SPEECHMATICS

Real-time Virtual Appliance 3.3.0

Table of Contents

Real-time Virtual Appliance

High-Level Summary

This release contains improvements to telephony models for English and Spanish, and a number of bugfixes.

The following languages now support advanced punctuation: English (en), German (de), Spanish (es), French (fr), Dutch (nl) and Danish (da).

It is recommended that customers on previous releases upgrade to this version.

Important Notices

The legacy V1 API that the Real-time Virtual Appliance currently supports will be discontinued in a future release. We recommend that customers familiarise themselves with the configuration object as described in the Speech API Guide. Future notices will be provided to announce the end of life of the V1 API, and provide detailed instructions on migrations to the V2 API.

What's New

- Improvements to the telephony models for English and Spanish

Issues Fixed

The following issues are addressed since the previous release:

Issue ID	Summary	Resolution Description
REQ-7268	The internal 'reset' method can cause a server panic	This messages will now no longer be seen in the logs when services are restarted.
REQ-7461	Memory leak affecting licensing container	A small memory leak affecting the licensing service has now been addressed.
REQ-9618	Spurious warnings in rt worker log even though transcription is successful	This has now been fixed. Warnings in the log like this: `WARNING ([5.4.659~1450-2d99]:OutputArcForce():word-align-lattice.cc:635) Partial word detected at end of utterance` will no longer be seen.
REQ-11644	Offline activation cannot generate deactivation code	Previously if you submitted the PUT/POST request for offlineactivation, any attempt to deactivate with a DELETE request failed; this has now been fixed.
REQ-12172	RTA Speech API Guide link from /help is broken	The link has now been fixed; it is possible to access the Speech API guide from `https://${APPLIANCE_HOST}:8080/help/`

Known Limitations

The following are known issues in this release:

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-7549	Memory leak affecting gRPC	There is a small memory leak in the gRPC Python server (https://github.com/grpc/grpc/issues/5913).

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-11087	Additional white space appearing before the very last end of sentence punctuation character.	Its been observed that additional white space can occur before a full-stop '.' in some transcripts.
REQ-11135	3.2.0 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 transcrtips. We plan to support a hesitation filtering capability in a future release for customers that do not want to see hesitations on transcripts.
REQ-11792	Speaker change token positioning is incorrect	We are aware of a consistent mis-placing of the speaker change token after the first word of the new speakers' sentence rather than before it.
REQ-11829	No worker available in RTVA in unclean disconnects	It is possible to get stuck in a cycle of receiving `"job_error - No worker can be scheduled because the service is at capacity"` messages if the connection was not closed cleanly.
REQ-12202	High memory usage when using custom dictionary	It has been observed that when using custom dictionary an additional 800-1700MB of memory is required (depending on the size of the wordlist used).

Supported Platforms

Virtual Appliance image (OVA) for installation on:

- VMware ESXi 6.5+ or VMware Workstation Player.
- VirtualBox 5.2+
- Amazon EC2

See the Installation and Admin Guide for details on the minimum specifications for the VM. The maximum number of concurrent jobs (maxworkers) that you can run on a single appliance is 32.

Form Factors

There are four variants of the Real-time Virtual Appliance.

Variant	Image Size	Max. Disk Space	Languages
nano	11GB	40GB	en
mini	16GB	40GB	en, de, es
midi	27GB	60GB	en, de, es, fr, ko, ja, nl, pt
maxi	45GB	80GB	en, de, es, fr, ko, ja, nl, pt, it, da, pl, ca, hi, ru, sv

Upgrade Path

Remove the license from your old appliance (see the Admin Guide), then re-import the new OVA and configure networking as per the Installation and Admin guide. You will need to re-apply the license code you have once the OVA has imported.

Installation

Upload the OVA to VMWare ESX, VMWare Workstation Player, or VirtualBox. See the Installation and Admin Guide for more information.

Real-time Virtual Appliance Installation and Admin Guide

This guide explains how to install and administer the Real-time Virtual Appliance using the Management REST API.

The simplest method of accessing the Management API is with a web browser by connecting to the Real-time Virtual Appliance using the following URL:

```
https://${APPLIANCE_HOST}:8080/docs/
```

Where ${APPLIANCE_HOST} is the the IP address or DNS name of the virtual appliance.

There are two flavors or "modes" of Speechmatics virtual appliance: real-time and batch. For the most part installation and administration are identical for both modes. Where differences exist this is explicitly noted in this guide.

System requirements

The Speechmatics Real-time Virtual Appliance operates on a hypervisor host system. For this version of the appliance, the following hypervisors are supported:

- VMware®
- VirtualBox
- AWS EC2

For the virtual appliance to operate as required, the host must meet the requirements and have the resources available as defined below.

Host requirements

The virtual appliance can operate in any VMware supported environment that claims support for VMware virtual hardware specification 9 and above (see https://kb.vmware.com/s/article/1003746).

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.

See below for minimum Real-time Virtual Appliance VM (guest) specifications; the host machine must have enough resources (processor, memory and storage) to run the hypervisor, the guest VMs you intend to host on it, plus any other processes you expect to run on it. Vendor guidelines should be followed for other host requirements and installation process.

For VMWare, the document Performance Best Practices for VMware vSphere® 6.0 contains a comprehensive overview of hardware considerations and recommendations on how to optimize your host platform. See https://www.vmware.com/support.html for up-to-date technical information on VMWare.

For VirtualBox, please consult the online documentation: https://www.virtualbox.org/wiki/Documentation

For Amazon EC2, the following link explains how to setup a VM using an Amazon S3 to store the OVA file: https://docs.aws.amazon.com/vm-import/latest/userguide/vmimport-image-import.html.

Real-time Virtual Appliance requirements

Real-Time Virtual Appliance

The Speechmatics Real-Time Real-time Virtual Appliance must be allocated the following *minimum* specification:

- 1 vCPU
- 4GB RAM
- Up to 38GB hard disk space

For each concurrent input stream the appliance requires an additional 1 vCPU and up to 1.5GB RAM. If you are using the custom dictionary (additional words) feature then each concurrent input stream that is configured to use it will require up to 3GB RAM.

Batch Virtual Appliance

For operation in batch mode, the following *minimum* specifications are required:

- 2 vCPUs
- 8GB RAM
- Up to 44GB hard disk space

For more details on how to scale your Real-time Virtual Appliance and protect resources by configuring limits on the maximum number of concurrent jobs that can be processed, please refer to the relevant section in the Admin Guide.

Downloading the appliance

A download link will be provided by Speechmatics through the solutions section of the support portal (https://support.speechmatics.com). The latest version of the appliance can be located within the solutions section. Select the required version number within the "Real-time Virtual Appliance" area (for example {{ book.product.version }}) to view the download link and all associated documentation for the virtual appliance. Once the download link is selected the download will begin, or a save file prompt will appear, enabling the file to be saved (the exact method will depend on the web browser being used). After the download a file with an ".ova" extension will be stored on the computer.

An account is required to access the documents and download link in the support portal. If an account is not available or the "Real-time Virtual Appliance" section is not visible in the support portal, please contact Speechmatics Support support@speechmatics.com for help.

Importing the appliance

Once the **.ova** file has been downloaded, it is ready to be imported into the host you have already prepared. Please ensure that the host meets the requirements stated earlier in this guide, then based on the hypervisor environment follow the instructions below.

VMware ESXi

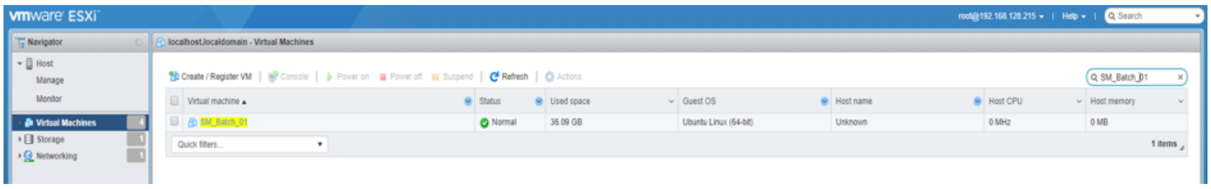
The following steps can be used to import the virtual appliance into VMWare ESXi 6.5:

- Open the vSphere web console on the host
- Choose "Virtual Machines" from the Navigator
- Select "Create/Register VM" option
- A wizard will appear:
    - Choose "Deploy a virtual machine from an OVF or OVA file" and click "Next"
    - Enter a VM name e.g. "SM_Batch_01", and drag the downloaded .ova file onto the window and click "Next"
    - Select a datastore that has enough capacity to store the virtual appliance and click "Next"

- From the "VM network" dropdown box, select a network
- Choose Thin or Thick disk provisioning (the Speechmatics Real-time Virtual Appliance supports either. Choose the options that is right for the hosting environment refer to VMWare documentation for help and click "Next"
- Check the details are correct and click "Finish"

- The virtual appliance will import. This can take a few minutes depending on the datastore chosen.

Once the VM has imported it should be visible on the vSphere web console:

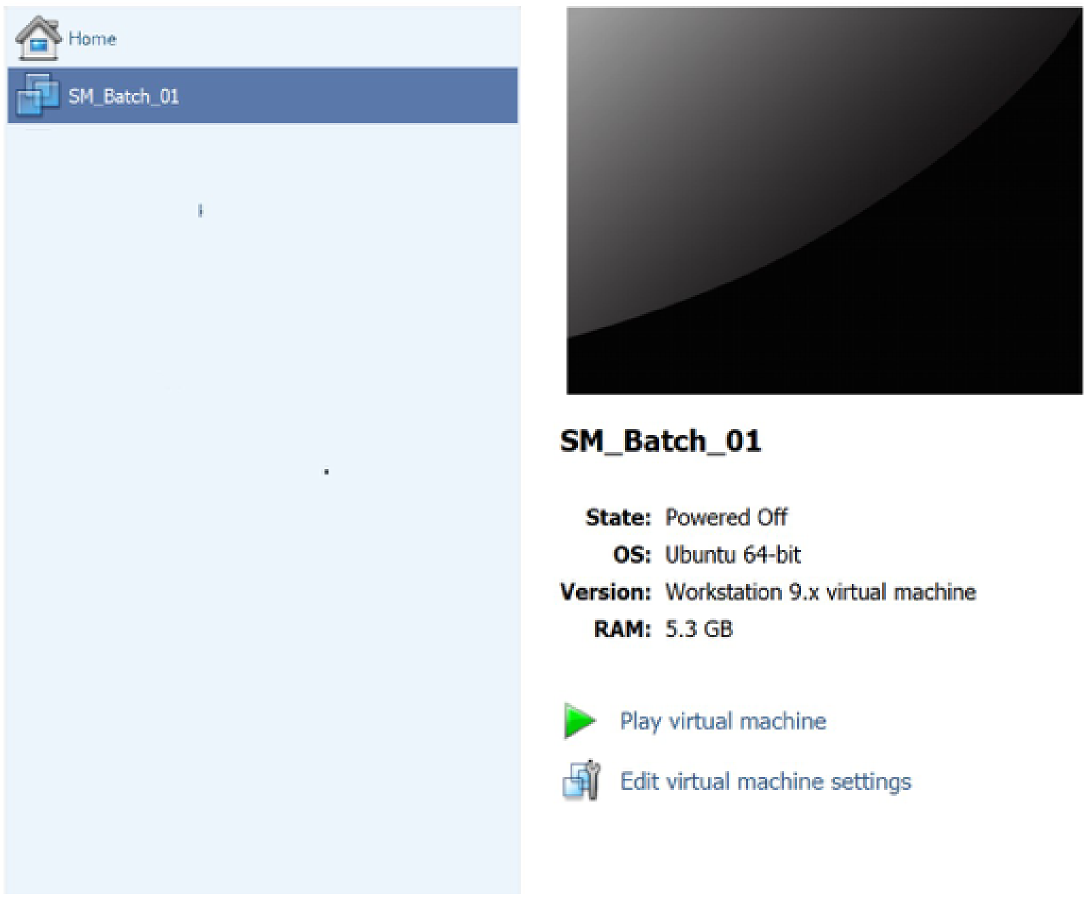


Virtual machine ▲	Status	Used space	Guest OS	Host name	Host CPU	Host memory
SM_Batch_01	Normal	36.09 GB	Ubuntu Linux (64-bit)	Unknown	0 MHz	0 MB

VMware Workstation Player

- Open VMware Workstation Player
- From the top options bar select "Player", then "File" and "Open..."
- The "Open Virtual Machine" window will appear. Navigate to the ".ova" file you downloaded earlier, select it, click "Open"
- Enter a VM name e.g. "SM_Batch_01"
- A default storage location for the virtual appliance will be shown, the can be changed if required. Click "Import".
- Dropdown box from the top options bar, click on "File"
- The virtual appliance will import. This can take a few minutes depending on the hard disk chosen

Once the VM has imported it should be visible on the Workstation player:



