWS SDK handles the serialization (converting an object to the wire format) and deserialization (converting the wire format back to an object) between the client and server transparently for both protocols.

AWS JSON - JSON is one of the most widely used and accepted data formats for communication between systems. This service uses AWS JSON as a medium to communicate between an AWS SDK client and the server. An HTTP request for a service API operation accepts input in the form of JSON, the operation is executed, and the response is returned to the SDK client in the form of JSON.

Smithy RPC v2 CBOR - Smithy RPC v2 CBOR (Concise Binary Object Representation) is a protocol developed by Amazon for more efficient data transmission. It is a data format designed to keep message sizes compact and to allow the format to evolve without requiring version negotiation between systems. The key difference from AWS JSON is that AWS JSON uses human-readable text (like `{"name": "John"}`), while Smithy RPC v2 CBOR uses a binary format (sequences of 1s and 0s). This binary approach creates smaller data packages that travel faster over the network and reduces serialization time since computers process binary data directly.

How do I get started with the Smithy RPC v2 CBOR protocol?

To get started, use the latest AWS SDK/CLI version to use the Smithy RPC v2 CBOR protocol for this service. The SDK will automatically use the Smithy RPC v2 CBOR protocol when available. Upgrade your AWS SDK to the specified version or any subsequent version.

Starting 2026-11-10, the following AWS SDKs will use the Smithy RPC v2 CBOR protocol, while the other SDKs will continue using the AWS JSON protocol:

- AWS SDK for C++
- AWS SDK for Go V2
- AWS SDK for Java 2.x
- AWS SDK for .NET v4
- AWS Tools for PowerShell v5
- AWS SDK for Swift
- AWS SDK for Kotlin

What are the risks of enabling the Smithy RPC v2 CBOR protocol?

The added support of Smithy RPC v2 CBOR in the SDKs is expected to be fully transparent and effortless for customers. However, if you are using a custom implementation of the AWS SDK, or a combination of custom clients and the AWS SDK, that depends on AWS JSON-specific serialization behavior, it may be incompatible with the Smithy RPC v2 CBOR protocol.

What if my solution does not support Smithy RPC v2 CBOR?

To use the AWS JSON protocol, downgrade your SDK to a version released before 2026-11-10.

What AWS Regions support the Smithy RPC v2 CBOR protocol?

The Smithy RPC v2 CBOR protocol is supported in all AWS Regions where this service is available.

What latency improvements can I expect when upgrading to SDK versions that support Smithy RPC v2 CBOR?

Smithy RPC v2 CBOR provides up to 13% lower latency and up to 15% reduction in payload size compared to AWS JSON. The binary encoding eliminates the overhead of JSON text parsing and string escaping, which reduces both network transfer time and serialization cost.

Will the AWS JSON protocol be deprecated?

AWS JSON protocol will continue to be supported through older versions of SDKs. Also, some language-specific SDKs will continue using AWS JSON as it is more performant than Smithy RPC

v2 CBOR due to the available JSON implementation for the language being faster than its CBOR implementation.

Where can I find more information about Smithy RPC v2 CBOR?

You can find more information about the AWS JSON protocol at AWS JSON 1.0 protocol and AWS JSON 1.1 protocol in the Smithy documentation, and about Smithy RPC v2 CBOR at Smithy RPC v2 CBOR protocol in the Smithy documentation.

Document history for Amazon Transcribe

- **Latest documentation update:** 13 November 2023

The following table describes important changes in each release of Amazon Transcribe. For notification about updates to this documentation, you can subscribe to an RSS feed.

Change	Description	Date
Added FAQ page for supported protocols	Added a new FAQ page with answers to common questions about the AWS JSON and Smithy RPC v2 CBOR wire protocols. For more informati on, see FAQs on supported protocols.	November 10, 2026
Feature Update	Amazon Transcribe now supports customer-managed KMS keys to encrypt the artifacts associated with your custom vocabularies, custom vocabulary filters, and custom language models.	September 25, 2026
Feature Update	PII redaction for batch transcription jobs now supports additional English dialects: Australian (en-AU), British (en-GB), Indian (en-IN), Irish (en-IE), New Zealand (en-NZ), Scottish (en-AB), South African (en-ZA), and Welsh (en-WL).	July 1, 2026

Feature Update	PII redaction for batch transcription jobs now supports additional entity types: AGE, AWS_ACCESS_KEY , AWS_SECRET_KEY , CA_HEALTH_NUMBER , CA_SOCIAL_INSURANCE_NUMBER , DATE_TIME , DRIVER_ID , INTERNATIONAL_BANK_ACCOUNT_NUMBER , IP_ADDRESS , LICENSE_PLATE , MAC_ADDRESS , PASSPORT_NUMBER , PASSWORD, SWIFT_CODE , URL, US_INDIVIDUAL_TAX_IDENTIFICATION_NUMBER , USERNAME, and VEHICLE_IDENTIFICATION_NUMBER .	May 22, 2026
Feature Update	PII redaction for batch transcription jobs now supports additional languages: French (fr-FR), Canadian French (fr-CA), German (de-DE), Swiss German (de-CH), Italian (it-IT), Portuguese (pt-PT), and Brazilian Portuguese (pt-BR).	May 1, 2026

