SPEECHMATICS

Batch Container 7.0.3

Table of Contents

Batch Container

Important Notices

The legacy V1 API and related output formats is no longer supported. V1 API examples have been removed from all batch container documentation. We recommend use of the V2 API and the config.json object documented in the Speech API. How to use the V2 API is documented within the Speech API document for 7.0.0.

What's New

7.0.3

- Internal bug fixes

7.0.2

- Internal bug fixes

7.0.1

- We have changed how some words submitted using a Custom Dictionary are recognised for all languages. This change will affect words that use a splitting character (e.g. COVID-19, catch-22). This change should provide more accurate transcription of such words.

7.0.0

- The V2 API is now the only supported method to transcribe files. The V1 API is no longer supported, and will be fully deprecated in an imminent release
- Updated English and Spanish language packs
- SubRip (srt) subtitle format. Customers may also modify how the SRT output is presented
- The working directory is no longer `/work`, but is instead `/home/smuser/work`
- The ability to pull files for transcription from an object store hosted by a cloud provider
- The ability to send notifications or callbacks to a customer-specified endpoint
- Users may also provide their own metadata information within a separate JSON file for better tracking and monitoring
- Users can cache one or many Custom Dictionaries within a shared cache location specified by themselves. This improves performance overhead when transcribing files using the same custom dictionary that has already been cached. **Users are responsible for managing their own cache**. How to do so is described in more detail in the Speech API Guide
- Users can run the container as a named user (e.g. not as root)

Issues Fixed

The following issues are addressed since the previous release:

Issue ID	Summary	Resolution Description
REQ-15418	Custom dictionary with splitting characters gets incorrect pronunciation	When using words with splitting characters in a Custom Dictionary (for example covid-19) where a number follows a word we now have the correct pronunciations created. Splitting characters include ["-", "_", "/", "<", ">", ":", " "]. This is for all languages **For v7.0.1 only**
REQ-13442	Some unicode characters would cause	This has now been resolved

	transcription to fail	
REQ-13990	The batch container will not run as a non-root user on Docker	This is now supported. Guidance on how to do this is in the Quick Start Guide
REQ-14062	Occasionally a file in Spanish would not be fully transcribed	This has been resolved with the latest release of Spanish

Known Limitations

Issue ID	Summary	Detailed Description and Possible Workarounds
REQ-1409	Proteus HCL with `<unk>` causes out of memory error	A custom dictionary list that contains the word '' causes the worker to crash.
REQ-10160	Advanced punctuation for Spanish (es) does not contain inverted marks.	Inverted marks [¿ ¡] are not currently available for Spanish advanced punctuation.
REQ-10627	Double full stops when acronym is at the end of the sentence	If there is an acronym at the end of the sentence, then a double full stop will be output, for example: "team G.B.."
REQ-11135	A previous release (6.1.0) introduced unwanted hesitations in transcripts.	Due to changes in the way that training data is now ingested to improve the accuracy of spontaneous speech for English (en) there is a greater likelihood that hesitations will be included in the output transcripts. We plan to support a hesitation filtering capability in a future release for customers that do not want to see hesitations on transcripts.

Supported Platforms

Docker (17.06.0+) running on Ubuntu, Debian, Fedora or CentOS.

Installation

Pull the Batch Container Docker image from the Speechmatics Docker repository.

Pre-requisites

You have a login (URL, username and password) for the Speechmatics Docker repository, and have a Docker environment (version 17.06.0 or above) running.

Related Documentation

- Speechmatics Batch Container Quick Start Guide version 7.0.0
- Speechmatics Batch Container API Guide version 7.0.0

For a complete list of languages that are supported by the Speechmatics Container, including those which have custom dictionary support, please go to the Speechmatics website: https://www.speechmatics.com/language-support/

Batch Container Quick Start Guide

This guide will walk you through the steps needed to deploy the Speechmatics Batch Container ready for transcription.

- Check system requirements
- Pull the Docker Image
- Run the Container

After these steps, the Docker Image can be used to create containers that will transcribe audio files. More information about using the Speechmatics container transcription service is detailed in the Speechmatics Container API guide.

System requirements

Speechmatics containerized deployments are built on the Docker platform. In order to operate the containers, the following requirements will need to be met.

Host requirements

An individual Docker image is required for each language transcription is required within. A single image can be used to create and run multiple containers concurrently, each running container will require the following resources:

- 1 vCPU
- 2-5GB RAM
- 100MB hard disk space

The host machine requires a processor with following minimum specification: Intel® Xeon® CPU E5-2630 v4 (Sandy Bridge) 2.20GHz (or equivalent). This is important because these chipsets (and later ones) support Advanced Vector Extensions (AVX). The machine learning algorithms used by Speechmatics ASR require the performance optimizations that AVX provides. You should also ensure that your hypervisor has AVX enabled.

Note: Each language pack required is distributed as a separate Docker image. Only the language packs required need to be installed on the Docker host.

Architecture

Each container:

- Provides the ability to transcribe recorded speech in a predefined language. The container will receive input from most audio and video formats, and will provide the following output:
  - Transcript word or punctuation
  - Word or punctuation confidence
  - License expiry information
  - Job configuration information
  - Metadata and tracking information if provided

    - Transcript word or punctuation
    - Word confidence
    - Timing information
    - Speaker change and labelling information
- Takes one input file and outputs the resulting transcript
- Can run in a mode that parallelises processing across multiple processor cores
- Can send notifications once a transcript has completed
- Can support pulling a file from an online location
- Supports input file sizes up to 2 hours in length or 4GB in size
- All data is transitory, once a container completes its transcription it removes all record of the operation, no data is persisted.

In addition, multiple instances of the container can be run on the same Docker host. This enables scaling of a single language or multiple-languages as required.

Accessing the Image

The Speechmatics Docker image is obtained from the Speechmatics Docker repository (jfrog.io). If you do not have a Speechmatics software repository account or have lost your details, please contact Speechmatics support support@speechmatics.com.

The latest information about the containers can be found in the solutions section of the support portal. If a support account is not available or the *Containers* section is not visible in the support portal, please contact Speechmatics support support@speechmatics.com for help.

Prior to pulling any Docker images, the following must be known:

- Speechmatics Docker URL – provided by the Speechmatics team
- Language Code – the ISO language code (for example `fr` for French)
- `LICENSE_KEY` - which is required to start a container
- `TAG` – which is used to identify the image version

Getting the Image

After gaining access to the relevant details for the Speechmatics software repository, follow the steps below to login and pull the Docker images that are required.

Software Repository Login

Ensure the *Speechmatics Docker URL* and software repository *username* and *password* are available. The endpoint being used will require Docker to be installed. For example:

```
docker login https://speechmatics-docker-example.jfrog.io
```

You will be prompted for username and password. If successful, you will see the response:

```
Login Succeeded
```

If unsuccessful, please verify your credentials and URL. If problems persist, please contact Speechmatics support.

Pulling the Image

To pull the Docker image to the local environment follow the instructions below. Each supported language pack comes as a different Docker image, so the process will need to be repeated for each language pack required.

Example: pulling Global English (en) with the 7.0.0 TAG:

```
docker pull speechmatics-docker-example.jfrog.io/transcriber-en:7.0.0
```

Example: pulling the Spanish (es) model with the 7.0.0 TAG:

```
docker pull speechmatics-docker-example.jfrog.io/transcriber-es:7.0.0
```

The image will start to download. This could take a while depending on your connection speed.

Note: Speechmatics require all customers to cache a copy of the Docker image(s) within their own environment. Please do not pull directly from the Speechmatics software repository for each deployment.

Licensing

The Docker images we provide have a configured expiry date and must be used in conjunction with the license key that has been issued to you. The Docker images and license key are specific to your organisation, and should not be shared with any third parties. License keys must be provided at runtime through a `LICENSE_KEY` environment value, like this:

```
-e LICENSE_KEY=f787b0051e2768b1f619d75faab97f23ee9b7931890c05f97e9f550702
```

Using the Container

Once the Docker image has been pulled into a local environment, it can be started using the Docker `run` command. More details about operating and managing the container are available in the Docker API documentation.

There are two different methods for passing an audio file into a container:

- STDIN: Streams audio file into the container though the standard command line entry point
- File Location: Pulls audio file from a file location

Here are some examples below to demonstrate these modes of operating the containers.

Example 1: passing a file using the `cat` command to the Spanish (es) container

```
cat ~/sm_audio.wav | docker run -i \
  -e LICENSE_KEY=f787b0051e2768b1f619d75faab97f23ee9b7931890c05f97e9f550702 \
  speechmatics-docker-example.jfrog.io/transcriber-es:7.0.0
```

Example 2: pulling an audio file from a volume-ma directory into the container

```
docker run -i -v ~/sm_audio.wav:/input.audio \
  -e LICENSE_KEY=f787b0051e2768b1f619d75faab97f23ee9b7931890c05f97e9f550702 \
  speechmatics-docker-example.jfrog.io/transcriber-es:7.0.0
```

NOTE: the audio file must be mapped into the container with `:/input.audio`

The Docker `run` options used are:

Name	Description
`--env, -e`	Set environment variables
`--interactive , -i`	Keep STDIN open even if not attached
`--volume , -v`	Bind mount a volume

See Docker docs for a full list of the available options.