SM_Batch_01

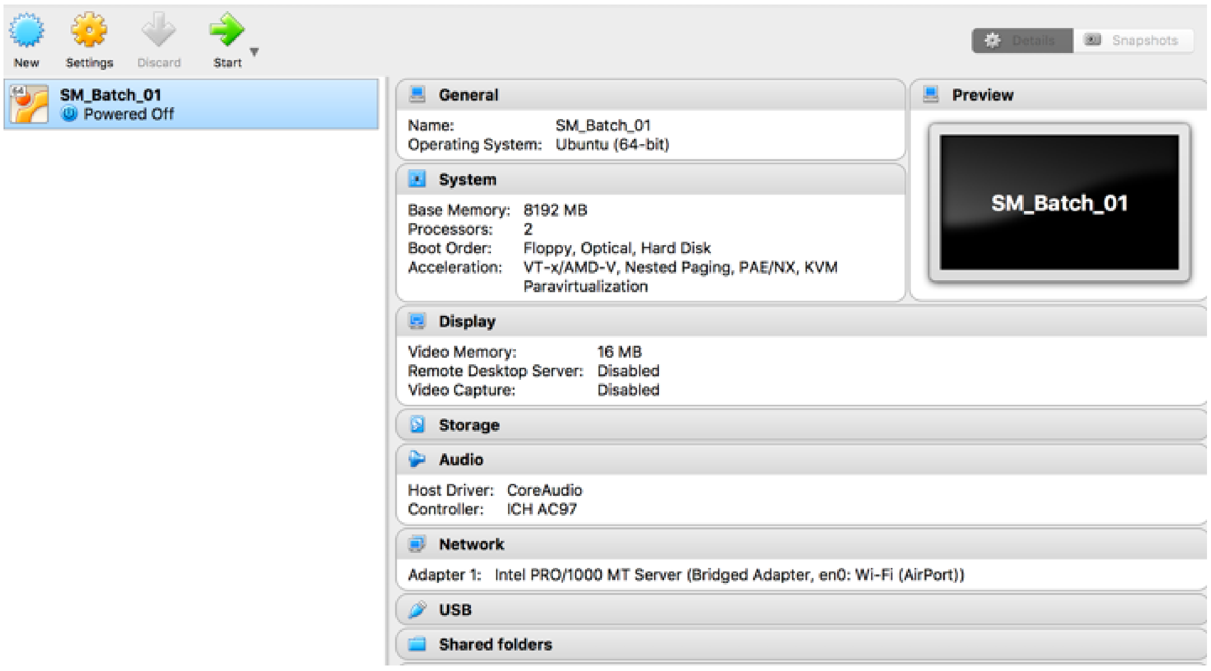
State: Powered Off
OS: Ubuntu 64-bit
Version: Workstation 9.x virtual machine
RAM: 5.3 GB

VirtualBox

The following steps can be used to import the virtual appliance into VirtualBox 5.2 or above.

- Open VirtualBox
- From the Manager window select "File", then "Import Appliance..."
- In the Name field, name the Real-time Virtual Appliance e.g. "SM_Batch_01"
- Browse to the OVA file and click on the "Import" button

Once the VM has imported it should be visible on the VirtualBox Manager:



General
Name: SM_Batch_01
Operating System: Ubuntu (64-bit)

System
Base Memory: 8192 MB
Processors: 2
Boot Order: Floppy, Optical, Hard Disk
Acceleration: VT-x/AMD-V, Nested Paging, PAE/NX, KVM Paravirtualization

Display
Video Memory: 16 MB
Remote Desktop Server: Disabled
Video Capture: Disabled

Storage

Audio
Host Driver: CoreAudio
Controller: ICH AC97

Network
Adapter 1: Intel PRO/1000 MT Server (Bridged Adapter, en0: Wi-Fi (AirPort))

USB

Shared folders

Amazon Web Services

This section explains how to create a Real-time Virtual Appliance EC2 instance on the Amazon Web Services (AWS) platform by using the AWS VM Import/Export tool. This tool is designed for importing VM images from the OVA file format provided by Speechmatics. You will import the image as an Amazon Machine Image (AMI), from which you can then launch machine instances.

The information in this section is taken from the official AWS documentation and parts of it have been extracted to focus more on the particulars of the Speechmatics Real-time Virtual Appliance. For more details of the Amazon VM image import process, please refer to https://docs.aws.amazon.com/vm-import/latest/userguide/vmimport-image-import.html

Prerequisites

There are a few pre-requisites that you will need to have setup before you can follow the instructions in this section:

- AWS Command Line Interface (CLI)
- Python 2.6.5 or higher

Please follow the recommendations on configuration of the AWS CLI by referring to the Getting Started guide.

Uploading the OVA file to S3

This section describes the process of uploading the Speechmatics OVA file to an Amazon S3 bucket from where it can be imported as an AMI instance. We recommend using a bucket in the same region where you want the AMI to be created and made available.

Once you've identified or created the S3 bucket on your account where the Speechmatics Real-time Virtual Appliance OVA will be uploaded to, you can use any of the tools below to help with the upload of the OVA file.

- The following AWS SDK libraries support S3 multipart upload (which is the recommended method given the large size of the OVA file):
  - AWS SDK for Java
  - AWS SDK for .NET
  - AWS SDK for PHP
  - AWS SDK for Python (Boto)
  - AWS SDK for Ruby
  - You can also use the Multipart Upload API directly
- User interface tools, for instance:
  - S3 Browser
  - CloudBerry S3 Explorer

For more information about the multipart uploads, see the AWS documentation.

Importing the OVA as AMI instance

After the Virtual Appliance OVA file has been successfully uploaded to an S3 bucket, it's time to import the image.

See the AWS documentation that covers uploading an image for full details.

The steps that you will perform in this section include (in orderr):

- Creating a Service Role on your AWS account
- Assigning a Role Policy to this Service Role
- Importing the OVA for the Real-time Virtual Appliance from the S3 bucket file

Creating an Import Service Role

First of all, a **service role** needs to be created on your AWS account. This allows certain operations, including downloading images from an S3 bucket.

Create a file named `trust-policy.json` with the following policy:

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

Then use the `create-role` command from the AWS CLI to create a role named `vmimport` . You need to specify the full path of the `trust-policy.json` file:

```
aws iam create-role --role-name vmimport --assume-role-policy-document file://trust-policy.json
```

You need to ensure the that `file://` prefix is prepended to the filename.

Creating a Role Policy

Create a file named `role-policy.json` with the following policy. Where you see `ova-bucket` it will need to be replaced with the name of the S3 bucket where the OVA file is stored.

```
{
   "Version":"2012-10-17",
   "Statement":[
      {
         "Effect":"Allow",
         "Action":[
            "s3:GetBucketLocation",
            "s3:GetObject",
            "s3:ListBucket"
         ],
         "Resource":[
            "arn:aws:s3:::ova-bucket",
            "arn:aws:s3:::ova-bucket/*"
         ]
      },
      {
         "Effect":"Allow",
         "Action":[
            "ec2:ModifySnapshotAttribute",
            "ec2:CopySnapshot",
            "ec2:RegisterImage",
            "ec2:Describe*"
         ],
         "Resource":"*"
      }
   ]
}
```

Use the `put-role-policy` command to attach the policy to the role. You must specify the full path to the location of the `role-policy.json`:

```
aws iam put-role-policy --role-name vmimport --policy-name vmimport --policy-document
file://role-policy.json
```

Importing the OVA

Importing the virtual appliance image (OVA) to Amazon EC2 as an Amazon Machine Image (AMI) is the next step.

Create a file named `containers.json` with the following content. Where you see `ova-bucket` it will need to be replaced with the name of the S3 bucket where the OVA file is stored. The below example has a file named `rtappliance-2.0.1-b21086.ova`.

```
[
  {
    "Description": "Real Time Virtual Appliance OVA",
    "Format": "ova",
    "UserBucket": {
        "S3Bucket": "ova-bucket",
        "S3Key": "rtappliance-1.1.0-b21086.ova"
    }
}]
```

Use the `import-image` command to create an import task (Specify the full path to the location of the `containers.json`):

```
aws ec2 import-image --description "Real Time Virtual Appliance OVA" --disk-containers
file://containers.json
```

The resulting JSON output will show an `ImportTaskId` which you can use to check the status of the import task. You do this by running the `describe-import-image-tasks` command:

```
aws ec2 describe-import-image-tasks --import-task-ids import-ami-abcd1234
```

You need to replace the task identifier with the `ImportTaskId` for your import task (`import-ami-abcd1234` in this example).

When the status is in the `completed` state the AMI is ready to use.

Security

The following ports should be opened to be able to submit jobs and manage and monitor the Speechmatics Virtual Appliance. For more background on creating security groups refer to the official AWS documentation

Real-time Virtual Appliance

Port	Description
8080/TCP	Used for the Management API to manage the virtual appliance
9000/TCP	WebSockets Speech API for submitting jobs
3000/TCP	Monitoring (Glances)

Batch Virtual Appliance

Port	Description
8080/TCP	Used for the Management API to manage the virtual appliance
8082/TCP	REST Speech API for submitting jobs
3000/TCP	Monitoring (Glances)

Launching a Virtual Appliance

Now that the Virtual Appliance has been imported, it will be available as an AMI which can be launched as an instance. To launch a Speechmatics Virtual Appliance, do the following:

- Login to the AWS console and find your image under **EC2 Service | Images**
- Right-click the image and choose **Launch**
- Refer to the **System requirements** section of the Speechmatics Quick Start Guide or Admin Guide to identify how much system resources is required for your set up. Choose the instance type that meets your requirement
- Choose **Review and Launch** from the console. Setup the Key Pair if required and choose **Launch** again.

Full instructions for launching instances can be found here:
https://docs.aws.amazon.com/AWSEC2/latest/WindowsGuide/launching-instance.html

Network Configuration

Before starting the virtual appliance for the first time, it is important to consider the network settings that will be used. The section below describes the options.

Network interface mapping

Whilst the virtual appliance is powered off, the virtual network adaptor should be mapped to the correct physical adaptor on the host. The virtual interface must be mapped to a physical adaptor on which the Speechmatics Real-time Virtual

Appliance will be contacted. Steps are provided below for the supported hypervisors.

VMware ESXi

There is nothing to configure here. The network as specified during the import stage described above will be used.

VMware Workstation Player

Speechmatics recommends using bridged network mode. To ensure bridged networking is selected:

- Open VMware Workstation Player
- Right click on the virtual appliance e.g. "SM_App_01", and select "Settings..."
- Select the "Network Adapter" in the devices list
  - Select "Bridged: Connected directly to the physical network"
- Click "OK"

This will result in the VM using an IP Address for its use that is independent from that of the host.

VirtualBox

Speechmatics recommends using bridged network mode. To ensure bridged networking is selected:

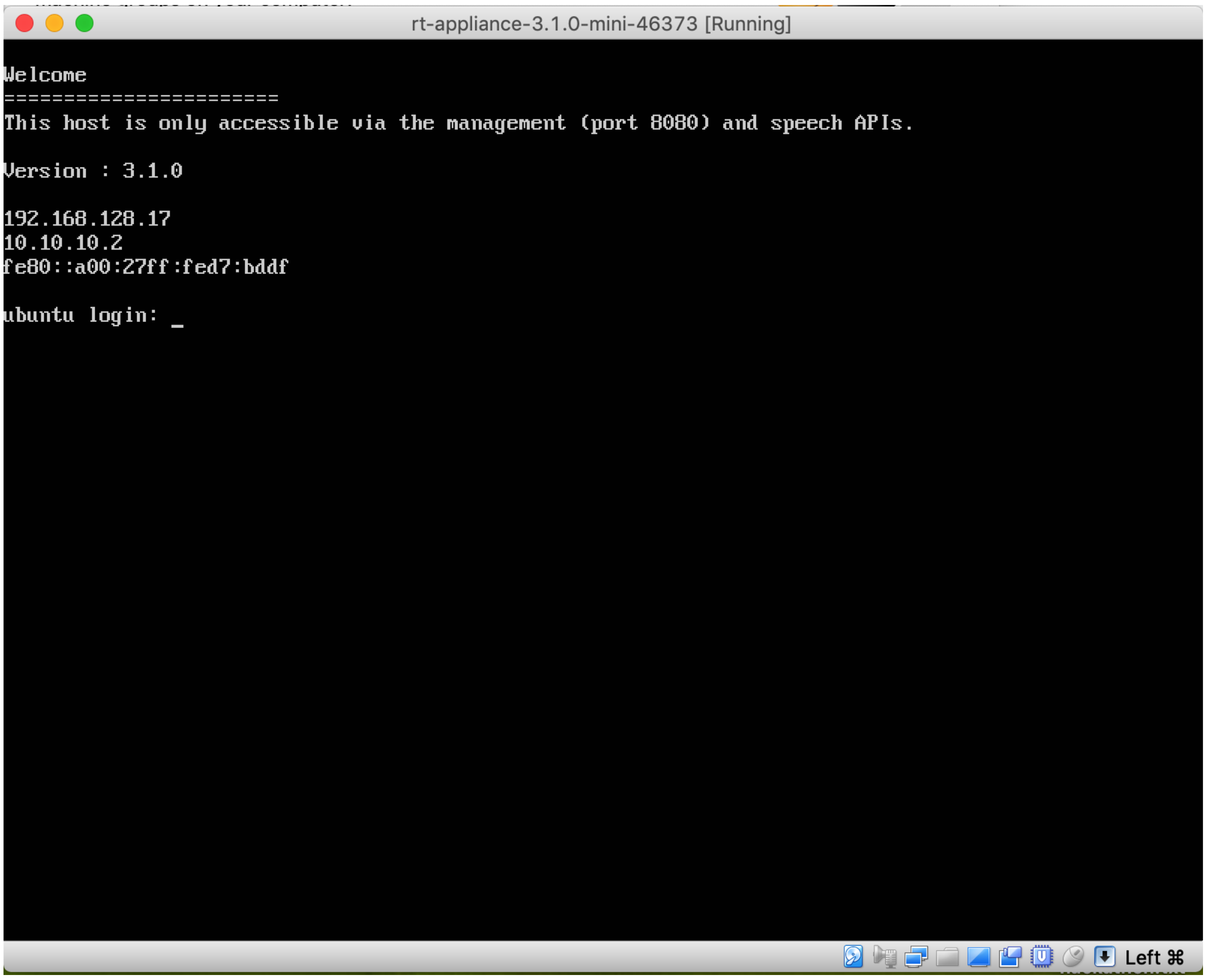
- Open VirtualBox
- Right click on the virtual appliance and select "Settings..."
- Select "Network" and from the "Attached to:" dropdown box, select "Bridged Adaptor"
- Click "OK"

This will result in the VM using an IP Address for its use that is independent from that of the host.