New feature	AWS HealthScribe now supports new templates for the clinical note summary: BIRP, SIRP, DAP, BH_SOAP, and PH_SOAP. For more information, see AWS HealthScribe Clinical Documentation file.	May 30, 2025
New feature	AWS HealthScribe now supports a GIRPP template for the clinical note summary. For more information, see AWS HealthScribe Clinical Documentation file.	February 4, 2025
New feature	AWS HealthScribe now supports streaming for transcribing medical conversations in real time.	January 29, 2025
Section Update	Updating ar-SA language support for 8000 frequency.	January 16, 2025
Section Update	Updating ar-SA language support for 8000 frequency.	January 16, 2025
Section Update	Updating the supported language section with unsupported language code that we do not support in AWS GovCloud (US) (US-West, us-gov-west-1), AWS GovCloud (US) (US-East, us-gov-east-1), or Africa (Cape Town, af-south-1) regions.	January 14, 2025

Section Update	Extend Transcribe Batch json output with new field called "audio_segments".	July 15, 2024
Feature Update	Update Max Speakers in diarization to be 30 instead of 10.	May 10, 2024
Section Update	Updates to generative call summarization and add error output details.	April 30, 2024
Section Update	Updates on custom vocabulary columns - IPA and SoundsLike.	April 30, 2024
Feature Update	Amazon Transcribe Call Analytics now supports generative call summarization.	November 29, 2023
Section Update	Update new PII redaction and Language Identification output format.	November 13, 2023
Feature Update	Diarization can now be combined with channel identification.	March 6, 2023
Feature Update	Channel identification can now be combined with diarization.	March 6, 2023
Section Update	IAM best practices have been updated.	February 13, 2023
New languages	Amazon Transcribe now supports Vietnamese and Swedish.	December 6, 2022

New feature	Amazon Transcribe now supports real-time Call Analytics.	November 28, 2022
Feature Update	Streaming redaction and identification is now available with Hindi and Thai.	November 11, 2022
Section Update	New PII categories are available for streaming redaction and identification.	September 14, 2022
Section Update	The custom language model section has been revised.	June 18, 2022
Section Update	Batch language identification can now identify multiple languages per audio file.	May 31, 2022
Guide Update	The Amazon Transcribe API Reference is now a standalone guide.	April 1, 2022
New chapter	A new SDK code examples chapter is included.	March 21, 2022
Feature update	Call Analytics now provides call summarization.	March 21, 2022
Chapter update	The introductory chapter now showcases Amazon Transcribe use cases.	March 21, 2022
Chapter update	The Getting started chapter has been updated to be method-specific.	March 21, 2022

Chapter update	The Streaming chapter has been updated and restructured.	March 21, 2022
Feature update	Language identification now supports custom vocabularies and custom vocabulary filters with streaming transcriptions.	March 11, 2022
New event	There is a new event type: Vocabulary events.	February 7, 2022
Section update	Updates have been made to the custom vocabularies section.	January 20, 2022
New feature	Language identification can now be used with streaming transcriptions.	November 23, 2021
New feature	Language identification can now be used with custom language models, custom vocabularies, vocabulary filtering, and content redaction.	October 29, 2021
New feature	Amazon Transcribe now supports custom language models with streaming transcriptions.	October 20, 2021
New feature	Amazon Transcribe can now generate subtitles for your video files.	September 16, 2021

New feature	Amazon Transcribe now supports PII redaction and identification for streaming.	September 14, 2021
New feature	Amazon Transcribe now supports AWS KMS encryptio n context for an added level of security for your AWS account resources.	September 10, 2021
New languages	Amazon Transcribe now supports Afrikaans, Danish, Mandarin Chinese (Traditio nal), Thai, New Zealand English, and South African English.	August 26, 2021
New feature	Amazon Transcribe now supports resource tagging.	August 24, 2021
New feature	Amazon Transcribe now supports Call Analytics for batch transcription jobs.	August 4, 2021
New feature	Amazon Transcribe now supports using custom vocabularies with batch custom language models.	May 12, 2021
New feature	Amazon Transcribe now supports partial result stabilization for streaming transcription.	May 11, 2021

New feature	Amazon Transcribe now supports Australian English, British English, Hindi, and US Spanish for custom language models.	March 19, 2021
New languages	Amazon Transcribe adds support for Italian and German for streaming audio transcription.	November 4, 2020
AWS Region expansion	Amazon Transcribe is now available in the Frankfurt (eu-central-1) and London (eu-west-2).	November 4, 2020
New feature	Amazon Transcribe adds support for interface VPC endpoints in batch transcrip tion.	October 9, 2020
New feature	Amazon Transcribe adds support for channel identific ation in streaming.	September 17, 2020
New feature	Amazon Transcribe adds support for automatic language identification in batch transcription.	September 15, 2020
New feature	Amazon Transcribe adds support for speaker partition ing in streaming.	August 19, 2020
New feature	Amazon Transcribe adds support for custom language models.	August 5, 2020