Both the methods will produce the same transcribed outcome. STDOUT is used to provide the transcription in a JSON format. Here's an example:


```
{
  "format": "2.4",
  "license": "productsteam build (Thu May 14 14:33:09 2020): 953 days remaining",
  "metadata": {
    "created_at": "2020-06-30T15:43:50.871Z",
    "type": "transcription",
    "transcription_config": {
      "language": "en",
      "diarization": "none",
      "additional_vocab": [
        {
          "content": "Met Office"
        },
        {
          "content": "Fitzroy"
        },
        {
          "content": "Forties"
        }
      ]
    }
  },
  "results": [
    {
      "alternatives": [
        {
          "confidence": 1.0,
          "content": "Are",
          "language": "en",
          "speaker": "UU"
        }
      ],
      "end_time": 3.61,
      "start_time": 3.49,
      "type": "word"
      },
    {
      "alternatives": [
        {
          "confidence": 1.0,
          "content": "on",
          "language": "en",
          "speaker": "UU"
        }
      ],
      "end_time": 3.73,
      "start_time": 3.61,
      "type": "word"
      },
    {
      "alternatives": [
```

```
        {
          "confidence": 1.0,
          "content": "the",
          "language": "en",
          "speaker": "UU"
        }
      ],
      "end_time": 3.79,
      "start_time": 3.73,
      "type": "word"
      },
    {
      "alternatives": [
        {
          "confidence": 1.0,
          "content": "rise",
          "language": "en",
          "speaker": "UU"
        }
      ],
      "end_time": 4.27,
      "start_time": 3.79,
      "type": "word"
    }
  ]
}
```


Working Directory

The working directory is `home/smuser/work` now rather than `work`. This is the case whether running the container as a root or non-root user.

Determining success

The exit code of the container will determine if the transcription was successful. There are two exit code possibilities:

- Exit Code == 0 : The transcript was a success; the output will contain a JSON output defining the transcript (more info below)
- Exit Code != 0 : the output will contain a stack trace and other useful information. This output should be used in any communication with Speechmatics support to aid understanding and resolution of any problems that may occur

Troubleshooting

Enabling Logging

If you are seeing problems then we recommend that you enable logging and open a support ticket with Speechmatics support: support@speechmatics.com.

To enable logging you add two environment variables:

- `SM_JOB_ID` - a job id, for example: `1`
- `SM_LOG_DIR` - the directory inside the container where to write the logs, for example: `/logs`

The following example shows how to do this, using the `-stderr=true` argument to dump the logs to stderr:

```
docker run --rm -e SM_JOB_ID=123 -e SM_LOG_DIR=/logs \
  -v ~/sm_audio.wav:/input.audio \
  -e LICENSE_KEY=f787b0051e2768b1f619d75faab97f23ee9b7931890c05f97e9f550702 \
  speechmatics-docker-example.jfrog.io/transcriber-en:7.0.0 \
  -stderr=true
```

When raising a support ticket it is normally easier to write the log output to a specific file. You can do this by creating a volume mount where the logs will be accessible from after the container has finished. Before running the container you need to create a directory for the log file and ensure it has the correct permissions. In this example we use a local logs directory to store the output of the log for a job with ID 124:

```
mkdir -p logs/124
sudo chown -R nobody:nogroup logs/
sudo chmod -R a+rwx logs/

docker run --rm -v ${PWD}/logs:/logs -e SM_JOB_ID=124 -e SM_JOB_ID=/logs \
  -v ~/sm_audio.wav:/input.audio \
  -e LICENSE_KEY=f787b0051e2768b1f619d75faab97f23ee9b7931890c05f97e9f550702 \
  speechmatics-docker-example.jfrog.io/transcriber-en:7.0.0
tail logs/124/sigurd.log
```

Common Problems

There are occassions where the transcription container will fail to transcribe the media file provided and will exit without error code 0 (success). Speechmatics heavily advise enabling logging (see instruction above). The logs will show some of the reasons for the failed job especially when multiple errors can cause the same error code. Below are some errors with suggestions and how they can be revolved.

Error Code	Error	Resolution

1	“err: signal: illegal instruction”	This means that the models couldn’t be loaded within the container. Please ensure that the host that’s running the Docker engine has an AVX compatible CPU.
		
		The following can also be done inside the container to check that AVX is listed in the CPU flags.
		
		$ docker run -it --entrypoint /bin/bash speechmatics-docker-example.jfrog.io/transcriber-en:7.0.0
		
		$ cat /proc/cpuinfo
			grep flags

1	“Unable to set up logging”	This can occur when a directory is volume mapped into the containers and a log file cannot be created into that directory. Example command to map in a tmp directory inside the container to /xxx path: $ docker run --rm -e SM_LOG_DIR=/xxx -e SM_JOB_ID=1 -v $PWD/tmp:/xxx speechmatics-docker-example.jfrog.io/transcriber-en:7.0.0
1	“err: licensing failed”	This generally occurs if either no license key or the wrong key is supplied. Use the license key provided by Speechmatics. Example command:

		$ docker run -i –v /home/user/config.json:/config.json -v /home/user/example.wav:/input.audio -e LICENSE_KEY= f787b0051e2768b1f619d75faab97f23ee9b7931890c05f97e9f550702 speechmatics-docker-example.jfrog.io/transcriber-en:7.0.0 --stderr
1	“/input.audio is not valid”	If volume mapping the file into the container, ensure that a valid audio file is being mapped in.
1	“failed to get sample rate”	The sample rate from the audio file that was passed for recognition did not have a sample rate. Check the audio file is valid and that a sample rate can be read. The following ffmpeg can be used to identify it there is a valid sample rate: $ ffmpeg -i /home/user/example.wav
1	“exit status 1”	If the container is memory (RAM) starved it can quit during the transcription process. Verify the minimum resource (CPU and RAM) requirements are being assigned to a transcription container. The inspect command in docker can be useful to identify if the lack of memory shutdown the container. Look out for the “OOMKilled” value. Here is an example. . $ docker inspect --format='{{json .State}}' $containerID
2	“The value of --parallel must be >= 1, but 0 was supplied” OR “invalid value "--stderr" for flag --parallel: parse error”	If using the parallel option to speed up the processing time on files more than 5 minutes in length the ––parallel switch needs to have an integer at least 1. A non-zero value must be provided if the parallel command is to be used. The example below shows a valid command: $ docker run -i –v /home/user/config.json:/config.json -v /home/user/example.wav:/input.audio -e LICENSE_KEY= f787b0051e2768b1f619d75faab97f23ee9b7931890c05f97e9f550702 speechmatics-docker-example.jfrog.io/transcriber-en:7.0.0 --parallel 2

If you still continue to face issues, please contact Speechmatics support support@speechmatics.com.

Building an Image

Using STDIN to pass files in and obtain the transcription may not be sufficient for all use cases. It is possible to build a new Docker Image that will use the Speechmatics Image as a layer. This will allow greater flexibility and a mechanism to fit into custom workflows. To include the Speechmatics Docker Image inside another image, ensure to add the pulled Docker image into the Dockerfile for the new application.

Requirements for a custom image

To ensure the Speechmatics Docker image works as expected inside the custom image, please consider the following:

- Any audio that needs to be transcribed must to be copied to a file called "/input.audio" inside the running container
- To initiate transcription, call the application `pipeline`. The pipeline will start the transcription service and use `/input.audio` as the audio source

- Once pipeline finishes transcribing, ensure you move the transcription data outside the container
- Shutdown the container after each transcription of an audio file

Dockerfile

To add a Speechmatics Docker image into a custom one, the Dockerfile must be modified to include the full image name of the locally available image.

Example: Adding Global English (en) with tag 7.0.0 to the Dockerfile

```
FROM speechmatics-docker-example.jfrog.io/transcriber-en:7.0.0
ADD download_audio.sh /usr/local/bin/download_audio.sh
RUN chmod +x /usr/local/bin/download_audio.sh
CMD ["/usr/local/bin/download_audio.sh"]
```

Once the above image is built, and a container instantiated from it, a script called `download_audio.sh` will be executed (this could do something like pulling a file from a webserver and copying it to `/input.audio` before starting the pipeline application). This is a very basic Dockerfile to demonstrate a way of orchestrating the Speechmatics Docker Image.

NOTE: For support purposes, it is assumed the Docker Image provided by Speechmatics has been unmodified. If you experience issues, Speechmatics support will require you to replicate the issues with the unmodified Docker image e.g. `speechmatics-docker-example.jfrog.io/transcriber-en:7.0.0`

Running a batch container as a non-root user

There are some use cases where you may not be able to run the batch container as a root user. This may be because you are working in a hosting environment that mandates the use of a named user rather than root.

You must start the container with the command `docker run -user $USERNUMBER:$GROUPID`. User number and group ID are non-zero numerical values from a value of **1** up to a value of **65535**. So a valid example would be:

```
docker run -user 1000:3000.
```

Getting Transcription Output as a non-root user

If you take transcription via the default STDOUT, then this will not change as a non-root user. An example is below:

```
docker run -u 1020:4000 \
 -v /Users/$USER/work/pipeline/mydev/config.json:/config.json \
 -v /Users/$USER/work/pipeline/mydev/input.audio:/input.audio \
    ${IMAGE_NAME}
```

If you want to map the output to a specific directory, you must volume map a directory to which a non-root user would have access.