IP Configuration

When the Speechmatics Real-time Virtual Appliance is started, the default behavior will be to dynamically acquire an IP address. If there is no DHCP service available on the network, it will fall back to an IP address automatically assigned.

The IP address information can be viewed by opening the virtual appliance console once it has booted, as shown below.



```
Welcome
=======================
This host is only accessible via the management (port 8080) and speech APIs.

Version : 3.1.0

192.168.128.17
10.10.10.2
fe80::a00:27ff:fed7:bddf

ubuntu login: _
```

The screen shot above shows the 10.10.10.2 IP address as the fallback address. The other address shown was allocated by DHCP and should be used for all communication.

If DHCP cannot be used, a static IP address can be configured as described below.

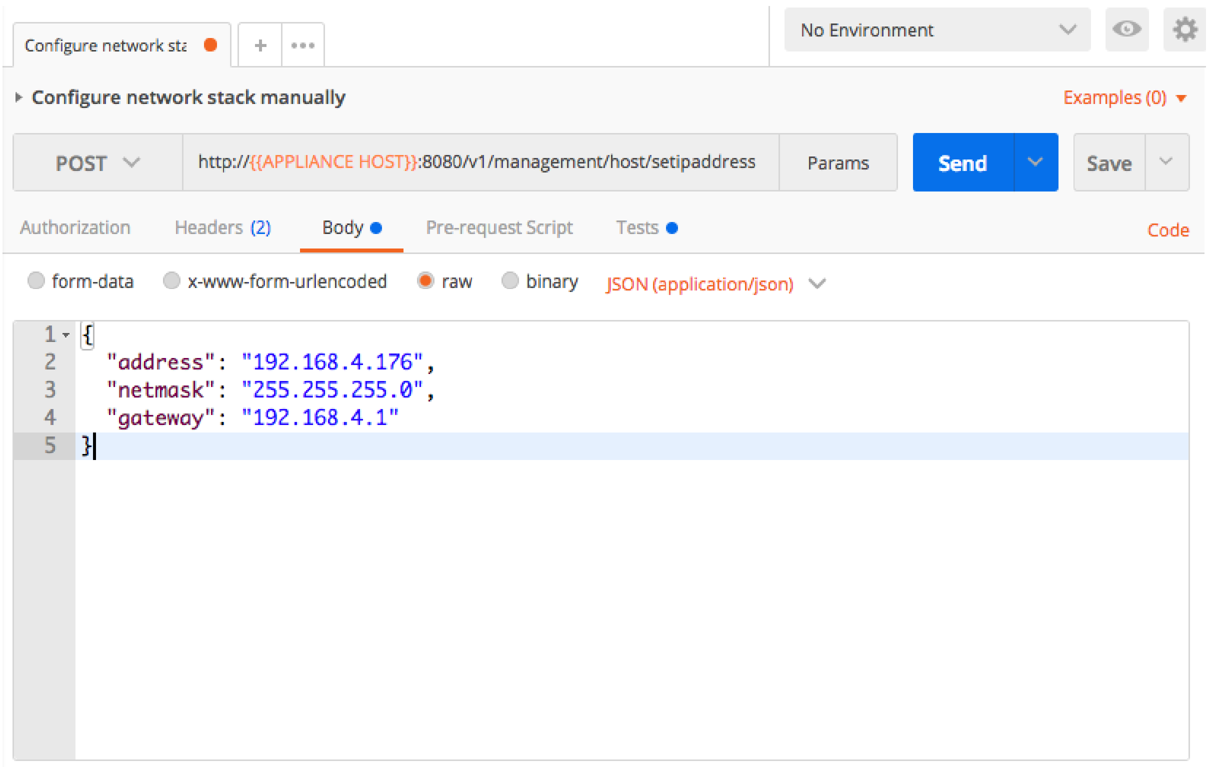
Configure static IP

To configure a static IP address, the Management REST API for the virtual appliance is used. The following information is required:

- **Method:** POST
- **URL:** http://*${APPLIANCE_HOST}*:8080/v1/management/host/setipaddress
- **Body Format:** JSON
- **Body:** address, netmask, gateway, nameservers

Where *${APPLIANCE_HOST}* is the hostname or IP address of your Real-time Virtual Appliance.

The example below shows use of Postman (available for free from the Chrome web store) to POST new IP settings.



```
http://{{APPLIANCE HOST}}:8080/v1/management/host/setipaddress
```

```
{
  "address": "192.168.4.176",
  "netmask": "255.255.255.0",
  "gateway": "192.168.4.1"
}
```

You can optionally specify a list of nameservers to use (if none are specified then, 8.8.8.8 is used), for example this time using curl from the command-line to make the POST request:

```
curl -L -X POST 'http://${APPLIANCE_HOST}:8080/v1/management/host/setipaddress' \
   -H 'Accept: application/json' \
   -H 'Content-Type: application/json' \
   -d '@network-config.json'
```

In this example, a local file network-config.json is used for the JSON configuration:

```
{
   "address": "192.168.128.96",
   "netmask": "255.255.255.0",
   "gateway": "192.168.4.1",
   "nameservers": ["208.67.222.222", "208.67.220.220"]
}
```

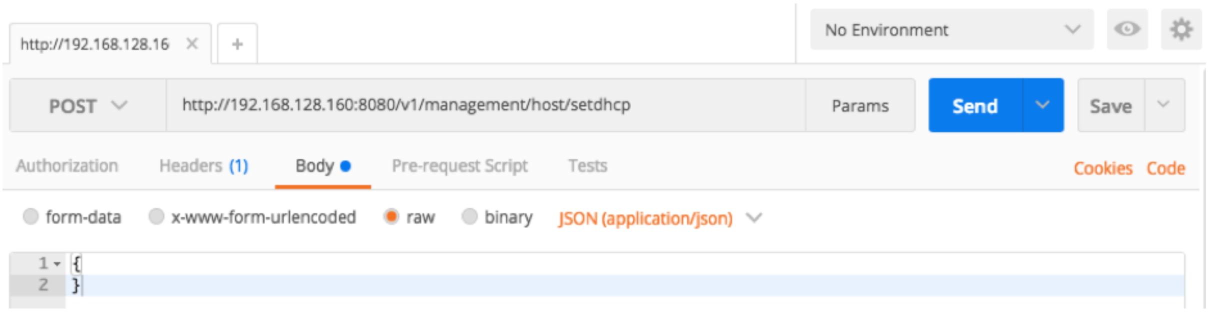
NOTE: once the POST is sent, the virtual appliance will automatically reboot. Check the console to verify the new IP address has been applied.

Configure DHCP IP

To configure a dynamic IP address using DHCP, the admin REST API is used as follows:

- **Method:** POST
- **URL:** http://*${APPLIANCE_HOST}*:8080/v1/management/host/setdhcp
- **Body format:** JSON

The example below shows how to use Postman to POST to the REST API in order to configure a DHCP address.



http://192.168.128.160:8080/v1/management/host/setdhcp

```
{
}
```

```
curl -L -X POST 'http://${APPLIANCE_HOST}:8080/v1/management/host/setdhcp' \
  -H 'Accept: application/json'
```

NOTE: once submitted, the virtual appliance will automatically reboot. Check the console to verify the new IP address has taken affect.

Licensing

The Speechmatics Real-time Virtual Appliance uses a cloud-based licensing mechanism, meaning that the under normal circumstances the appliance must be connected to the Internet, at least when the license is activated.

Note: For deployments where this is not possible, and where an offline license has been provided, it is possible to license the appliance without an Internet connection. Consult the Admin Guide for details on how to apply an offline license.

Your appliance must have been activated with a valid license before the Speech API can be used. Use of the Management API does not require a license. Please contact Speechmatics support support@speechmatics.com if you do not have a license.

Applying License

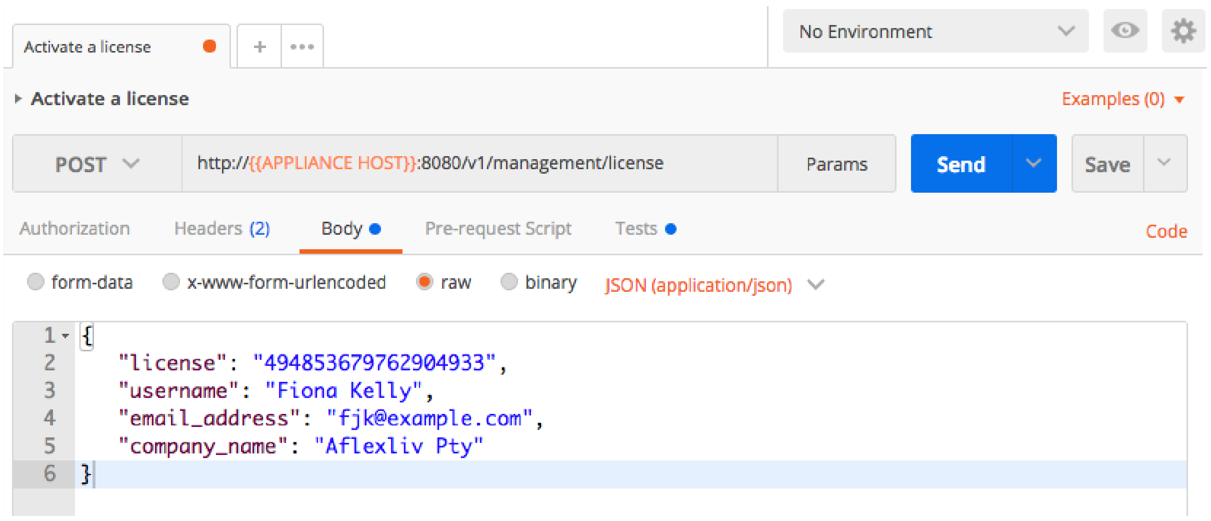
To apply the license that you have received from Speechmatics you use the Management API. The following information will be required:

- **Method:** POST
- **URL:** http://*${APPLIANCE_HOST}*:8080/v1/management/license
- **Body format:** JSON
- **Body:** license, username, email_address, company_name

Where *${LICENSE_CODE}* is the license code you've been provided with. The other fields (username, email_address and company_name) are optional, but we recommend that you fill them in with your details to help with support.

Note: Make sure when applying the license, that all the appliance services are running and that you have a route to the Internet; otherwise the activation will fail.

The example below shows how to use Postman to POST to the REST API to apply (activate) the license.



http://{{APPLIANCE HOST}}:8080/v1/management/license

```
{
    "license": "494853679762904933",
    "username": "Fiona Kelly",
    "email_address": "fjk@example.com",
    "company_name": "Aflexliv Pty"
}
```

Or, the same request from the command-line:

```
curl -L -X POST 'http://${APPLIANCE_HOST}:8080/v1/management/license' \
   -H 'Accept: application/json' \
   -H 'Content-Type: application/json' \
   -d '{
       "license": "494853679762904933",
       "username": "Fiona Kelly",
       "email_address": "fjk@example.com",
       "company_name": "Aflexliv Pty"
    }' \
   | jq
```

The response should indicate that the licensed status is true. The licensing activation requires a connection to the Internet (using port 80). If you are behind a corporate firewall that does not allow a direct connection to the Internet then you can configure the appliance to use a network proxy or relay server to allow you to license the appliance. Full instructions on how to set this up are to be found later on in this guide.

Verify and Go (Real-Time)

This section explains how to verify the correct operation of the Real-Time Virtual Appliance using the Websockets Speech API.

The first time that the Real-Time Virtual Appliance is started up there are no **persistent workers** configured. This means that workers will be spun up dynamically to the limit of the maxworkers limit. If you want to pre-allocate workers so that they are ready for incoming streams, you can select a language to configure as a persistent worker. You can find out the list of available languages on your appliance using the REST API:

- **Method:** GET
- **URL:** http://*${APPLIANCE_HOST}*:8080/v1/management/persistentworkers

This will return as JSON output the list of persistent workers (by language code) and the number of instances; initially they will all be zero. The list of supported languages are available on the Speechmatics website https://www.speechmatics.com/language-support/, or you can consult the release notes.