New feature	Amazon Transcribe adds support for interface VPC endpoints in streaming.	June 26, 2020
New feature	Amazon Transcribe adds support for vocabulary filtering in streaming.	May 20, 2020
New feature	Amazon Transcribe adds support for creating a custom vocabulary of words to filter from a transcription.	December 20, 2019
New feature	Amazon Transcribe adds support for queueing transcription jobs.	December 19, 2019
New languages	Amazon Transcribe adds support for Gulf Arabic, Hebrew, Japanese, Malay, Swiss German, Telugu, and Turkish.	November 21, 2019
AWS Region expansion	Amazon Transcribe is now available in the Asia Pacific (Tokyo) (ap-northeast-1).	November 21, 2019
New feature	Amazon Transcribe adds support for alternative transcriptions.	November 20, 2019
New languages	Amazon Transcribe adds support for Dutch, Farsi, Indonesian, Irish English, Portuguese, Scottish English, Tamil, and Welsh English.	November 12, 2019

New language	Amazon Transcribe now supports streaming transcription for Australian English (en-AU).	October 25, 2019
AWS Region expansion	Amazon Transcribe is now available in the China (Beijing) (cn-north-1) and China (Ningxia) (cn-northwest-1).	October 9, 2019
New feature	Amazon Transcribe allows you to provide your own KMS key to encrypt your transcription output files. For more information, see the OutputEncryptionKMSKeyId parameter of the StartStreamTranscription API.	September 24, 2019
New languages	Amazon Transcribe adds support for Chinese (Mandarin), Simplified, Mainland China and Russian.	August 23, 2019
New feature	Amazon Transcribe adds support for streaming audio transcription using the WebSocket protocol.	July 19, 2019
New feature	AWS CloudTrail now records events for the StartStreamTranscription API.	July 19, 2019
AWS Region expansion	Amazon Transcribe is now available in the US West (N. California) (us-west-1).	June 27, 2019

New language	Amazon Transcribe adds support for Modern Standard Arabic.	May 28, 2019
New feature	Amazon Transcribe now transcribes numeric words into numbers for US English. For example, "forty-two" is transcribed as "42".	May 23, 2019
New language	Amazon Transcribe adds support for Hindi and Indian English.	May 15, 2019
New SDK	The AWS SDK for C++ now supports Amazon Transcribe.	May 8, 2019
New language	Amazon Transcribe adds support for Spanish.	April 19, 2019
AWS Region expansion	Amazon Transcribe is now available in the EU (Frankfurt) (eu-central-1) and Asia Pacific (Seoul) (ap-northeast-2).	April 18, 2019
New language	Amazon Transcribe adds support for streaming transcription in British English, French, and Canadian French.	April 5, 2019
New feature	The AWS SDK for Ruby V3 now supports Amazon Transcribe	March 25, 2019

New feature	Amazon Transcribe allows custom vocabularies, which are lists of specific words that you want Amazon Transcrib e to recognize in your audio input.	March 25, 2019
New languages	Amazon Transcribe adds support for German and Korean.	March 22, 2019
New language	Amazon Transcribe now supports streaming transcrip tion for US Spanish (es-US).	February 7, 2019
AWS Region expansion	Amazon Transcribe is now available in the South America (São Paulo) (sa-east-1).	February 7, 2019
AWS Region expansion	Amazon Transcribe is now available in the Asia Pacific (Mumbai) (ap-south-1), Asia Pacific (Singapore) (ap-south east-1), EU (London) (eu-west-2), and EU (Paris) (eu-west3).	January 24, 2019
New languages	Amazon Transcribe adds support for French, Italian, and Brazilian Portuguese.	December 20, 2018
New feature	Amazon Transcribe now supports transcription of audio streams.	November 19, 2018

New languages	Amazon Transcribe adds support for Australian English, British English, and Canadian French.	November 15, 2018
AWS Region expansion	Amazon Transcribe is now available in Canada (Central) (ca-central-1) and Asia Pacific (Sydney) (ap-southeast-2).	July 17, 2018
New feature	You can now specify your own location to store the output from a transcription job.	July 11, 2018
New feature	Added AWS CloudTrail and Amazon CloudWatch Events integration.	June 28, 2018
New feature	Amazon Transcribe adds support for custom vocabularies.	April 4, 2018
New guide	This is the first release of the *Amazon Transcribe Developer Guide*.	November 29, 2017

AWS Glossary

For the latest AWS terminology, see the AWS glossary in the *AWS Glossary Reference*.