Running a Batch Container as a non-root user on Kubernetes

Please Note The examples below **do not** constitute an explicit recommendation to run as non-root user, merely a guideline on how to do so with Kubernetes only where this is an unavoidable requirement.

If you require named users to be deployed on Kubernetes Pods, you must set the following Security Config. The user and group **must** correspond to the user and group you use when starting the container

```
securityContext:

  runAsUser: {non-zero numerical value between 0 and 65535}
  runAsGroup: {non-zero numerical value between 0 and 65535}
```

There is more information on how to configure security settings on Kubernetes pods here

Some Kubernetes deployments may mandate the use of PodSecurity Admissions Controllers. These provide stricter security requirements. More information on them can be found here. If your deployment does require this set up, here is an example configuration that would allow you to carry out transcription as a non-root user.

```
apiVersion: policy/v1beta1
kind: PodSecurityPolicy
metadata:
  name: restricted
  annotations:
    seccomp.security.alpha.kubernetes.io/allowedProfileNames:
'docker/default,runtime/default'
    apparmor.security.beta.kubernetes.io/allowedProfileNames: 'runtime/default'
    seccomp.security.alpha.kubernetes.io/defaultProfileName:  'runtime/default'
    apparmor.security.beta.kubernetes.io/defaultProfileName:  'runtime/default'
spec:
  privileged: false
  # Required to prevent escalations to root.
  allowPrivilegeEscalation: false
  requiredDropCapabilities:
    - ALL
  # Allow core volume types.
  volumes:
    - 'configMap'
    - 'emptyDir'
    - 'projected'
    - 'secret'
    - 'downwardAPI'
    # Assume that persistentVolumes set up by the cluster admin are safe to use.
    - 'persistentVolumeClaim'
  hostNetwork: false
  hostIPC: false
  hostPID: false
  runAsUser:
    # Require the container to run without root privileges.
    rule: 'MustRunAsNonRoot'
  seLinux:
    # This policy assumes the nodes are using AppArmor rather than SELinux.
    rule: 'RunAsAny'
  supplementalGroups:
    rule: 'MustRunAs'
    ranges:
      # Forbid adding the root group.
      - min: 1
        max: 65535
  fsGroup:
    rule: 'MustRunAs'
    ranges:
      # Forbid adding the root group.
      - min: 1
        max: 65535
  readOnlyRootFilesystem: false
```

Batch Container API Guide

This guide will walk you through using Speechmatics' v2.4 API in order to invoke features of the Speechmatics Batch Container.

For information on getting started and accessing the Speechmatics software repository please refer to Speechmatics Container Quick Start Guide.

Transcription Output Format

The transcript output will consist of:

- JSON format version (examples can be seen in the sections below)
  - V2.4 - used when the `config.json` configuration object is used (only supported approach)
- Diarization information
  - Channel Diarization - channel labelling with relevant transcription in enclosed block
  - Speaker Diarization - speaker identification, speakers (F# and M#) are identified at the beginning of the transcript with timing information
  - Speaker Change - Identifying when different speakers as an element in the JSON output
  - Speaker Change with Channel Diarization - Channel labelling with relevant transcription in enclosed block, speaker change elements additionally output at relevant sections
  - No diarization - unidentified speakers will be displayed as "UU" at the beginning of the transcript
- Header information to show license expiry date
- A full stop to delimit sentences, irrespective of language being transcribed
- A word, confidence and timing information for each transcribed word
- Transcription output additionally in txt or srt format
- Notification information that can be used to generate callbacks
- Metadata about the job that was submitted as part of an optional `jobInfo` file

Feature Usage

This section explains how to use additional features beyond plain transcription of speech to text.

As part of the Speechmatics' V2.4 API, you must use the `config.json` object unless otherwise specified in examples below

Please Note the V1 API is no longer maintained. Using environmental variables to call speech features is neither recommended nor supported except where this document explicitly designates.

Configuration Object

The config object, if used, is a JSON structure that is passed as a separate volume-mapped file (mapped to `/config.json`) when carrying out transcription like this:

```
docker run -i -v ~/Projects/ba-test/data/shipping-forecast.wav:/input.audio \
  -v ~/tmp/config.json:/config.json \
  speechmatics-docker-example.jfrog.io/transcriber-en:7.0.0
```

Here is a simple example of a config object file (`~/tmp/config.json` from the above example). It requests transcription in English and lists additional custom dictionary words as part of the `additional_vocab` property:

```
{
  "type": "transcription",
  "transcription_config": {
```

```
    "language": "en",
    "additional_vocab": ["Met Office", "Fitzroy", "Forties"]
  }
}
```

The transcript output will also show the configuration information within the `config.json` file, as shown below:

```
{
  "format": "2.4",
  "license": "productsteam build (Thu May 14 14:33:09 2020): 953 days remaining",
  "metadata": {
    "created_at": "2019-03-01T17:21:34.002Z",
    "type": "transcription",
    "transcription_config": {
      "language": "en",
      "diarization": "none",
      "additional_vocab": [
        {
          "content": "Met Office"
        },
        {
          "content": "Fitzroy"
        },
        {
          "content": "Forties"
        }
      ]
    }
  },
```

Diarization

Diarization is the ability to identify a speaker in an audio file. This identification is only related to a single audio file only. For audio files that contain multiple channels or streams, it is possible to use **channel diarization** and apply custom labels to each channel or stream. If your audio file contains only a single channel or stream then you should choose **speaker diarization**. By default, containers will transcribe a file with diarization *disabled*. In the JSON output, files without diarisation requested will always show the speaker as 'UU'. Users can also use **speaker_change** to allow changes in the speaker to be detected and then marked in the transcript. Detection of speaker change is done without identifying which segments were spoken by the same speaker. Users can also combine **speaker_change** with **channel diarization** to identify both channel and speaker changes.

Note: Enabling diarization increases the amount of time taken to transcribe an audio file. The amount of time will vary depending on the length of the file.

Speaker Diarization

To enable speaker diarization the following must be set when you are using the config object:

```
{
  "type": "transcription",
  "transcription_config": {
    "language": "en",
    "diarization": "speaker"
  }
}
```

When enabled, the output will contain the speaker Identifiers, these are explained below:

- `M#` - Identifies a male speaker. The # will be a number identifying an individual male speaker
- `F#` - identified a female speaker. The # will be a number identifying an individual female speaker
- `UU` - Speaker is not identified (or diarization is disabled)

The example below shows relevant parts of a transcript with 3 male speakers. The output shows the configuration information passed in the `config.json` object and relevant segments with the different speakers in the JSON output. Only part of the transcript is shown here to highlight how different speakers are displayed in the output.


```
"format": "2.4",
"license": "productsteam build (Thu May 14 14:33:09 2020): 953 days remaining",
  "metadata": {
    "created_at": "2020-07-01T13:26:48.467Z",
    "type": "transcription",
    "transcription_config": {
      "language": "en",
      "diarization": "speaker"
      }
  },
 "results": [
    {
      "alternatives": [
        {
          "confidence": 0.93,
          "content": "You",
          "language": "en",
          "speaker": "M2"
        }
      ],
      "end_time": 0.51,
      "start_time": 0.36,
      "type": "word"
    },
    {
      "alternatives": [
        {
          "confidence": 1.0,
          "content": "When",
          "language": "en",
          "speaker": "M1"
        }
      ],
      "end_time": 12.6,
      "start_time": 12.27,
      "type": "word"
    },
    {
      "alternatives": [
        {
          "confidence": 1.0,
          "content": "And",
          "language": "en",
          "speaker": "M3"
```

```
        }
      ],
      "end_time": 80.63,
      "start_time": 80.48,
      "type": "word"
    }
```

In our JSON output, `start_time` identifies when a person starts speaking and `end_time` identifies when they finish speaking.

Channel Diarization

Channel diarization allows individual channels in an audio file to be labelled. This is ideal for audio files with multiple channels (up to 6). By default the feature is disabled. The following information is required within the `config.json` object to enable channel diarization on a 2-channel file that will use labels `Customer` and `Agent`:

```
{
  "type": "transcription",
  "transcription_config": {
    "language": "en",
    "diarization": "channel",
    "channel_diarization_labels": ["Customer", "Agent"]
  }
}
```

If the config object file is called `config.json` then you would start the transcription job like this:

```
docker run -i -v ~/Projects/ba-test/data/shipping-forecast.wav:/input.audio \
  -v ~/tmp/config.json:/config.json \
  speechmatics-docker-example.jfrog.io/transcriber-en:7.0.0
```