Use the Management API to set `persistent_workers` with a count of at least 1 and the language code to use. For example, to set French as a persistent worker, use the following method:

- **Method:** POST
- **URL:**

  http://*${APPLIANCE_HOST}*:8080/v1/management/persistentworkers
- **Body Format:** JSON
- **Body:** `{ "persistent_workers": [ { "count": "1", "id": "fr" } ] }`

This will return an updated list of persistent workers with entry for French (fr) updated to 1.

```
curl -s -L -X POST 'http://${APPLIANCE_HOST}:8080/v1/management/persistentworkers' \
   -H 'Accept: application/json' \
   -d '{ "persistent_workers":
      [ { "count": "1", "id": "fr" } ]
   }'
```

Verify the service

Check that all the Speechmatics services within the appliance are up and running before trying to open a WebSockets connection. The Management API can be used for this.

```
curl -s -L -X GET 'http://${APPLIANCE_HOST}:8080/v1/management/services' \
   -H 'Accept: application/json'
```

Go!

The Speechmatics Real-time Virtual Appliance is now ready to use.

The Speech API Guide provides details of how to use WebSockets to stream audio to the Speechmatics engine in real time and obtain transcripts. For all WebSocket communication, ensure that the IP address identified in the steps above is used.

SSL Configuration

When the appliance is imported it contains a default self-signed certificate, so you can use HTTPS to access the appliance via the Management, Monitoring and Speech APIs. However, we recommend replacing this default SSL certificate with your own certificate, signed by your organisation or a trusted third-party certificate authority (CA).

Default behaviour

By default, our appliances allow connections over HTTP. The services on the appliance expose several ports for HTTP access, such as 8080 for the management API and 3000 for the monitoring API.

Since version 3.4.0 of the appliances, we also support HTTPS access to these services over port 443. To use HTTPS simply change the protocol used for API calls from `'http'` to `'https'`, and remove the port from the URL. If you are copying the examples from this document you can set the `$APPLIANCE_HOST` environment variable like this: `export APPLIANCE_HOST=localhost`.

Management API Examples

```
curl -L -X GET "http://${APPLIANCE_HOST}:8080/v1/management/services" \
   -H 'Accept: application/json'
```

To modify this to use a secure API call, change `http://` to `https://` and remove the port number `:8080` from the URL:

```
curl -L -X GET "https://${APPLIANCE_HOST}/v1/management/services" \
    -H 'Accept: application/json'
```

Note: If you are using a self-signed certificate (your own, or the Speechmatics certificate that is used by default), then you will see a warning like this when using the above curl command:

```
curl: (60) SSL certificate problem: self signed certificate
```

Warning: The default SSL certificate on the appliance is a self-signed certificate created by Speechmatics, which is not signed by any certificate authority. Your HTTP client or web browser may warn that this is insecure. This warning can be suppressed, for example with cURL by adding the `--insecure` flag, however customers who are serious about security should not be using the self-signed certificate. We recommend uploading your own SSL certificate to the appliance. Instructions for doing this can be found below.

Important: We have added `--insecure` to some of them cURL examples in this document so that the command trusts the self signed certificate. You won't need this option once you've uploaded your own certificate and configured your own system to trust it.

Monitoring API Example

With access to the Monitoring API (available on port 3000 if you are using HTTP) you will need to prefix the endpoint with `/monitor`. For example:

```
curl --insecure -L -X GET "https://${APPLIANCE_HOST}/monitor/api/3/mem"
```

Speech API Example

Access to the REST Speech API (available on port 8082 using HTTP), is also possible via HTTPS:

```
curl --insecure -L -X GET "https://${APPLIANCE_HOST}/v1.0/user/1/jobs/"
```

Using your own SSL certificate and private key

To use your own SSL certificate you'll need to upload your *certificate* file as well as the associated *private key* file.

- The **private key** file normally has a '.key' extension and should look similar to the example below.

```
-----BEGIN RSA PRIVATE KEY-----
xqgLwi4gJ9+9Qkavpk3WpPFTTYUfVrCJNviKEn5wA1tuutqLQkRTcxJtrEk8trKI
fCxeZo35yVhYmDGUIuAdAcPRTPj0XZkXQRhkITmD8TYMc/sVlJpFr+TAssGzute8
... 21 lines redacted ...
+bLv4aqI9tZrwpyeziaOuyQRhYodpAjhCyCFMkJjY59BKv/cqMHx8FPDQmaZ9Xs0
SmE9JAknDgF5yLHm1Q6WZ1/L/M4SkgIqEglF7ifLd5M3wskpmHia6/f8Fa2KwbBJ
-----END RSA PRIVATE KEY-----
```

Note: We do not currently support encrypted/password protected private key files.

- The **certificate** file should be PEM encoded and normally has a '.crt' or '.pem' extension. It should look similar to this:

```
-----BEGIN CERTIFICATE-----
MIIGuzCCBaOgAwIBAgIIHlfyznYUA8wDQYJKoZIhvcNAQELBQAwgbQxCzAJBgNV
BAYTAlVTMRAwDgYDVQQIEwdBcml6b25hMRMwEQYDVQQHEwpTY290dHNkYWxlMRow
... 32 lines redacted ...
P4LMbjCA4mqQvlipibeSAN1E4OrFL47zLcy+H9M0+Rw2CUiwL8QZFq+TAiIZ34tC
UVCh52xpB9/BhO++QbGd1zObqDhcGEg8pJpJIycej9t4GN1eqNSudn0ibsQWev8=
-----END CERTIFICATE-----
```

Both files should be in PKCS8 format. If you have to upload a certificate chain, then the file you upload should contain the individual certificates concatenated, with your organisation's certificate first.

Uploading the certificate and key to the appliance

To upload your own certificate to the appliance you will need to make a POST request to the `/v1/security/sslcertificate` endpoint. This can be done using an HTTP client on the command line or with the management interface in a browser.

With the example shown here set `APPLIANCE_HOST` as appropriate (e.g. `export APPLIANCE_HOST=localhost` if your appliance is running locally):

```
curl --insecure -X POST "https://${APPLIANCE_HOST}/v1/security/sslcertificate" \
    -F "keyfile=@appliance.key" -F "certfile=@appliance.crt"
```

Warning: Do not upload these files over HTTP, or you risk leaking the private key for your certificate.

If the upload succeeds then you should receive an HTTP 200 response with a success message:

```
{
   "success": true,
   "message": "certificate and private_key applied successfully"
}
```

Be aware that setting a new certificate will cause the web server in the appliance to restart which can take around five seconds. During this period, requests will still be served, however the old certificate will be used. Existing connections such as job uploads or WebSocket streams will not be interrupted.

You can check the certificate on the appliance by using the `openssl` tool:

```
$ openssl s_client -connect ${APPLIANCE_HOST}:443
```

Disabling HTTP access

If desired, HTTP access may be disabled, which will cause any requests to the appliance using HTTP to fail. To do this, make a POST request to the `/v1/security/insecureports` endpoint, with a JSON body containing `{"enable_insecure_ports": false}`:

```
curl -X POST "https://${APPLIANCE_HOST}/v1/security/insecureports" \
    -H "Content-Type: application/json" \
    -d "{ \"enable_insecure_ports\": false}"
```

If the request succeeded then you should receive an HTTP 200 response. The web server in the appliance will take around five seconds to restart. Now, when attempting to make an HTTP request to the appliance you should see that no response is returned:

```
curl -X GET "http://${APPLIANCE_HOST}:8080/v1/management/services"

curl: (52) Empty reply from server
```

Enable Basic Authentication for Admin

An admin password can be set to enable HTTP basic authentication for an `admin` user. Note that **authentication is only enforced when using HTTPS**. If you set an admin password then you **must** also disable HTTP access as described in the previous section. If you do not do this then it will be possible for someone else to override the admin password by making an unauthorized HTTP request.

To set a password, make a POST request to the `/v1/security/adminpassword` endpoint. The username for basic auth is always `admin`.

```
curl -X POST "https://${APPLIANCE_HOST}/v1/security/adminpassword" \
  -H "Content-Type: application/json" \
  -d "{ \"password\": \"example\" }"

{"success":true,"message":"nginx_restart"}
```

If this request was successful then you should receive an HTTP 200 response with a success message. The web server in the appliance will take around five seconds to restart. All requests to HTTPS endpoints will now require a valid `Authorization` header as specified by RFC7617. Unauthenticated requests will fail, for example:

```
$ curl -X GET "https://${APPLIANCE_HOST}/v1/management/services"
<html>
<head><title>401 Authorization Required</title></head>
<body>
<center><h1>401 Authorization Required</h1></center>
<hr><center>nginx/1.17.6</center>
</body>
</html>
```

Authenticated requests should succeed. If you are using curl then the `--user` flag can be used to set the username and password (separated with a colon). If you're using the Management UI in a browser than a prompt will appear for a username and password.

```
$ curl --insecure -X GET --user "admin:example"
"https://${APPLIANCE_HOST}/v1/management/services"
```

If you have disabled HTTP access then it should now be impossible to make requests to the appliance without knowing the admin password. Please be aware that plain HTTP access does **not** require the admin password, and should be disabled if you are using a password.

FAQs

How do I reset the SSL settings?

If you have made a mistake in your SSL configuration, it is possible to reset the appliance to it's default settings. This will return it to using the self-signed certificate from Speechmatics, and will delete any configured admin password. If you have disabled HTTP access then you need to know the existing admin password in order to do this.

To do this, make a DELETE request to the `/v1/security/reset` endpoint:

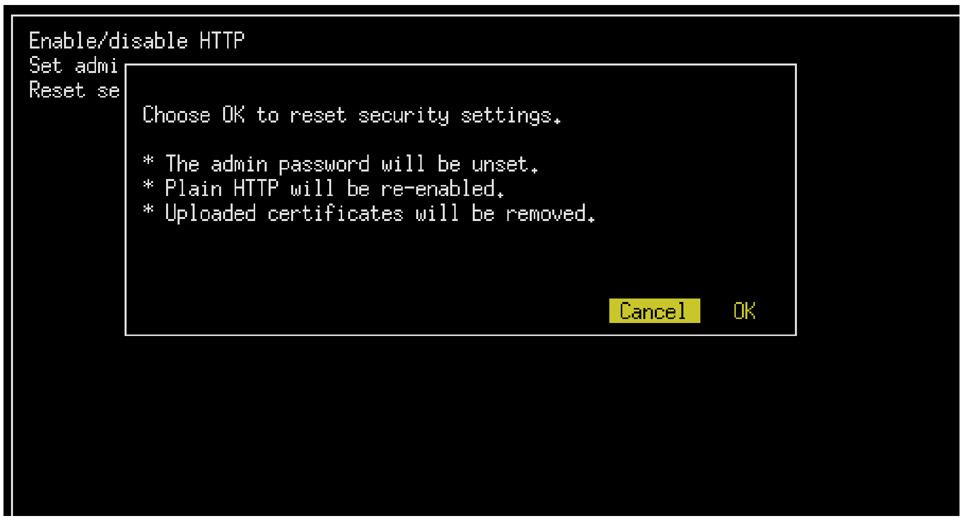
```
$ curl -X DELETE --user "admin:$PWD" "https://${APPLIANCE_HOST}/v1/security/reset"
{"success": true, "message": "nginx_restart"}
```

What if I forget the admin password?

If you have forgotten the admin password you have set, and have disabled HTTP access to the appliance then it will not be possible to interact with the appliance over HTTP/HTTPS. Fortunately there is a way to reset the SSL configuration if you have direct access to the appliance's console (through the hypervisor that you use).

See the 'Administration -> Services -> Console for Advanced Troubleshooting' section for instructions on how to access the console.

Once you have opened the console open the 'Security' menu and select the 'Reset security' option to reset all security settings. It is also possible to toggle HTTP access and set the admin password using this interface.



```
Enable/disable HTTP
Set admi
Reset se

Choose OK to reset security settings.

* The admin password will be unset.
* Plain HTTP will be re-enabled.
* Uploaded certificates will be removed.
```

What versions of SSL/TLS do you support?

We support TLS 1.2 and TLS 1.3. We do not support earlier versions of TLS/SSL as these are considered weak. In general we would recommend you keep your client frameworks up to date with the latest security patches and try to use the strictest TLS configuration that you can.

What cipher suites do you support?

For TLS 1.3 we support the following cipher suites that are considered strong (in server-preferred order):

- TLS_AES_256_GCM_SHA384
- TLS_CHACHA20_POLY1305_SHA256
- TLS_AES_128_GCM_SHA256

For TLS 1.2 we support the following cipher suites that are considered strong (in server-preferred order):

- TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384
- TLS_ECDHE_RSA_WITH_CHACHA20_POLY1305_SHA256
- TLS_ECDHE_RSA_WITH_ARIA_256_GCM_SHA384
- TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256
- TLS_ECDHE_RSA_WITH_ARIA_128_GCM_SHA256

Other cipher suites are available for TLS 1.2, but they are considered to be weak. Our recommendation is that you select one of the above cipher suites.

Networking

Network Requirements

When the virtual appliance is started for the first time it will automatically try to acquire an IP address using DHCP. If it is able to successfully acquire an address, it will be displayed on the VM console along with the fallback IP address: 10.10.10.2. However, if there is no DHCP server available on the network only the 10.10.10.2 IP address will be displayed.

The 10.10.10.2 address is a fallback address enabling communication with the virtual appliance when no DHCP services are available. This address should be used temporarily to set a static IP address if no DHCP is available. To do this, ensure that the client connecting to this address is on the same network by assigning it a suitable IP address (e.g. 10.10.10.3/24).

Note: The appliance uses an internal network of **10.254.0.0/22**. You need to ensure that any network you use does not have an IP address conflict with anything in this range: 10.254.0.0 - 10.254.3.255.

Configure Static IP

The virtual appliance can be configured to work on any IP network.

Setting a static IP requires three parameters: the IP address, subnet mask and default gateway. You set the static IP address like this:

```
curl -L -X POST 'http://${APPLIANCE_HOST}:8080/v1/management/host/setipaddress' \
   -H 'Accept: application/json' \
   -H 'Content-Type: application/json' \
   -d '{
       "address": "192.168.128.160",
       "netmask": "255.255.255.0",
       "gateway": "192.168.128.1"
    }' \
   | jq
```

Note: Once the POST is sent, the virtual appliance will automatically reboot. Check the console (or make an API call) to verify the new IP address has taken affect.

Configure DHCP

You can also change back to using DHCP. Before undertaking this, ensure the network the virtual appliance is on has DHCP enabled.

```
curl -L -X POST 'http://${APPLIANCE_HOST}:8080/v1/management/host/setdhcp' \
  -H 'Accept: application/json'
```

NOTE: once submitted, the virtual appliance will automatically reboot. Check the console to verify the new IP address has taken affect.

Firewall Ports

There are several firewall rules that may need to be enabled to ensure the communication can be made to the virtual appliance:

- 8080/TCP - Used for the Management API to manage the virtual appliance
- 8082/TCP - Speech API for submitting jobs
- 3000/TCP - Monitoring

Using Proxies

If the network that you are deploying your appliance into does not have a direct route to the Internet, you may need to use a proxy server in order to talk to the cloud-based license service. See the relevant section in Licensing (below) for details on how to set this up.

Virtual Appliance Scaling

Real-Time Virtual Appliance Scaling

This section explains how to scale the Real-Time Virtual Appliance, and gives advice on how to make sure you've allocated enough resources for your workload.

Worker Limits

The number of concurrent workers can be restricted using the Management API. This can be used to ensure that the system resources do not get exhausted by clients starting more sessions than expected. The maximum number of concurrent workers is set for the entire system, irrespective of which language packs are being used. The default number of maximum concurrent workers is 1.

View Maximum Workers

Use a GET request to the `maxworkers` endpoint to view the maximum number of workers:

```
curl -L -X GET 'http://${APPLIANCE_HOST}:8080/v1/management/maxworkers' \
   -H 'Accept: application/json' \
   | jq
```

This shows the maximum number of workers that can run concurrently on the appliance. If more sessions are opened by clients using the Speech API then you will receive the job error: `No worker can be scheduled because the service is at capacity`.

Setting Maximum Workers

Before changing the maximum number of concurrent workers for real-time transcription, it is important that the virtual appliance has enough system resources (CPU and RAM) to support the new requirement (see the Real-time Virtual Appliance system requirements). This example shows how to set the maximum number of concurrent workers to 5:

```
curl -L -X POST 'http://${APPLIANCE_HOST}:8080/v1/management/maxworkers' \
    -H 'Accept: application/json' \
    -H 'Content-Type: application/json' \
    -d '{ "count": "5" }'
```

As a rule of thumb, each concurrent worker will require 1 vCPU and up to 2GB RAM.

Batch Virtual Appliance Scaling

This section explains how to scale the Batch Virtual Appliance, and gives advice on how to make sure you've allocated enough resources for your workload.

Worker Limits

The number of concurrent workers (jobs) can be restricted using the Management API. This can be used to ensure that the system resources do not get exhausted by clients starting more transcriptions than expected. The maximum number of concurrent workers is set for the entire system, irrespective of which language packs are being used. The default number of maximum concurrent workers is 1.

View Maximum Workers

Use a GET request to the maxworkers endpoint to view the maximum number of workers:

```
curl -L -X GET 'http://${APPLIANCE_HOST}:8080/v1/management/maxworkers' \
   -H 'Accept: application/json' \
   | jq
```

The response will indicate the maximum number of workers that can run concurrently on the appliance. If more jobs are submitted by clients using the Speech API then these will be queued up and processed once there is spare capacity on the appliance.

Setting Maximum Workers

Before changing the maximum number of concurrent workers, it is important that the virtual appliance has enough system resources (CPU and RAM) to support the new requirement (see the Batch Virtual Appliance system requirements).

This example shows how to set the maximum number of concurrent workers to 5:

```
curl -L -X POST 'http://${APPLIANCE_HOST}:8080/v1/management/maxworkers' \
    -H 'Accept: application/json' \
    -H 'Content-Type: application/json' \
    -d'{ "count": "5" }'
```

As a rule of thumb, each concurrent worker will require 1 vCPU and up to 5GB of RAM (depending on the quality of the audio).

If the number of jobs submitted exceeds the maximum number of concurrent workers then jobs will start to be queued, and the real-time factor (RTF) will increase, meaning you will wait longer for your transcripts to be made available.

Monitoring

Appliance resources can be monitored at a system-wide level. Exhaustion of any of the resources can have a negative impact on the speed of the transcription.

The following resources that can be monitored:

Resource ID (rID)	Description
cpu	Provides the CPU usage across all the vCPU assigned
mem	Provides the total RAM usage of the appliance

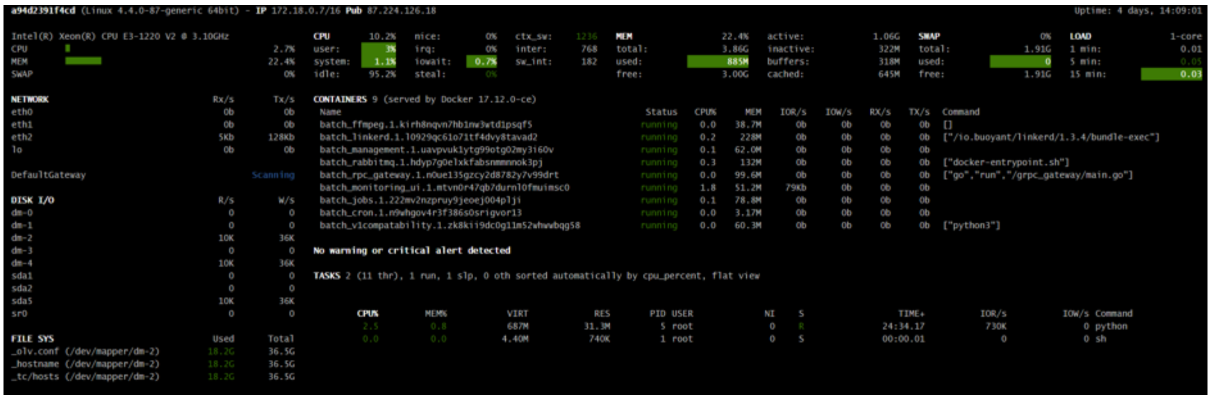
Here is an example GET request for the `mem` (RAM) resource:

```
curl -L -X GET 'http://${APPLIANCE_HOST}:8080/v1/management/resource/mem' \
    -H 'Accept: application/json' \
    | jq
```

Here is an example response:

```
{
    "rId": "mem",
    "percentage": 10.9,
    "value": 0,
    "intValue": 0
}
```

For advanced monitoring, a utility called Glances is available that runs on TCP port 3000. It allows real-time resource stats to be monitored on the Real-time Virtual Appliance. The easiest way to access this is via a web browser using the link http://${APPLIANCE_HOST}:3000/ in the address bar.



```
a94d2391f4cd (Linux 4.4.0-87-generic 64bit) - IP 172.18.0.7/16 Pub 87.224.126.18                                  Uptime: 4 days, 14:09:01

Intel(R) Xeon(R) CPU E3-1220 V2 @ 3.10GHz    CPU      10.2%  nice:     0%  ctx_sw:  1236  MEM       22.4%  active:    1.06G  SWAP     0%  LOAD    1-core
CPU                                  2.7%    user:       3%  irq:      0%  inter:    768  total:    3.86G  inactive:   322M  total: 1.91G  1 min:    0.01
MEM                                 22.4%    system:   1.1%  iowait: 0.7%  sw_int:   182  used:      885M  buffers:    318M  used:      0  5 min:    0.05
SWAP                                   0%    idle:    95.2%  steal:    0%                 free:     3.00G  cached:     645M  free:  1.91G  15 min:   0.03

NETWORK                      Rx/s    Tx/s    CONTAINERS 9 (served by Docker 17.12.0-ce)
eth0                           0b      0b     Name                                            Status   CPU%    MEM  IOR/s  IOW/s  RX/s  TX/s  Command
eth1                           0b      0b     batch_ffmpeg.1.kirh8nqvn7hb1mw3wtd1psqf5         running   0.0  38.7M     0b     0b    0b    0b  []
eth2                          5Kb   128Kb     batch_linkerd.1.10929qc61o71tf4dvy8tavad2        running   0.2   228M     0b     0b    0b    0b  ["/io.buoyant/linkerd/1.3.4/bundle-exec"]
lo                             0b      0b     batch_management.1.uavpvuk1ytg99otg02my3i60v     running   0.1  62.0M     0b     0b    0b    0b
                                              batch_rabbitmq.1.hdyp7g0elxkfabsnmmnnok3pj       running   0.3   132M     0b     0b    0b    0b  ["docker-entrypoint.sh"]
DefaultGateway                   Scanning     batch_rpc_gateway.1.n0ue135gzcy2d8782y7v99drt    running   0.0  99.6M     0b     0b    0b    0b  ["go","run","/grpc_gateway/main.go"]
                                              batch_monitoring_ui.1.mtvn0r47qb7durnl0fmuimsc0  running   1.8  51.2M   79Kb     0b    0b    0b
DISK I/O                      R/s     W/s     batch_jobs.1.222mv2nzpruy9jeoej004plji           running   0.1  78.8M     0b     0b    0b    0b
dm-0                            0       0     batch_cron.1.n9whgov4r3f386s0srigvor13           running   0.0  3.17M     0b     0b    0b    0b
dm-1                            0       0     batch_v1compatability.1.zk8kii9dc0g11m52whwwbqg58 running  0.0  60.3M     0b     0b    0b    0b  ["python3"]
dm-2                          10K     36K
dm-3                            0       0    No warning or critical alert detected
dm-4                          10K     36K
sda1                            0       0    TASKS 2 (11 thr), 1 run, 1 slp, 0 oth sorted automatically by cpu_percent, flat view
sda2                            0       0
sda5                          10K     36K
sr0                             0       0          CPU%    MEM%     VIRT     RES   PID USER      NI  S        TIME+     IOR/s      IOW/s Command
                                                    2.5     0.8     687M   31.3M     5 root       0  R     24:34.17      730K          0 python
FILE SYS                     Used   Total           0.0     0.0    4.40M    740K     1 root       0  S     00:00.01         0          0 sh
_olv.conf (/dev/mapper/dm-2) 18.2G  36.5G
_hostname (/dev/mapper/dm-2) 18.2G  36.5G
_tc/hosts (/dev/mapper/dm-2) 18.2G  36.5G
```

It is also possible to access the Glances API using XML-RPC or HTTP REST (for JSON output), for example:

```
curl -L -X GET 'http://${APPLIANCE_HOST}:3000/api/2/mem/percent' \
    -H 'Accept: application/json' \
    | jq
```

For more information on the HTTP REST interface, consult the Glances documentation.

Services

The virtual appliance has internal services that are required for operation.

There are system-wide services, and services specific to transcription workers for a given language.