For each named channel, the words will be listed in its own labelled block, for example:

```
  {
  "format": "2.4",
  "license": "productsteam build (Thu May 14 14:33:09 2020): 953 days remaining",
  "metadata": {
    "created_at": "2020-07-01T14:11:43.534Z",
    "type": "transcription",
    "transcription_config": {
      "language": "en",
      "diarization": "channel"
    }
  },
  "results": [
    {
      "alternatives": [
        {
          "confidence": 0.87,
          "content": "I",
          "language": "en"
        }
      ],
```

```
    "channel": "channel_1",
    "end_time": 14.34,
    "start_time": 14.21,
    "type": "word"
  },
  {
    "alternatives": [
      {
        "confidence": 0.87,
        "content": "would",
        "language": "en"
      }
    ],
    "channel": "channel_1",
    "end_time": 14.62,
    "start_time": 14.42,
    "type": "word"
  },
  {
    "alternatives": [
      {
        "confidence": 0.87,
        "content": "love",
        "language": "en"
      }
    ],
    "channel": "channel_1",
    "end_time": 15.14,
    "start_time": 14.71,
    "type": "word"
  },
  {
    "alternatives": [
      {
        "confidence": 0.79,
        "content": "to",
        "language": "en"
      }
    ],
    "channel": "channel_1",
    "end_time": 16.71,
    "start_time": 16.3,
    "type": "word"
  },
  {
    "alternatives": [
      {
        "confidence": 0.67,
        "content": "To",
        "language": "en"
      }
    ],
    "channel": "channel_2",
    "end_time": 10.39,
```

```
      "start_time": 10.17,
      "type": "word"
    },
    {
      "alternatives": [
        {
          "confidence": 0.64,
          "content": "the",
          "language": "en"
        }
      ],
      "channel": "channel_2",
      "end_time": 10.68,
      "start_time": 10.52,
      "type": "word"
    },
    {
      "alternatives": [
        {
          "confidence": 0.71,
          "content": "unknown",
          "language": "en"
        }
      ],
      "channel": "channel_2",
      "end_time": 11.27,
      "start_time": 10.75,
      "type": "word"
    }
```


Note:

- Transcript output is provided sequentially **by channel**. So if you have two channels, all of channel 1 would be output first, followed by all of channel 2, and so on
- You must choose either channel or speaker diarization, you cannot choose both
- If you specify `channel` as a diarisation option, and do not assign channel_diarization_labels then default labels will be used (channel_1, channel_2 etc)
- Spaces cannot be used in the channel labels

Speaker Change Detection

This feature allows changes in the speaker to be detected and then marked in the transcript. Typically it is used to make some changes in the user interface to indicate to the reader that someone else is talking. Detection of speaker change is done without detecting which segments were spoken by the same speaker. The config used to request speaker change detection looks like this:

```
{
  "type": "transcription",
  "transcription_config": {
    "diarization": "speaker_change",
    "speaker_change_sensitivity": 0.8
  }
}
```

Note: Speaker change is only recorded as JSON V2 output, so make sure you use the `json-v2` format when you retrieve the transcript.

The `speaker_change_sensitivity` property, if used, must be a numeric value between 0 and 1. It indicates to the algorithm how sensitive to speaker change events you want to make it. A low value will mean that very few changes will be signalled (with higher possibility of false negatives), whilst a high value will mean you will see more changes in the output (with higher possibility of false positives). If this property is not specified, a default of 0.4 is used.

Speaker change elements in the `results` array appear like this:

```
{
  "type": "speaker_change",
  "start_time": 0.55,
  "end_time": 0.55,
  "alternatives": []
}
```

Note: Although there is an `alternatives` property in the speaker change element it is always empty, and can be ignored. The `start_time` and `end_time` properties are always identical, and provide the time when the change was detected.

A speaker change indicates where we think a different person has started talking. For example, if one person says "Hello James" and the other responds with "Hi", there should be a `speaker_change` element between "James" and "Hi", for example:

```
{
  "format": "2.4",
  "job": {
....
  "results": [
    {
      "start_time": 0.1,
      "end_time": 0.22,
      "type": "word",
      "alternatives": [
        {
            "confidence": 0.71,
            "content": "Hello",
            "language": "en",
            "speaker": "UU"
        }
      ]
    },
    {
      "start_time": 0.22,
      "end_time": 0.55,
      "type": "word",
      "alternatives": [
        {
            "confidence": 0.71,
            "content": "James",
            "language": "en",
            "speaker": "UU"
        }
      ]
    },
    {
```

```
      "start_time": 0.55,
      "end_time": 0.55,
      "type": "speaker_change",
      "alternatives": []
    },
    {
      "start_time": 0.56,
      "end_time": 0.61,
      "type": "word",
      "alternatives": [
        {
            "confidence": 0.71,
            "content": "Hi",
            "language": "en",
            "speaker": "UU"
        }
      ]
    }
  ]
}
```

- Note: You can only choose **speaker_change** as an alternative to **speaker** or **channel** diarization.

Speaker Change Detection With Channel Diarization

Speaker change can be combined with channel diarization. It will process channels separately and indicate in the output both the channels and the speaker changes. For example, if a two-channel audio contains two people greeting each other (both recorded over the same channel), the config submitted with the audio can request the speaker change detection:

```
{
  "type": "transcription",
  "transcription_config": {
    "diarization": "channel_and_speaker_change",
    "speaker_change_sensitivity": 0.8
  }
}
```

The output will have special elements in the `results` array between two words where a different person starts talking. For example, if one person says "Hello James" and the other responds with "Hi", there will a `speaker_change` json element between "James" and "Hi".

```
{
    "format": "2.4",
    "job": {
....
    },
    "metadata": {
....
    },
    "results": [
    {
      "channel": "channel_1",
      "start_time": 0.1,
      "end_time": 0.22,
```

```
      "type": "word",
      "alternatives": [
        {
            "confidence": 0.71,
            "content": "Hello",
            "language": "en",
            "speaker": "UU"
        }
      ]
    },
    {
      "channel": "channel_1",
      "start_time": 0.22,
      "end_time": 0.55,
      "type": "word",
      "alternatives": [
        {
            "confidence": 0.71,
            "content": "James",
            "language": "en",
            "speaker": "UU"
        }
      ]
    },
    {
      "channel": "channel_1",
      "start_time": 0.55,
      "end_time": 0.55,
      "type": "speaker_change",
      "alternatives": []
    },
    {
      "channel": "channel_1",
      "start_time": 0.56,
      "end_time": 0.61,
      "type": "word",
      "alternatives": [
        {
            "confidence": 0.71,
            "content": "Hi",
            "language": "en",
            "speaker": "UU"
        }
      ]
    }
  ]
}
```


- Note: Do not try to request **speaker_change** and **channel diarization** as multiple options: only **channel_and_speaker_change** is an accepted parameter for this output.

Custom Dictionary

This allows a custom dictionary wordlist to be added to the container at runtime. Having additional words can improve the likelihood it will be output in the final transcription. For any audio file being transcribed one custom

dictionary can be provided.

Prior to using this feature, consider the following requirements:

- File encoding must be UTF-8
- Maximum number of words or phrases in the list is 1000
- Each word or phrase needs to be on an individual line
- You should remove empty lines, and lines that contain only a hyphen ('-') character. You should not include hyphens within the custom dictionary
- If want to add acronyms (for example 'CEO'), you should use the sounds feature (see below)
- Including the word does not always ensure it will be recognised. We recommend use of the `sounds_like` feature to increase the likelihood of recognition if this is the case

To enable this feature, you use the `additional_vocab` property of the config object:

```
{
  "type": "transcription",
  "transcription_config": {
    "language": "en",
    "additional_vocab": [
      "speechmagic",
      "supercalifragilisticexpialidocious",
      "Techcrunch",
      "Yahoo! Answers"
    ]
  }
}
```

The Custom Dictionary feature supports the `sounds_like` extension that allows you to pass alternate pronunciations to words. For example the phrases "North Utsire" and "South Utsire" could be added as follows:

```
{
  "type": "transcription",
  "transcription_config": {
    "language": "en",
    "additional_vocab": [
      { "content": "North Utsire", "sounds_like": ["North at Sierra"]},
      { "content": "South Utsire", "sounds_like": ["South at Sierra"]},
      "Fitzroy",
      "Forties",
      { "content": "CEO", "sounds_like": ["C.E.O."]}
    ]
  }
}
```

You can see the custom dictionary entries in the transcription output below as well

Example response:

```
{
  "format": "2.4",
  "metadata": {
    "created_at": "2020-07-01T14:36:15.297Z",
    "type": "transcription",
    "transcription_config": {
      "language": "en",
```

```
      "diarization": "none",
      "additional_vocab": [
        {
          "content": "North Utsire",
          "sounds_like": [
            "North at Sierra"
          ]
        },
        {
          "content": "South Utsire",
          "sounds_like": [
            "South at Sierra"
          ]
        },
        {
          "content": "Fitzroy"
        },
        {
          "content": "Forties"
        },
        {
          "content": "CEO",
          "sounds_like": [
            "C.E.O."
          ]
        }
      ]
    }
  }
```

Note: `additional_vocab` items that are multi-word phrases will be output as a single word (e.g. Yahoo! Answers would be a single `content` item rather than two)

Using the Shared Custom Dictionary Cache

Processing a large custom dictionary repeatedly can be CPU consuming and inefficient. The Speechmatics Batch Container includes a cache mechanism for custom dictionaries to limit excessive resource use. By using this cache mechanism, the container can reduce the overall time needed for speech transcription when repeatedly using the same custom dictionaries. You will see performance benefits on re-using the same custom dictionary from the second time onwards.

It is not a requirement to use the shared cache to use the Custom Dictionary.

The cache volume is safe to use from multiple containers concurrently if the operating system and its filesystem support file locking operations. The cache can store multiple custom dictionaries in any language used for batch transcription. It can support multiple custom dictionaries in the same language.