For the Batch Virtual Appliance, this table lists the services:

Service Name (Begins with)	Description	Required Status
batch_ffmpeg...	Conversion of audio	Running
batch_rpc_gateway...	RPC endpoint	Running
batch_license...	Licensing service	Running
batch_linkerd...	Internal Networking	Running
batch_notifier...	Callback function	Running
batch_management...	Management functions	Running
batch_rabbitmq...	Job Queue	Running
batch_monitoring_ui...	Monitoring Web GUI	Running
batch_cron...	Completed job clean-up	Running
batch_v1compatibility...	V1 REST API	Running
batch-jobs...	Used to perform ASR and transcription	Running
batch_nginxlb...	HTTP gateway	Running

The service will always have a current state, these states include:

Service Status	Description
running	Service has started and is running
created	Service is in the process of starting
exited	Service has stopped and is no longer running

Service status

This can be used to ensure all services have the required status to operate (see table above). Example: GET to list services and corresponding status:

```
curl -L -X GET 'http://${APPLIANCE_HOST}:8080/v1/management/services' \
    -H 'Accept: application/json' \
    | jq
```

If the appliance has been licensed then you will see a return like this (for the Batch Virtual Appliance):

```
{
  "service_status": [
    {
      "service": "batch_management.1.iu9c86o1weubdmi99wddlcklr",
      "status": "running"
    },
    {
      "service": "batch_ffmpeg.1.jre8d5lempzmsqfki9o871a62",
      "status": "running"
    },
    {
      "service": "batch_notifier.1.idqxysv0srzeg2vkkorh1zjfh",
      "status": "running"
    },
    {
      "service": "batch_linkerd.1.af4t03setx3m64s15s9yawysl",
      "status": "running"
    },
    {
      "service": "batch_batch-cron.1.y5ql8ryyqlxwlxy84q9q3lfrn",
      "status": "running"
    },
    {
      "service": "batch_license.1.7ecytnlzd6hso3jxauvbyvfyi",
      "status": "running"
    },
    {
      "service": "batch_rabbitmq.1.l8ny2q6b2xhz0yr5bxwodbtog",
      "status": "running"
    },
    {
      "service": "batch_monitoring_ui.1.v180r4tq7dlcbfhc3vxyukpdo",
      "status": "running"
    },
    {
      "service": "batch_rpc_gateway.1.fb3ryh2a4d41sy628bhiogyx4",
      "status": "running"
    },
    {
      "service": "batch_postgres.1.wfy284tvznpnmgi22xaw11b55",
      "status": "running"
    },
    {
      "service": "batch_jobs.1.z4vftf8vv42uzmxd4ra3235ao",
      "status": "running"
    },
    {
      "service": "batch_v1compatibility.1.j5rvj3wpqwdfnl848ci42iix2",
      "status": "running"
    }
  ]
}
```

Note: After a service is restarted it will have a random string identifier post fixed to its name.

Service restart

If required for troubleshooting you may need to restart all the services. During the restart, all transcription will stop. The following command performs a service restart:

```
$ curl -X DELETE 'http://<APPLIANCE HOST>:8080/v1/management/services' \
    -H 'Accept: application/json'
```

Access Logs

The individual services on the system provide log files that can be collected to help with troubleshooting. The service name will need to be provided when retrieving logs. See above for instructions on how to view the names of the running services

The following parameters are available when accessing logs:

Name	Description	Required Status
name	Name of the service to collect the logs for	Required
count	Number of log lines wanted, defaults to 100; if all lines are to be returned set to -1	Optional

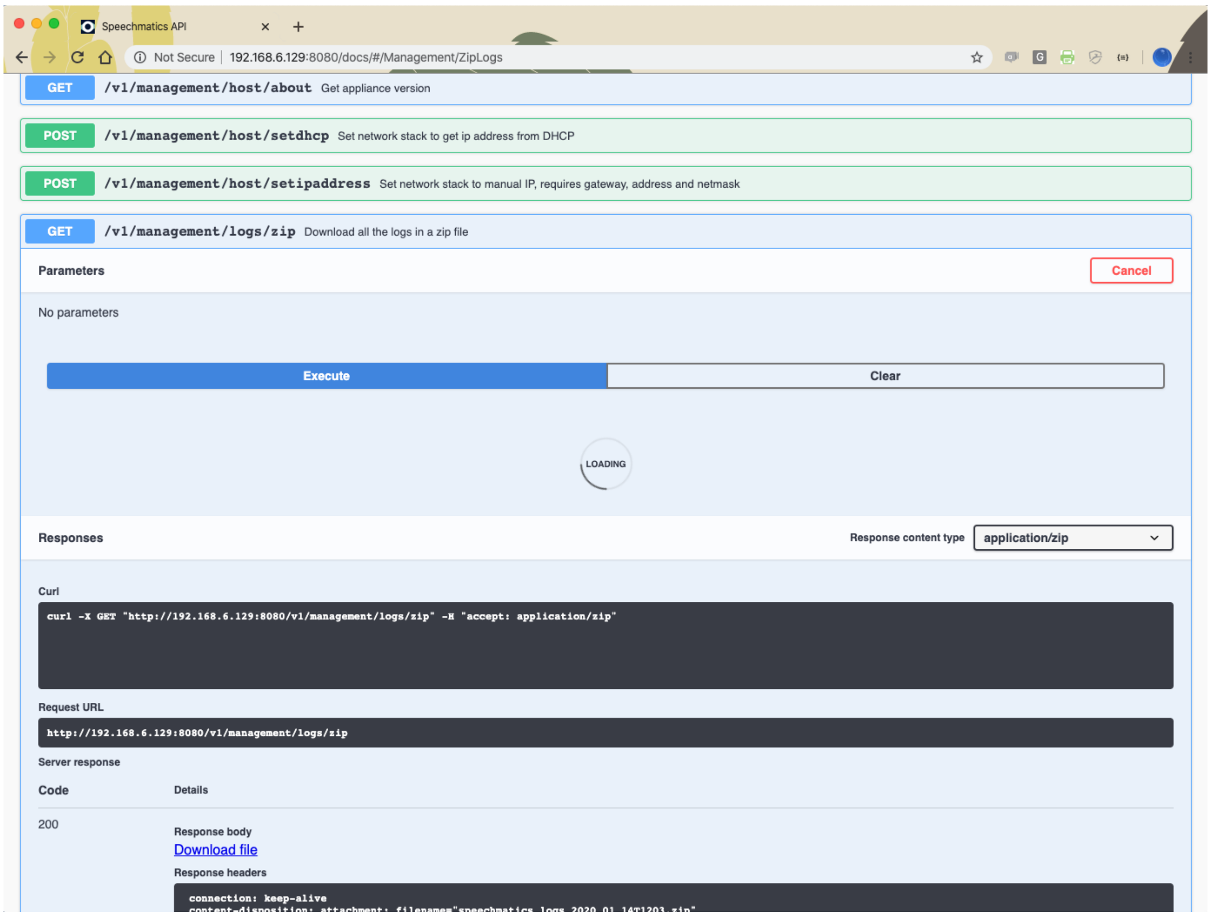
Example: GET to retrieve logs for batch_monitoring_ui service:

```
curl -L -X GET
'http://${APPLIANCE_HOST}:8080/v1/management/logs/batch_monitoring_ui.1.mtvn0r47qb7durnl0fmuimsc0'
 \
    -H 'Accept: application/json' \
    | jq -r '.log_lines'
```

If you want to download *all* the logs (in order to provide information for a support ticket for instance) as a ZIP file, then it is possible to do this using the following command:

```
curl -L -X GET 'http://${APPLIANCE_HOST}:8080/v1/management/logs/zip' \
    -H 'Accept: application/json' \
    -o ./speechmatics.zip
```

It is also possible to do this directly from the Swagger UI by entering in the following URL to you r browser: http://192.168.6.129:8080/docs/#/Management/ZipLogs, and then clicking on the download link when the ZIP file is ready.



GET /v1/management/host/about Get appliance version

POST /v1/management/host/setdhcp Set network stack to get ip address from DHCP

POST /v1/management/host/setipaddress Set network stack to manual IP, requires gateway, address and netmask

GET /v1/management/logs/zip Download all the logs in a zip file

Parameters

No parameters

Responses

Curl

```
curl -X GET "http://192.168.6.129:8080/v1/management/logs/zip" -H "accept: application/zip"
```

Request URL

```
http://192.168.6.129:8080/v1/management/logs/zip
```

Server response

Code	Details
200	Response body Download file Response headers

```
connection: keep-alive
content-disposition: attachment; filename="speechmatics_logs_2020_01_14T1203.zip"
```

System restart

If the virtual appliance becomes unresponsive, there might be a need to restart it. If this is the case, it's recommended that the system is restarted through the management API, like this:

```
curl -L -X DELETE 'http://${APPLIANCE_HOST}:8080/v1/management/reboot'
```

If the Management API is not available, then you should reboot the appliance from the hypervisor console. For further information on how to restart the virtual machine via the console, please follow the manufacturers advice.

Troubleshooting

There may be times unexpected behavior is observed with the Real-time Virtual Appliance. If this is the case the following should be performed/checked:

- Check the license is valid (see licensing)
- Check the worker services are running
- Check the resources (CPU, memory & disk) to ensure they are not exhausted
- Restart all the services
- Restart the virtual appliance
- Collect logs and contact Speechmatics support: support@speechmatics.com.

Transcription job failure

If your transcription job fails with an `error` job status, more information can be found by looking at the logs from the *jobs* container (using the Management API, as previously described). Search the logs for the job id corresponding with your failure. If you see a `SoftTimeLimitExceeded` exception, this indicates that the job took longer than anticipated and as such was terminated. This is typically caused by poor VM performance, in particular slow disk IO operations

(IOPS). If issues persist it may be necessary to improve the disk IO performance on the underlying host, or you may need to increase the RAM available to the VM such that memory caches can be taken advantage of. Please consult the section above on Host requirements, and the optimization advice specific to your hypervisor to ensure that you are not over-committing your compute resources.

Illegal instruction errors

If jobs fail repeatedly and you see `Illegal instruction` errors in the log information for these jobs then it is likely that the host hardware you are running on does not support AVX. The host machine requirements for the Real-time Virtual Appliance must meet the 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 can check this by looking in the management log when the appliance starts up. If you see a message like this:

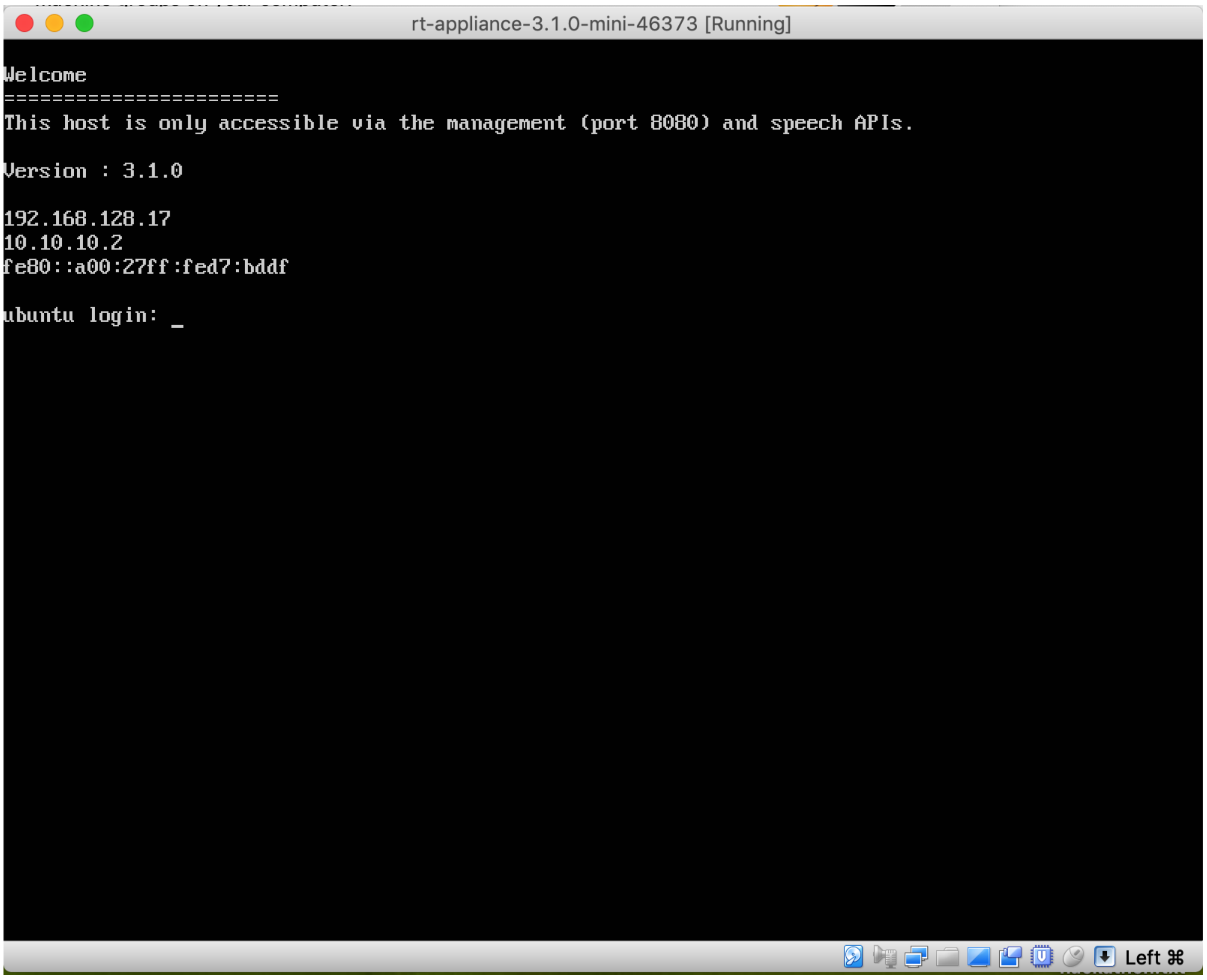
```
2019-03-26 16:53:07,136    sm_management.app    ERROR    Processor not AVX capable. Tensorflow
language models cannot run.
```

Then it means that your host's CPU does not support AVX, or that your hypervisor does not have AVX support.

A console is available to help with advanced troubleshooting in the event that the Management API is unavailable. It is described in the next section.

Console for Advanced Troubleshooting

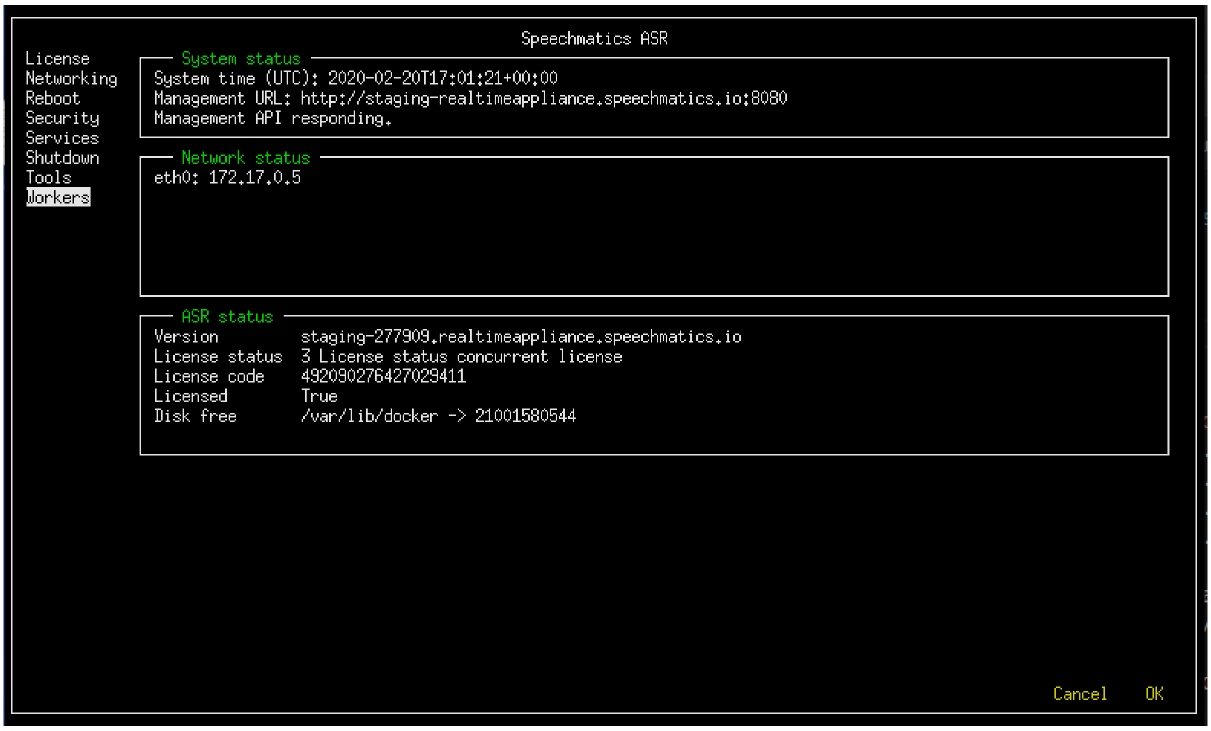
In the event that the Management API is unavailable (it is unresponsive, or there is no network connectivity) you can use the console to restore network connectivity, restart the appliance, or view information about services. To use this you need to use your hypervisor's GUI to access the logon screen for the appliance.



```
Welcome
=======================
This host is only accessible via the management (port 8080) and speech APIs.

Version : 3.1.0

192.168.128.17
10.10.10.2
fe80::a00:27ff:fed7:bddf

ubuntu login: _
```

From this screen use the CTRL+ALT+F5 key combination to get to the console. Once you are in the console you have the following menu options available:

- License
- Networking
- Reboot
- Services
- Shutdown
- Tools
- Workers



```
                                  Speechmatics ASR
 License      ─ System status ─────────────────────────────────────────────────
 Networking    System time (UTC): 2020-02-20T17:01:21+00:00
 Reboot        Management URL: http://staging-realtimeappliance.speechmatics.io:8080
 Security      Management API responding.
 Services
 Shutdown     ─ Network status ────────────────────────────────────────────────
 Tools         eth0: 172.17.0.5
 Workers

              ─ ASR status ────────────────────────────────────────────────────
               Version         staging-277909.realtimeappliance.speechmatics.io
               License status  3 License status concurrent license
               License code    492090276427029411
               Licensed        True
               Disk free       /var/lib/docker -> 21001580544

                                                                  Cancel    OK
```

The home screen shows high-level information about the appliance: IP addressing, software version and license status.

In the **System status** panel the **API responding** indicator shows the state of the Management API. **Network status** shows the IP address the appliance is currently configured with, and **ASR status** shows the license state and available storage space on the appliance.

In the event that you need to provide information to Speechmatics support you may be asked to connect to the console and provide this information. This section provides some tips on how to use the console to perform basic troubleshooting yourself.

Note: We recommend that you use the Management API for most troubleshooting tasks as it is easier to use. The console can be used in the event that the Management API is unavailable, but it does not provide all the features of the Management API.

License

The Licensing Troubleshooting section provides detailed instructions on how to use the Management API to resolve common licensing issues. If you cannot use the Management API then you can still use console to check the license status and perform basic licensing steps.

Networking

You can use the networking option to configure a static IP address, or use DHCP.

Reboot and Shutdown

Reboot and Shutdown options exist to allow you to restart or shutdown the appliance from the console. You will be asked to select OK to confirm.

Services

From this menu you can access the list of services that are running on the appliance. Selecting a service shows the log entries for that service.

Tools

This menu allows you to access a number of useful Unix utilities that can be used for advanced troubleshooting. In order to help progress a support ticket you may be asked to provide the output (ie. a screenshot) from running one of these commands.

Workers

This allows you to view and change the maximum number of workers allowed to run concurrently.

Security

The appliance is designed to be installed within your own security perimeter. It has its own firewall installed to only allow ingress to ports that are required for its management, monitoring and Speech APIs.

Overview

The appliance uses a microservices architecture running on a customized Ubuntu machine. AppArmor default security policies are used to protect the OS and running applications on the appliance.

Data on the appliance (including audio and video data that is submitted via the Speech API, logs, and output transcripts) are encrypted on disk.

Firewall Ports

There are several firewall rules that may need to be enabled to ensure the communication can be made to the virtual appliance:

- 8080/TCP - Used for the Management API to manage the virtual appliance
- 3000/TCP - Monitoring (Glances)
- 8082/TCP - REST Speech API for submitting jobs (batch ASR)
- 9000/TCP - Websocket Speech API for real-time ASR

Securing your Deployment

The Websocket Speech API for real-time uses the secure `wss` protocol (using a self-signed certificate). However, access to the Management API, Monitoring API (Glances) and Speech API is not secured (`http` only), and no authorization tokens or passwords are required for access to the APIs. It is therefore up to the customer to deploy the appliance behind a load balancer or gateway that can provide those features if you need them. This is especially important if you are intending to deploy your appliances onto a public cloud (for example as an Amazon EC2 instance).

Custom Dictionary Cache

:::note Cache availability The custom dictionary cache is only available in the Real-time Virtual Appliance. :::

The Speechmatics Real-time Virtual Appliance includes a cache mechanism for custom dictionaries. By using this cache mechanism, the appliance is able to reduce the time needed for processing a custom dictionary before starting the recognition of an audio stream.

Once the Real-time Virtual Appliance has started a recognition session with a given custom dictionary, any subsequent streams with the identical custom dictionary sent to the appliance by any client will benefit from a reduced setup time. Transcription requests using an already cached custom dictionary will start recognition in less than 3 seconds if the custom dictionary is up to our recommended limit of 1000 words.

:::note V1 API When using V1 API the setup time spent processing a cached entry is about 3 seconds longer compared to V2 API. :::

Size available

The size available for storing Custom Dictionary Cache entries depends on the variant of the Real-time Virtual Appliance.

Variant	Cache Space
nano	100MB
mini	150MB
midi	200MB
maxi	250MB
plus	300MB

Size of cache entries

The entry size varies depending on the amount of information included in the custom dictionary. For guidance, the table below displays some values for different custom dictionaries.

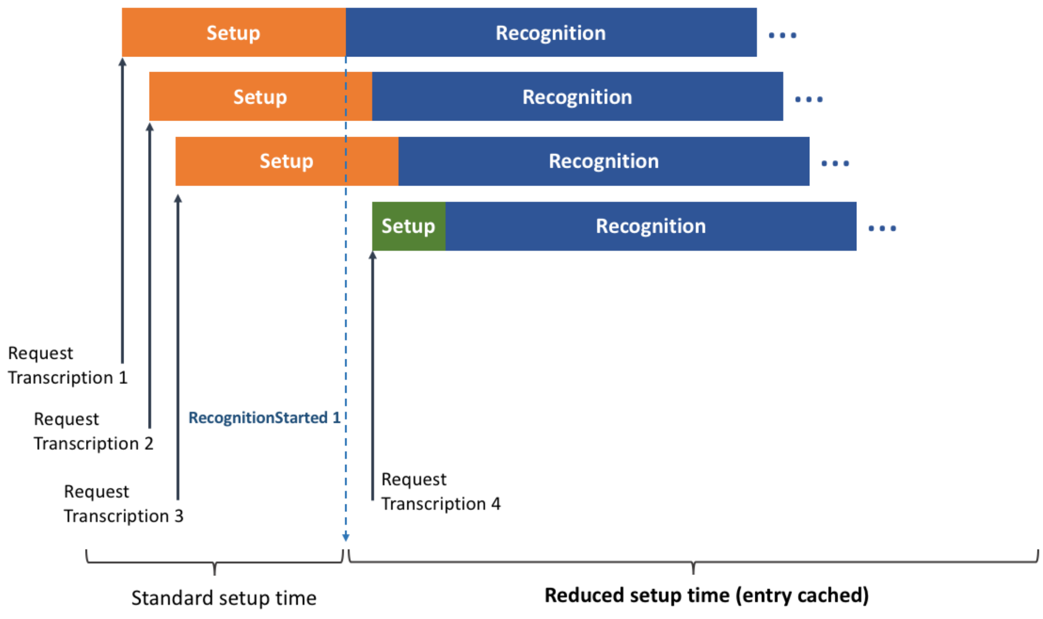
Custom Dictionary	Cache Used Bytes	Cache Entry Size
None (empty cache)	14KiB	NA
1000 words	120KiB	106KiB
1000 words + 1 sounds like each	188KiB	174KiB

:::note Size of empty cache The custom dictionary cache needs to keep certain metadata files in order to function. For this reason the used_bytes reported will never be `0`, even if the cache is not storing any entry. The amount of storage used by an empty cache is typically around `14KiB`. :::

Cache life cycle

When a custom dictionary is used for a transcription, it is automatically cached by the Real-time Virtual Appliance. This allows for a reduced setup time on any further transcriptions using the same custom dictionary. Custom dictionary cache is persisted in disk, making it available between reboots. If there is no space left in the cache for a new entry, entries are deleted in order of when they were last used when submitting a job.

Cache entries are ready to be consumed by any subsequent transcription request, from any client, after a `RecognitionStarted` message is emitted by the Real-time Appliance. For this reason, transcription requests made in parallel, each with the same custom dictionary that is new to the appliance, won't benefit from a reduced setup time.



A custom dictionary could be cached beforehand by requesting a transcription for an empty audio file. After receiving the `RecognitionStarted` message from the appliance, other requests using the same custom dictionary will benefit from a reduced startup time.

Administering the Cache

When the Real-time Virtual Appliance is started for the first time the cache will be empty. The management API allows you to retrieve cache usage data and to purge the cache contents.

View Cache Usage

Cache usage reports the maximum cache size and the used number of bytes in the cache.

In order to retrieve usage statistics, send a GET request to the `/v1/management/cache` endpoint:

```
curl -L -X GET http://${APPLIANCE_HOST}:8080/v1/management/cache \
-H 'Accept: application/json' \
| jq
```

Here is an example response:

```
{
  "total_bytes": "105188352",
  "used_bytes": "192512"
}
```

Purge Cache Contents

It is possible to remove all contents in the cache.

In order to purge the cache contents, send a DELETE request to the `/v1/management/cache` endpoint:

```
curl -L -X DELETE http://${APPLIANCE_HOST}:8080/v1/management/cache \
-H 'Accept: application/json' \
| jq
```

Here is an example response:

```
{
  "confirmation": "Custom dictionary cache purged successfully",
  "usage": {
    "total_bytes": "105188352",
    "used_bytes": "14336"
  }
}
```

Introduction

Overview

The Websocket Speech API allows communication from a client application over a WebSocket connection to the Speechmatics ASR engine on port 9000. This connection provides the ability to convert a stream of audio into a transcript providing the words and timing information as the live audio is processed.

The WebSocket can be used directly as described in this document. Speechmatics provides Python libraries that can be used to wrap the WebSocket interface, and provide the ability to connect directly to a microphone or RTSP feed.