If a custom dictionary is small enough to be stored within the cache volume, this will take place automatically if the shared cache is specified.

For more information about how the shared cache storage management works, please see **Maintaining the Shared Cache**.

We highly recommend you ensure any location you use for the shared cache has enough space for the number of custom dictionaries you plan to allocate there. How to allocate custom dictionaries to the shared cache is documented below.

How to set up the Shared Cache

The shared cache is enabled by setting the following value when running transcription:

- Cache Location: You must volume map the directory location you plan to use as the shared cache to `/cache` when submitting a job
- `SM_CUSTOM_DICTIONARY_CACHE_TYPE`: (mandatory if using the shared cache) This environment variable must be set to `shared`
- `SM_CUSTOM_DICTIONARY_CACHE_ENTRY_MAX_SIZE`: (optional if using the shared cache). This determines the maximum size of any single custom dictionary that can be stored within the shared cache in **bytes**
  - E.G. a `SM_CUSTOM_DICTIONARY_CACHE_ENTRY_MAX_SIZE` with a value of 10000000 would set a total storage size of **10MB**
  - For reference a custom dictionary wordlist with 1000 words produces a cache entry of size around 200 kB, or **200000** bytes
  - A value of `-1` will allow **every** custom dictionary to be stored within the shared cache. This is the **default** assumed value
  - A custom dictionary cache entry **larger** than the `SM_CUSTOM_DICTIONARY_CACHE_ENTRY_MAX_SIZE` will still be used in transcription, but will not be cached

Maintaining the Shared Cache

If you specify the shared cache to be used and your custom dictionary is within the permitted size, Speechmatics Batch Container will always try to cache the custom dictionary. If a custom dictionary cannot occupy the shared cache due to other cached custom dictionaries within the allocated cache, then older custom dictionaries will be removed from the cache to free up as much space as necessary for the new custom dictionary. This is carried out in order of the least recent custom dictionary to be used.

Therefore, you must ensure your cache allocation large enough to handle the number of custom dictionaries you plan to store. We recommend a relatively large cache to avoid this situation if you are processing multiple custom dictionaries using the batch container (e.g 50 MB). If you don't allocate sufficient storage this could mean one or multiple custom dictionaries are deleted when you are trying to store a new custom dictionary.

It is recommended to use a docker volume with a dedicated filesystem with a limited size. If a user decides to use a volume that shares filesystem with the host, it is the user's responsibility to purge the cache if necessary.

Creating the Shared Cache

In the example below, transcription is run where an example local docker volume is created for the shared cache. It will allow a custom dictionary of up to 5MB to be cached.

```
docker volume create speechmatics-cache

docker run -i -v /home/user/sm_audio.wav:/input.audio \
  -v /home/user/config.json:/config.json:ro \
  -e SM_CUSTOM_DICTIONARY_CACHE_TYPE=shared \
  -e SM_CUSTOM_DICTIONARY_CACHE_ENTRY_MAX_SIZE=5000000 \
  -v speechmatics-cache:/cache \
  -e LICENSE_KEY=f787b0051e2768bcee3231f619d75faab97f23ee9b7931890c05f97e9f550702 \
  speechmatics-docker-example.jfrog.io/transcriber-en:7.0.0
```

Viewing the Shared Cache

If all set correctly and the cache was used for the first time, a single entry in the cache should be present.

The following example shows how to check what Custom Dictionaries are stored within the cache. This will show the **language**, the **sampling rate**, and the **checksum** value of the cached dictionary entries.

```
ls $(docker inspect -f "{{.Mountpoint}}" speechmatics-cache)/custom_dictionary
en,16kHz,db2dd9c0d10faa8006d8a3fabc86aef6b6e27b3ccbd2a945d3aae791c627f0c5
```

Reducing the Shared Cache Size

Cache size can be reduced by removing some or all cache entries.

```
rm -rf $(docker inspect -f "{{.Mountpoint}}" speechmatics-cache)/custom_dictionary/*
```

> ***[NOTE] Manually purging the cache***
>
> *Before manually purging the cache, ensure that no containers have the volume mounted, otherwise an error during transcription might occur. Consider creating a new docker volume as a temporary cache while performing purging maintenance on the cache.*

Output Locale

It is possible to optionally specify the language locale to be used when generating the transcription output, so that words are spelled correctly, for cases where the model language is generic and doesn't already imply the locale.

Currently, Global English is the only language pack that supports different output locales. The following locales are supported:

- en-AU: supports Australian English
- en-GB: supports British English
- en-US: supports American English

The `output_locale` configuration setting is used for this. As an example, the following configuration uses the Global English (en) language pack with an output locale of British English (en-GB):

```
{
  "type": "transcription",
  "transcription_config": {
    "language": "en",
    "output_locale": "en-GB"
  }
}
```

Advanced Punctuation

Some language models now support advanced punctuation. This uses machine learning techniques to add in more naturalistic punctuation to make the transcript more readable. As well as putting punctuation marks in more naturalistic positions in the output, additional punctuation marks such as commas (,) and exclamation and question marks (!, ?) will also appear.

There is no need to explicitly enable this in the job configuration; languages that support advanced punctuation will automatically output these marks. If you do not want to see these punctuation marks in the output, then you can explicitly control this through the `punctuation_overrides` settings in the config.json file, for example:

```
{
  "type": "transcription",
  "transcription_config": {
    "language": "en",
    "punctuation_overrides": {
      "permitted_marks": [".", ","]
    }
```

```
  }
}
```

Both plain text and JSON output supports punctuation. JSON output places punctuation marks in the results list marked with a `type` of `"punctuation"`. So you can also filter on the output if you want to modify or remove punctuation.

A sample JSON output containing punctuation looks like this:

```
{
  "alternatives": [
    {
      "confidence": 1,
      "content": ",",
      "language": "en",
      "speaker": "UU"
    }
  ],
  "attaches_to": "previous",
  "end_time": 10.15,
  "is_eos": false,
  "start_time": 10.15,
  "type": "punctuation"
}
```

Note: Advanced punctuation is a V2 feature so, only the V2 output format will show advanced punctuation marks.

`is_eos_` is a parameter only passed in the transcription output when Advanced punctuation is used. EOS stands for 'end of sentence' and will only give a Boolean value of either true or false.

If you specify the punctuation_overrides element for languages that do not yet support advanced punctuation, then it will be ignored.

Notifications

Speechmatics allows customers to receive callbacks to a web service they control. Speechmatics will then make a HTTP POST request once the transcription is available. If you wish to enable notifications, you must add the `notification_config` only as part of the `config.json` object. This is separate to the transcription_config. The following parameters are available:

- `url` : **(mandatory)** The URL to which a notification message will be sent upon completion of the job. If `contents` is empty, then the body of the message will be empty
- `contents` : **(optional)** Specifies a list of item(s) to be attached to the notification message. If only one item is listed, it will be sent as the body of the request with Content-Type set to an appropriate value such as `application/octet-stream` or `application/json.` If multiple items are listed they will be sent as named file attachments using the multipart content type. Examples of what can be sent include the following:
  - `data` : The audio file submitted for the job.
  - `jobinfo` : A summary of the job. This will **only** be provided if you provide a jobinfo.json file when submitting a file for transcription. Please see the relevant section for information
  - `transcript.json-v2` : The transcript in `json-v2` format.
  - `transcript.txt` : The transcript in `txt` format.
  - `transcript.srt` : The transcript in `srt` format.
- `method` : **(optional)** the method to be used with HTTP and HTTPS URLs. If no option is chosen, the default is **POST**. PUT is now supported to allow uploading of content directly to an object store such as

S3.

- `auth_headers` : **(optional)** A list of additional headers to be added to the notification request when using http or https. This is intended to support authentication or authorization, for example by supplying an OAuth2 bearer token.

If you want to upload content directly to an object store, for example Amazon S3, you **must** ensure that the URL grants the Speechmatics container appropriate permissions when carrying out notifications. Pre-authenticated URLs, generated by an authorsed user, allow non-trusted devices access to upload to access stores. AWS carries this out via generating pre-signed URLs. Microsoft Azure allows similar acess via Shared Access Signatures.

Please see the section [How to transcribe files stored online](### How to transcribe files stored online) for details of how to pull files from online storage locations for transcription, and more information on pre-authenticated URLs

An example request for transcription in English with `notification_config` is shown below:

```
{
     "type": "transcription",
     "transcription_config": { "language": "en" },
     "notification_config": [
       {
         "url": "https://collector.example.org/callback",
         "contents": [ "transcript", "data" ],
         "auth_headers": ["Authorization: Bearer eyJ0eXAiOiJKV1QiLCJhb"]
       }
     ]
   }
```

If the callback is unsuccessful, it will repeat up to three times in total. If, after three times, it is still unsuccessful, it will process only the transcript via STDOUT.

How to generate multiple transcript formats

In addition to our primary JSON format, the Speechmatics container can output transcripts in the plain text (TXT) and SubRip (SRT) subtitle format. This can be done by using `--allformats` command and then specifying `<$EXAMPLE_DIRECTORY>` parameter within the transcription request. The `<$EXAMPLE_DIRECTORY>` is where **all** supported transcript formats will be saved. Users can also use `--all-formats` to generate the same response.

This directory must be mounted into the container so the transcripts can be retrieved after container finishes. You will receive a transcript in all currently supported formats: JSON, TXT, and SRT.

The following example shows how to use `--allformats` parameter. In this scenario, after processing the file, three separate transcripts would be found in the `~/tmp/output` directory. These transcripts would be in JSON, TXT, and SRT format.

```
docker run \
  -v ~/Projects/ba-test/data/shipping-forecast.wav:/input.audio \
  -v ~/tmp/config.json:/config.json \
  -v ~/tmp/output:/example_output_dir_name \
  speechmatics-docker-example.jfrog.io/transcriber-en:7.0.0 \
  --allformats /example_output_dir_name
```

SubRip Subtitles

SubRip (SRT) is a subtitling format that can be used in to generate subtitles for video content or other workflows. Our SRT output will generate a transcript together with corresponding alignment timestamps. We follow best

practice as recommended by major broadcasters in our default line length and number of lines output.

You can change the maximum number of lines supported, and the maximum character space within a line, by using configuration options as part of the output_config, which is part of the overall `config.json` object described below:

```
{
  "type": "transcription",
  "transcription_config": {
    ...
  },
  "output_config": {
    "srt_overrides": {
      "max_line_length": 37,
      "max_lines": 2
    }
  }
}
```

- `max_line_length` : sets maximum count of characters per subtitle line including white space (default: `37`).
- `max_lines` : sets maximum count of lines in a subtitle section (default: `2`).

How to transcribe files stored online

If you want to access a file stored in cloud storage, for example AWS S3 or Azure Blob Storage, you can use the `fetch_data` parameter within the `config.json` object. The `fetch_data` parameter specifies a cloud storage location.

You must ensure the URL you provide grants Speechmatics appropriate privileges to access the necessary files, otherwise this will result in a transcription error. Cloud providers like AWS and Azure allow temporary access to non-privileged parties to access and upload objects to cloud storage via generation of authenticated URLs by an authorised user. AWS recommends using pre-signed URLs to grant access when accessing objects from and uploading to S3. Azure recommends use of shared access signatures when accessing from and uploading to Azure Storage. Speechmatics supports both of these options

A pre-generated URL will containe authorization parameters within the URL. These can include information about how long the URL is valid for and what permissions access to the URL enables. More information is present on the page of each cloud provider

To successfully call data objects stored online using the Speechmatics container you must use the following parameters:

- `url` : (mandatory if you want to access an online file) the location of the file
- `auth_headers` : (optional) If your cloud storage solution requires authentication. The `auth_headers` parameter provides the headers necessary to access the resource. This is intended to support authentication or authorization when using http or https, for example by supplying an OAuth2 bearer token

An example is below:

```
{
  "type": "transcription",
  "transcription_config": {
    "language": "en"
  },
  "fetch_data": {
```

```
    "url": "https://example.s3.amazonaws.com/folder/file.mp3?
&AWSAccessKeyId=...&Expires=...&Signature=..."
  }
}
```

How to track a file

The jobInfo file

You can optionally submit additional information to the batch container that can then be used as further or tracking metadata. To do so you must submit a `jobInfo` file as a sepatate json object. This file is separate to the `config.json` object when submitting a request. The `jobInfo` file must include a unique id, the name and duration of the data file, and the UTC date the job was created. This information is then available in job results and in callbacks.

When using a `jobInfo` file you **must** submit the following mandatory properties:

- `created_at` - The UTC time the job was created at. An example is `"2019-01-17T17:50:54.113Z"`
- `data_name` - The name of the file submitted as part of the job. An example is `example.wav`. This does not need to match the actual file name
- `duration` - The length of the audio file. This must be an integer value in **seconds** and must be at least 0
- `id` - A customer-unique ID that is assigned to a job. This is not a value provided by Speechamtics

Optional Metadata

You may also submit the following **optional** properties as part of metadata tracking. These are properties that are unique to your organisation that you may wish to or are required to track through a company workflow or where you are processing large amounts of files. This information will then be available in the jobInfo output and in notification callbacks:

- `tracking` - Parent of the following child properties. If you are submitting metadata for tracking this **must** be included
  - `title` - The title of the job
  - `reference` - External system reference
  - `tag` - Any tags by which you associate files or data
  - `details` - Customer-defined JSON structure. These can include information valuable to you about the job

An example `jobInfo.json` file is below, with optional metadata inserted

```
{
  "created_at": "2020-06-26T12:12:24.625Z",
  "data_name": "example_file",
  "duration": 5,
  "id": "1",
  "tracking": {
    "title": "ACME Q12018 Statement",
    "reference": "/data/clients/ACME/statements/segs/2018Q1-seg8",
    "tags": [
      "quick-review",
      "segment"
    ],
    "details": {
      "client": "ACME Corp",
      "segment": 8,
```

```
      "seg_start": 963.201,
      "seg_end": 1091.481
    }
  }
}
```

Running the JobInfo file

Here is an example of processing a file on the batch container with an example jobInfo file:

```
docker run -v /PATH/TO/FILE/jobInfo.json:/jobInfo.json \
  -v /PATH/TO/FILE/config.json:/config.json \
  -v /PATH/TO/FILE/audio.wav:/input.audio \
  -e LICENSE_KEY=$license speechmatics-docker-prod-productsteam.jfrog.io/transcriber-
en:7.0.0
```

jobInfo Output Example

Here is an example of the json output when using a jobInfo file, with the first word of the transcript. You can see the output is divided into several sections:

- The license information, including the time of build and number of days remaining
- The information present in the jobInfo file, including any metadata or tracking information
- The configuration information presented in the config.json file
- The results of the transcript, including the word, confidence score, diarization information etc.

```
{
  "format": "2.4",
  "license": "productsteam build (Thu May 14 14:33:09 2020): 953 days remaining",
  "job": {
    "created_at": "2020-07-01T12:46:34.393Z",
    "data_name": "example.wav",
    "duration": 128,
    "id": "1",
    "tracking": {
      "details": {
        "client": "ACME Corp",
        "segment": 8,
        "seg_start": 963.201,
        "seg_end": 1091.481
      },
      "reference": "/data/clients/ACME/statements/segs/2018Q1-seg8",
      "tags": [
        "quick-review",
        "segment"
      ],
      "title": "ACME Q12018 Statement"
    }
  },
  "metadata": {
    "created_at": "2020-07-01T12:47:28.470Z",
    "type": "transcription",
    "transcription_config": {
      "language": "en",
      "diarization": "speaker"
    }
```

```
  },
  "results": [
    {
      "alternatives": [
        {
          "confidence": 1.0,
          "content": "This",
          "language": "en",
          "speaker": "M1"
        }
        ],
      "end_time": 1.98,
      "start_time": 1.86,
      "type": "word"
    }
  ]
}
```


NB When using the jobInfo file the format output will show 2 `created_at` parameters. The `created_at` under `job` is when the file was submitted for transcription The `createdDate` under `metadata` is when the output was produced. The time difference between the two provides the total transcription time, including any system delays as well as the actual time taken to process the job.

Full API Reference

Below are the full API references for the `config.json` and the `jobInfo.json` files.

config.json API Reference

The `config.json` is constructed of multiple configuration settings, each of which is responsible for a separate section of transcription output. All configuration settings are passed within the `type` object Only `transcription_config` is mandatory.

- type (**Mandatory**): Within type you must pass all other config information
- transcription_config: (**Mandatory**) Information about what language and features you want to use in the batch container
- fetch_data: (**Optional**) If you wish to transcribe a file stored online, you may pass this within the config.json file
- notification_config: (**Optional**) If you want to use callbacks, this documents where and how they are sent
- output_config: (**Optional**) If you want to retrieve files in SRT format, and you want to alter the default settings in how SRT appears only.

transcription_config

Name	Type	Description	Required
language	string	Language model to process the audio input, normally specified as an ISO language code	Yes
additional_vocab	[object]	List of custom words or phrases that should be recognized. Alternative pronunciations can be specified to aid recognition.	No
punctuation_overrides	[object]	Control punctuation settings. Only valid with languages that support advanced punctuation. These are English, French, German, Spanish, Dutch, Malay, and Turkish.	No

diarization	string	The default is `none`. You may specify options of `speaker`, `channel`,`speaker_change`, `channel_and_speaker_change`, or `none`	No
channel_diarization_labels	[string]	Transcript labels to use when using collating separate input channels. Only applicable when you have selected `channel` as a diarization option	No
output_locale	string	Only applicable with global English. Correct maps words to local spellings. Options are, `en-AU`, `en-GB`, or `en-US`	No

fetch_data

Name	Type	Description	Required
url	string	The online location of the file.	Yes
auth_headers	[string]	A list of additional headers to be added to the input fetch request when using http or https. This is intended to support authentication or authorization, for example by supplying an OAuth2 bearer token.	No

notification_config

Name	Type	Description	Required
url	string	The url to which a notification message will be sent upon completion of the job. If only one item is listed, it will be sent as the body of the request with `Content-Type` set to an appropriate value such as `application/octet-stream` or `application/json`. If multiple items are listed they will be sent as named file attachments using the multipart content type. If contents is not specified, the transcript item will be sent as a file attachment named `data_file`, for backwards compatibility. If the job was rejected or failed during processing, that will be indicated by the status, and any output items that are not available as a result will be omitted. The body formatting rules will still be followed as if all items were available. The user-agent header is set to `Speechmatics API V2` in all cases.	Yes
content	[string]	Specifies a list of items to be attached to the notification message. When multiple items are requested, they are included as named file attachments.	No
method	string	The method to be used with http and https urls. The default is POST.	No
auth_headers	[string]	A list of additional headers to be added to the input fetch request when using http or https. This is intended to support authentication or authorization, for example by supplying an OAuth2 bearer token.	No

output_config

Name	Type	Description	Required

srt_overrides	object	Parameters to override the defaults for SubRip (srt) subtitle format. `- max_line_length`: sets maximum count of characters per subtitle line including white space (default: 37). `-max_lines`: sets maximum number of lines per subtitle segment (default: 2).	No

jobInfo reference

Name	Type	Description	Required
createdAt	dateTime	The UTC date time the job was created.	Yes
data_name	string	Name of the data file submitted for job.	No
duration	integer	The file duration (in seconds).	No
tracking	object	Additional tracking information	No

tracking metadata within the jobInfo file

The following information can be passed within the tracking object as part of the jobInfo file

Name	Type	Description	Required
title	string	The title of the job.	No
reference	string	External system reference.	No
tags	[string]	A set of keywords	No
details	object	Customer-defined JSON structure.	No

For a full JobInfo example please see the example above in [How to track a file](###How to track a file.)

Ability to run a container with multiple cores

For customers who are looking to improve job turnaround time and who are able to assign sufficient resources, it is possible to pass a parallel transcription parameter to the container to take advantage of multiple CPUs. The parameter is called `parallel` and the following example shows how it can be used. In this case to use 4 cores to process the audio you would run the container like this:

```
docker run -i -rm -v ~/tmp/shipping-forecast.wav:/input.audio \
  -v ~/tmp/config.json:/config.json \
  speechmatics-docker-example.jfrog.io/transcriber-en:7.0.0 \
  --parallel=4
```

Depending on your hardware, you may need to experiment to find the optimum performance. We've noticed significant improvement in turnaround time for jobs by using this approach.

If you limit or are limited on the number of CPUs you can use (for example your platform places restrictions on the number of cores you can use, or you use the `--cpu` flag in your `docker run` command), then you should ensure that you do not set the `parallel` value to be more than the number of available cores. If you attempt to use a setting in excess of your free resources, then the container will only use the available cores.

If you simply increase the parallel setting to a large number you will see diminishing returns. Moreover, because files are split into 5 minute chunks for parallel processing, if your files are shorter than 5 minutes then you will see no parallelization (in general the longer your audio files the more speedup you will see by using parallel processing).

If you are running the container on a shared resource you may experience different results depending on what other processes are running at the same time.

The optimum number of cores is *N/5*, where *N* is the length of the audio in minutes. Values higher than this will deliver little to no value, as there will be more cores than chunks of work. A typical approach will be to increment the parallel setting to a point where performance plateaus, and leave it at that (all else being equal).

For large files and large numbers of cores, the time taken by the first and last stages of processing (which cannot be parallelized) will start to dominate, with diminishing returns.

Formatting Common Entities

Overview

Entities are commonly recognisable classes of information that appear in languages, for example numbers and dates. Formatting these entities is commonly referred to as Inverse Text Normalisation (ITN). Speechmatics will output entities in a predictable, consistent written form, reducing post-processing work required aiming to make the transcript more readable.

The language pack will use these formatted entities by default in the transcription for all outputs (JSON, text and srt). Additional metadata about these entities can be requested via the API including the spoken words without formatting and the entity class that was used to format it.

Supported Languages

Entities are supported in the following languages:

- Cantonese
- Chinese Mandarin (Simplified and Traditional)
- English
- French
- German
- Hindi
- Italian
- Japanese
- Portuguese
- Russian
- Spanish

Using the enable_entities parameter

Speechmatics now includes an `enable_entities` parameter. This can be requested via the API. By default this is `false`.

Changing `enable_entities` to `true` will enable a richer set of metadata in the JSON output only. Customers can choose between the default written form, spoken form, or a mixture, for their own workflows.

The changes are as following:

- A new `type` - `entity` in the JSON output in addition to `word` and `punctuation`. For example: "1.99" would have a `type` of `entity` and a corresponding `entity_class` of `decimal`
- The `entity` will contain the formatted text in the `content` section, like other words and punctuation
  - The `content` can include spaces, non-breaking spaces, and symbols (e.g. $/£/%)
- A new output element, `entity_class` has been introduced. This provides more detail about how the entity has been formatted. A full list of entity classes is provided below.

- The start and end time of the entity will span all the words that make up that entity
- The entity also contains two ways that the content will be output:
  - `spoken_form` - Each individual `word` within the entity, written out in words as it was spoken. Each individual word has its own start time, end time, and confidence score. For example: "one", "million", "dollars"
  - `written_form` - The same output as within `entity` content, with a type of `word` instead. If there are spaces in the content it will be split into individual words. For example: "$1", "million"

Configuration example

Please see an example configuration file that would request entities:

```
{
  "type": "transcription",
  "transcription_config": {
        "language": "en",
        "enable_entities": true
    }
}
```

Different entity classes

The following `entity_classes` can be returned. Entity classes indicate how the numerals are formatted. In some cases, the choice of class can be contextual and the class may not be what was expected (for example "2001" may be a "cardinal" instead of "date"). The number of `entity_classes` may grow or shrink in the future.

N.B. Please note existing behaviour for English where numbers from zero to 10 (excluding where they are output as a decimal/money/percentage) are output as **words** is unchanged.

Entity Class	Formatting Behaviour	Spoken Word Form Example	Written Form Example
alphanum	A series of three or more alphanumerics, where an alphanumeric is a digit less than 10, a character or symbol	triple seven five four	77754
cardinal	Any number greater than ten is converted to numbers. Numbers ten or below remain as words. Includes negative numbers	nineteen	19
credit card	A long series of spoken digits less than 10 are converted to numbers. Support for common credit cards	one one one one two two two two three three three four four four four	1111222233334444
date	Day, month and year, or a year on its own. Any words spoken in the date are maintained (including "the" and "of")	fifteenth of January twenty twenty two	15th of January 2022
decimal	A series of numbers divided by a separator	eighteen point one two	18.12
fraction	Small fractions are kept as words ("half"), complex fractions are	three sixteenths	3/16

	converted to numbers separated by "/"		
money	Currency words are converted to symbols before or after the number (depending on the language)	twenty dollars	$20
ordinal	Ordinals greater than 10 are output as numbers	forty second	42nd
percentage	Numbers with a per cent have the per cent converted to a % symbol	duecento percento	200%
span	A range expressed as "x to y" where x and y correspond to another entity class	one hundred to two hundred million pounds	100 to £200 million
time	Times are converted to numbers	eleven forty a m	11:40 a.m.
word	Entities that do not match a specific class	hundreds	hundreds

Output locale styling

Each language has a specific style applied to it for thousands, decimals and where the symbol is positioned for money or percentages.

For example

- English contains commas as separators for numbers above 9999 (example: "20,000"), the money symbol at the start (example: "$10") and full stops for decimals (example: "10.5")
- German contains full stops as separators for numbers above 9999 (example: "20.000"), the money symbol comes after with a non-breaking space (example: "10 $") and commas for decimals (example: "10,5")
- French contains non-breaking spaces as separators for numbers above 9999 (example: "20 000"), the money symbol comes after with a non-breaking space (example: "10 $") and commas for decimals (example: "10,5")

Example output

Here is an example of a transcript requested with `enable_entities` set to true:

- An `entity` that is "17th of January 2022", including spaces
  - The start and end times span the entire entity
  - An `entity_class` of `date`
  - The `spoken_form` is split into the following individual words: "seventeenth", "of", "January", "twenty", "twenty", "two". Each word has its own start and end time
  - the `written_form` split into the following individual words: "17th", "of", "January", "2022". Each word has its own start and end time

Note:

- By default and when speaker diarization is enabled, `speaker` parameter is added per word within the entity, spoken and written form
- When channel diarization is enabled, `channel` parameter is only added on the `results` parent within the entity and not included in spoken and written form

```
"results": [
 {
```

```
      "alternatives": [
        {
          "confidence": 0.99,
          "content": "17th of January 2022",
          "language": "en",
          "speaker": "UU"
        }
      ],
      "end_time": 3.14,
      "entity_class": "date",
      "spoken_form": [
        {
          "alternatives": [
            {
              "confidence": 1.0,
              "content": "seventeenth",
              "language": "en",
              "speaker": "UU"
            }
          ],
          "end_time": 1.41,
          "start_time": 0.72,
          "type": "word"
        },
        {
          "alternatives": [
            {
              "confidence": 1.0,
              "content": "of",
              "language": "en",
              "speaker": "UU"
            }
          ],
          "end_time": 1.53,
          "start_time": 1.41,
          "type": "word"
        },
        {
          "alternatives": [
            {
              "confidence": 1.0,
              "content": "January",
              "language": "en",
              "speaker": "UU"
            }
          ],
          "end_time": 2.04,
          "start_time": 1.53,
          "type": "word"
        },
        {
          "alternatives": [
            {
              "confidence": 1.0,
```

```
            "content": "twenty",
            "language": "en",
            "speaker": "UU"
          }
        ],
        "end_time": 2.46,
        "start_time": 2.04,
        "type": "word"
      },
      {
        "alternatives": [
          {
            "confidence": 1.0,
            "content": "twenty",
            "language": "en",
            "speaker": "UU"
          }
        ],
        "end_time": 2.79,
        "start_time": 2.46,
        "type": "word"
      },
      {
        "alternatives": [
          {
            "confidence": 0.97,
            "content": "two",
            "language": "en",
            "speaker": "UU"
          }
        ],
        "end_time": 3.14,
        "start_time": 2.79,
        "type": "word"
      }
    ],
    "start_time": 0.72,
    "type": "entity",
    "written_form": [
      {
        "alternatives": [
          {
            "confidence": 0.99,
            "content": "17th",
            "language": "en",
            "speaker": "UU"
          }
        ],
        "end_time": 1.33,
        "start_time": 0.72,
        "type": "word"
      },
      {
        "alternatives": [
```

```
            {
              "confidence": 0.99,
              "content": "of",
              "language": "en",
              "speaker": "UU"
            }
          ],
          "end_time": 1.93,
          "start_time": 1.33,
          "type": "word"
        },
        {
          "alternatives": [
            {
              "confidence": 0.99,
              "content": "January",
              "language": "en",
              "speaker": "UU"
            }
          ],
          "end_time": 2.54,
          "start_time": 1.93,
          "type": "word"
        },
        {
          "alternatives": [
            {
              "confidence": 0.99,
              "content": "2022",
              "language": "en",
              "speaker": "UU"
            }
          ],
          "end_time": 3.14,
          "start_time": 2.54,
          "type": "word"
        }
      ]
    }
  ]
```


If `enable_entities` is set to `false`, the output is as below:


```
  "results": [
    {
      "alternatives": [
        {
          "confidence": 0.99,
          "content": "17th",
          "language": "en",
          "speaker": "UU"
        }
      ],
      "end_time": 1.33,
```

```
      "start_time": 0.72,
      "type": "word"
    },
    {
      "alternatives": [
        {
          "confidence": 0.99,
          "content": "of",
          "language": "en",
          "speaker": "UU"
        }
      ],
      "end_time": 1.93,
      "start_time": 1.33,
      "type": "word"
    },
    {
      "alternatives": [
        {
          "confidence": 0.99,
          "content": "January",
          "language": "en",
          "speaker": "UU"
        }
      ],
      "end_time": 2.54,
      "start_time": 1.93,
      "type": "word"
    },
    {
      "alternatives": [
        {
          "confidence": 0.99,
          "content": "2022",
          "language": "en",
          "speaker": "UU"
        }
      ],
      "end_time": 3.14,
      "start_time": 2.54,
      "type": "word"
    }
  ]
}
```


Batch Container Migration Guide

Overview

This is a guide for customers who are updating to V8.0.0 or later (October 2020). It documents changes in the batch container, and how you, a customer, may need to reintegrate your batch container with any other systems. It is provided in addition to our standard release notes and documentation pack.

As part of this upgrade, some V1 features that are no longer supported have been completely deprecated, and will cease to work as announced in the v6.2.0 release.

In all cases, replacements are supported via our V2 input, and are documented in our Speech API Guide.

The changes below should show no loss of any feature or functionality as a result of the migration.

Scope

The scope of this document shows:

- What changes you, the customer, must make to use the Speechmatics batch container v8.0.0 if you have been using previous versions of the container
- If you are still using deprecated V1 features, this document will show which ones are no longer supported, and what you must use instead to ensure output
- Examples of our V2 output, and how it differs from our V1 output
- What changes have been made to licensing, and how you, the customer, must license a container prior to using it

The scope of this document excludes

- How to start the Batch Container - this is documented in our quick start guide
- Our Speech API - this is documented in the Speech API guide
- List of software packages used - this is covered in our release and attribution list
- Recommendations for any custom workflows or integrations you have built

What has changed

License File

Previously Speechmatics built batch containers with their own license file integrated within the container for each language required by a customer. For simplicity and replicability we have moved to a generic customer-agnostic container for each language, with each customer now receiving a separate license file to use with the container(s) they are licensed for.

Please note: The contents of the license file are confidential. They should be shared on the principles of least privilege. Speechmatics is not responsible for how you handle, store, or share licensing information.

Speechmatics Support will provide you with a new license file. The license is a JSON file called `license.json` and has the following JSON structure:

Item	Description
Customer name	This is your company's name
Id	This is internal to Speechmatics
Is-Trial	Whether the license is for a trial use of Speechmatics or not
Metadata	What Features a container is licensed to use. These can include: Speaker Diarization Channel Diarization Speaker Change Batch Container use

	Real-time container use Language: any supported language Language: A supported individual language (e.g. English)
NotValidAfter	The date after which the license expires and can no longer be used to run the container. The date is in ISO format
ValidFrom	The date from which this license is valid.
Signed Claims Token	A unique reference number used to validate the license file when running the container. Generated by Speechmatics

The values in this license file will reflect each customer's individual contract arrangement with Speechmatics.

An example license file is below:

```
{
   "contractid": 1,
   "creationdate": "2020-03-24 17:43:35",
   "customer": "Speechmatics",
   "id": "c18a4eb990b143agadeb384cbj7b04c3",
   "is_trial": true,
   "metadata": {
       "key_pair_id": 1,
       "request": {
           "customer": "Speechmatics",
           "features": [
               "MAPBA",
               "LANY"
           ],
           "isTrial": true,
           "notValidAfter": "2021-01-01",
           "validFrom": "2020-01-01"
       }
   },
   "signedclaimstoken": "example",
}
```

How this affects you

Previously the batch container was licensed by use of the environment variable `LICENSE_KEY`. This is no longer a valid variable and will not license the product. Instead you may either license the product via the two methods described below:

- **Volume mapping the license file into the container**. Volume map the location of the license file into the container when running transcription jobs, like the Configuration Object. Please see below for an example:

`docker run -i -v $AUDIO_FILE:/input.audio -v $CONFIG_JSON:/config.json -v /my_license.json:/license.json batch-asr-transcriber-en:8.0.0`

- **Use the value of the 'signed claims token'** from the license file and pass it as the value of the `LICENSE_TOKEN` variable when running a transcription job. See an example of using LICENSE_TOKEN below:

```
docker run -i -v $AUDIO_FILE:/input.audio -v $CONFIG_JSON:/config.json -e LICENSE_TOKEN='example' batch-asr-transcriber-en:8.0.0
```

If you lose a license file or it is no longer secure, Speechmatics can generate a new one. Please contact Speechmatics support if this is the case.

V1 Deprecation

In the Speechmatics container you can still process a media file for transcription without use of the V2 configuration object. This will generate our JSON v2 output without any alteration or changes to the text.

From the V8.0.0 release, the configuration file is now the only way by which you can modify the transcription output in the Speechmatics container. If you want to use features such as diarization, punctuation overrides, output locale etc. you must use the configuration object to request these features.

If you already do so, then you do not need to make any changes to how you use the container.

All JSON transcription output will now be in the V2.4 output.

As part of the v7.0.0 release support for V1 features was withdrawn. As part of this release all V1 features have now since been removed. Where applicable, these have been replaced by options within the configuration object. This includes the following:

V1 Item	Type	Replaced By
DIARIZE. Enables speaker diarization	environment variable	Use the diarization:speaker parameter within the configuration object
DIARISE. Enables speaker diarization	environment variable	Use the diarization:speaker parameter within the configuration object
CHANNEL_DIARISATION. enables channel diarization on stereo files	environment variable	Use the diarization:channel parameter within the configuration object
CHANNEL_DIARISATION_LABELS. Provides labels to different speakers when using channel diarization	environemnt variable	Replaced by the parameter channel_diarization_labels in the configuration object
LICENSE_KEY. used to license the batch container	environment variable	Replaced by LICENSE_TOKEN
`/extra_words.txt`. Used as a custom dictionary to generate additional vocabulary objects	text file	Use the additional vocab parameter within the configuration object to generate a custom dictionary
`/build_date`. Documents the date the batch container was built by	text file	Replaced by the new licensing file, and no longer needed
`/license_days`. How many days the license has to run	text file	Replaced by the new licensing file, and no longer needed

Changes to Notifications

Notifications are still supported in the batch container as before. There are a few changes in how single and multi-part notifications are generated and encoded, and this is noted below for integration purposes:

- If you request `transcript`, this will now be output in the JSON-V2 format rather than the deprecated V1 JSON format

- If you want to request an empty notification, you **must** specify `contents` to be blank by using `[]`. An example is provided below
- Notifications now have the `charset=utf8` on all transcript types. Ensure that your workflow can support this
- For receiving notifications, `Content-Type` header's used to be set always to `application/octet-stream`. This value now corresponds to actual content of the notification and is `application/json` in case of JSON-v2 content, `text/plain` in case of an SRT contetn, and `application/octet-stream` for TXT content

An example notification configuration that would generate a notification with no contents is shown below. This is a change from the previous version of batch container.

```
{
    "notification_config": [{
        "url": "http://localhost:8080",
        "contents": []
    }]
}
```