A Python wrapper library called **smwebsocket-py** is provided that shows how to use the new V2 API.

Terms

For the purposes of this guide the following terms are used.

Term	Description
Client	An application connecting to the Real-time Virtual Appliance using the Speech API. The client will provide audio containing speech, and process the transcripts received as a result.
Server	The Real-Time Container or Applaince providing the ASR engine which processes human speech and returns transcripts in real-time.
Management API	The REST API that allows administrators to manage the virtual appliance over port 8080. To access the documentation you can use the following URI: https://${APPLIANCE_HOST}:8080/help/, where ${APPLIANCE_HOST} is the IP address or hostname of your appliance.
Speech API	The WebSocket API that allows users to submit ASR jobs over server port 9000. The endpoint `wss://${APPLIANCE_HOST}:9000/v2` is used for the Speech API. This is the API that is described in this document.
Real-Time Container	A Docker container that provides real-time ASR transcription.
Real-Time Virtual Appliance	An appliance (VM) that provides real-time ASR transcription.

Input Formats

A wide variety of input sources are supported, including:

- Raw audio (microphone)
- File (.wav, .mp3 and .mp4 are tested)
- RTMP, RTSP or HLS stream

A Python client library is supplied that shows how to get data from some of these sources.

If you attempt to use an audio file format that is not supported, then you will see the following error message:

```
Error / job_error: An internal error happened while processing your file. Please check that your
audio format is supported.
```

Transcript Outputs

The output format from the Speech API is JSON. It is described in detail in the API Reference. There are two types of transcript that are provided: final transcripts and partial transcripts. Which one you decide to consume will depend on your use case, and your latency and accuracy requirements.

Final transcripts

Final transcripts are sentences or phrases that are provided based on the Speechmatics ASR engine automatically determining the best point at which to provide the transcript to the client. These transcripts occur at irregular intervals. Once output, these transcripts are considered final, they will not be updated after output.
The timing of the output is determined by Speechmatics based on the ASR algorithm. This is affected by pauses in speech and other parameters resulting in a latency between audio input and output of up to 10 seconds. This 10 second default can be changed with the `max_delay` property in `transcription_config` when starting the recognition session. Final transcripts provide the most accurate transcription.

Partial transcripts

A partial transcript is a transcript that can be updated at a later point in time. It is believed to be correct at the time of output, but once further audio data is available, the Speechmatics ASR engine may use the additional context that is available to update parts of the transcript that have already been output. These transcripts are output as soon as any transcript is available, regardless of accuracy, and are therefore available at very low latency. These are the fastest way to consume transcripts but at the cost of needing to accept updates at a later point. Partial transcripts provide latency values between audio input and initial output of less than 1 second. This is the least accurate transcription method, but can be used in conjunction with the final transcripts to enable fast display of the transcript, adjusting over time. Partial transcripts must be explicitly enabled (using the `enable_partials` setting) in the config for the session, otherwise only final transcripts will be output.

Advanced Punctuation

Some language models (English, French, German and Spanish currently) now support *advanced punctuation*. This uses machine learning techniques to add more naturalistic punctuation and make the transcript more readable.

The WebSocket Protocol

WebSockets are used to provide a two-way transport layer between your client and the Real-Time Appliance, enabling use with most modern web-browsers, and programming languages. See RFC 6455 for the detailed specification of the WebSocket protocol.

The wire protocol used with the WebSocket consists mostly of packets of stringified JSON objects which comprise a message name, plus other fields that are message dependant. The only exception is that a binary message is used for transmitting the audio.

You can develop your real time client using any programming language that supports WebSockets. This document provides a list of the messages that are required for the client and server communication. Some of the messages are required to be sent in a particular order (outlined below) whilst others are optional. There are some examples provided at the end of this document on how to access the Speech API using JavaScript. For a working Python example, please refer to our reference Python client implementation `smwebsocket-py`, which is available for download. Please contact support@speechmatics.com for details of how to download this Python client.

Realtime API

This page specifies the Realtime API at its current state. The basic elements in the communication are the following:

- **Client** - An application connecting to the API, providing the audio and processing the transcripts received from the **Server**.
- **Server** (also called **API**) - An entry point of the API, allows external connections and provides the transcripts back.
- **Worker** - An internal speech recognizer. This is an internal entity that actually runs the heavy speech recognition.

This is a specification for Speechmatics Real-time API version 2.1.0.

Client ⇔ API endpoint

The communication is done using WebSockets, which are implemented in most of the modern web-browsers, as well as in many common programming languages (namely C++ and Python, for instance using http://autobahn.ws/).

Messages

Each message that the **Server** accepts is a stringified JSON object with the following fields:

- `message` (String): The name of the message we are sending. Any other fields depend on the value of the `message` and are described below.

The messages sent by the **Server** to a **Client** are stringified JSON objects as well.

The only exception is a binary message sent from the **Client** to the **Server** containing a chunk of audio which will be referred to as `AddAudio`.

The following values of the `message` field are supported:

StartRecognition

Initiates recognition, based on details provided in the following fields:

- `message: "StartRecognition"`
- `audio_format` (Object:AudioType): Required. Audio stream type you are going to send: see Supported audio types.
- `transcription_config` (Object:TranscriptionConfig): Required. Set up configuration values for this recognition session, see Transcription config.

A `StartRecognition` message must be sent exactly once after the WebSocket connection is opened. The client must wait for a `RecognitionStarted` message before sending any audio.

In case of success, a message with the following format is sent as a response:

- `message: "RecognitionStarted"`
- `id` (String): Required. A randomly-generated GUID which acts as an identifier for the session. e.g. "807670e9-14af-4fa2-9e8f-5d525c22156e".

In case of failure, an error message is sent, with `type` being one of the following: `invalid_model, invalid_audio_type, not_authorised, insufficient_funds, not_allowed, job_error`

An example of the `StartRecognition` message:

```
{
  "message": "StartRecognition",
  "audio_format": {
    "type": "raw",
    "encoding": "pcm_f32le",
    "sample_rate": 16000
  },
  "transcription_config": {
    "language": "en",
    "output_locale": "en-US",
    "additional_vocab": ["gnocchi", "bucatini", "bigoli"],
    "diarization": "speaker_change",
    "enable_partials": true,
    "punctuation_overrides": {
      "permitted_marks": [",", "."]
    }
  }
}
```

The example above starts a session with the Global English model ready to consume raw PCM encoded audio with float samples at 16kHz. It also includes an `additional_vocab` list containing the names of different types of pasta. `speaker_change` diarization is enabled, and partials are enabled meaning that `AddPartialTranscript` messages will be received as well as `AddTranscript` messages. Punctuation is configured to restrict the set of punctuation marks that will be added to only commas and full stops.

AddAudio

Adds more audio data to the recognition job started on the WebSocket using `StartRecognition`. The server will only accept audio after it is initialized with a job, which is indicated by a `RecognitionStarted` message. Only one audio stream in one format is currently supported per WebSocket (and hence one recognition job). `AddAudio` is a binary message containing a chunk of audio data and no additional metadata.

AudioAdded

If the `AddAudio` message is successfully received, an AudioAdded message is sent as a response. This message confirms that the **Server** has accepted the data and will make a corresponding **Worker** process it. If the **Client** implementation holds the data in an internal buffer to resubmit in case of an error, it can safely discard the corresponding data after this message. The following fields are present in the response:

- `message: "AudioAdded"`
- `seq_no` (Int): Required. An incrementing number which is equal to the number of audio chunks that the server has processed so far in the session. The count begins at 1 meaning that the 5th `AddAudio` message sent by the client, for example, should be answered by an `AudioAdded` message with `seq_no` equal to 5.

Possible errors:

- `data_error`, `job_error`, `buffer_error`

When sending audio faster than real time (for instance when sending files), make sure you don't send too much audio ahead of time. For large files, this causes the audio to fill out networking buffers, which might lead to disconnects due to WebSocket ping/pong timeout. Use AudioAdded messages to keep track what messages are processed by the engine, and don't send more than 10s of audio data or 500 individual AddAudio messages ahead of time (whichever is lower).

Implementation details

Under special circumstances, such as when the client is sending the audio data faster than real time, the **Server** might read the data slower than the **Client** is sending it. The **Server** will not read the binary `AddAudio` message if it is larger than the internal audio buffer on the **Server**. Note that for each **Worker**, there is a separate buffer. In that case, the server will read any messages coming in on the WebSocket, until enough space is made in the buffer by passing the data to a corresponding **Worker**. The **Client** will only receive the corresponding AudioAdded response message once the binary data is read. The WebSocket might eventually fill all the TCP buffers on the way, causing a corresponding WebSocket to fail to write and close the connection with prejudice. The **Client** can use the bufferedAmount attribute of the WebSocket to prevent this.

AddTranscript

This message is sent from the **Server** to the **Client**, when the **Worker** has provided the **Server** with a segment of transcription output. It contains the transcript of a part of the audio the **Client** has sent using `AddAudio` - the **final transcript**. These messages are also referred to as **finals**. Each message corresponds to the audio since the last

`AddTranscript` message. The transcript is final - any further `AddTranscript` or `AddPartialTranscript` messages will only correspond to the newly processed audio. An `AddTranscript` message is sent when we reach an endpoint (end of a sentence or a phrase in the audio), or after 10s if we haven't reached such an event. This timeout can be further configured by setting `transcription_config.max_delay` in the `StartRecognition` message.

- `message: "AddTranscript"`
- `metadata` (Object): Required.
  - `start_time` (Number): Required. An approximate time of occurrence (in seconds) of the audio corresponding to the beginning of the first word in the segment.
  - `end_time` (Number): Required. An approximate time of occurrence (in seconds) of the audio corresponding to the ending of the final word in the segment.
  - `transcript` (String): Required. The entire transcript contained in the segment in text format. Providing the entire transcript here is designed for ease of consumption; we have taken care of all the necessary formatting required to concatenate the transcription results into a block of text. This transcript lacks the detailed information however which is contained in the `results` field of the message - such as the timings and confidences for each word.
- `results` (List:Object):
  - `type` (String): Required. One of 'word', 'punctuation' or 'speaker_change'. 'word' results represent a single word. 'punctuation' results represent a single punctuation symbol. 'word' and 'punctuation' results will both have one or more `alternatives` representing the possible alternatives we think the word or punctuation symbol could be. 'speaker_change' results however will have an empty `alternatives` field. 'speaker_change' results will only occur when using speaker_change diarization.
  - `start_time` (Number): Required. The start time of the result **relative to** the start_time of the whole segment as described in `metadata`.
  - `end_time` (Number): Required. The end time of the result **relative to** the start_time of the segment in the message as described in `metadata`. Note that punctuation symbols and speaker_change results are considered to be zero-duration and thus for those results `start_time` is equal to `end_time`.
  - `channel` (String): Optional. For multi-channel audio, this indicates the channel in the audio for which the result comes from.
  - `is_eos` (Boolean): Optional. Only present for 'punctuation' results. This indicates whether or not the punctuation mark is considered an end-of-sentence symbol. For example full-stops are an end-of-sentence symbol in English, whereas commas are not. Other languages, such as Japanese, may use different end-of-sentence symbols.
  - `alternatives` (List:Object): Optional. For 'word' and 'punctuation' results this contains a list of possible alternative options for the word/symbol.
    - `content` (String): Required. A word or punctuation mark.
    - `confidence` (Number): Required. A confidence score assigned to the alternative. Ranges from 0.0 (least confident) to 1.0 (most confident).
    - `display` (Object): Optional. Information about how the word/symbol should be displayed.
      - `direction` (String): Required. Either 'ltr' for words that should be displayed left-to-right, or 'rtl' vice versa.
    - `language` (String): Optional. The language that the alternative word is assumed to be spoken in. Currently this will always be equal to the language that was requested in the initial `StartRecognition` message. In the future we may support multi-language transcription and thus this field may become more useful.
    - `speaker` (String): Optional. An identifier for the speaker that spoke the alternative word. Currently this field is unused since the Realtime Appliance does not support speaker diarization as the Batch Appliance does. In the future we may extend support for this diarization mode in which case the field may become more useful.

AddPartialTranscript

A partial-transcript message. The structure is the same as `AddTranscript`. A partial transcript is a transcript that can be changed and expanded by a future `AddTranscript` or `AddPartialTranscript` message and corresponds to the part of audio since the last `AddTranscript` message. For `AddPartialTranscript` messages the `confidence` field for `alternatives` has no meaning and will always be equal to 0.

Partials will only be sent if `transcription_config.enable_partials` is set to `true` in the `StartRecognition` message.

SetRecognitionConfig

Allows the **Client** to configure the recognition session even after the initial `StartRecognition` message.

- `message: "SetRecognitionConfig"`
- `transcription_config` (Object:TranscriptionConfig): A TranscriptionConfig object containing the new configuration for the session, see Transcription config.

The following is an example of such a configuration message:

```
{
  "message": "SetRecognitionConfig",
  "transcription_config": {
    "language": "en",
    "max_delay": 3.5,
    "enable_partials": true
  }
}
```

Note: The `language` property is a mandatory element in the `transcription_config` object; however it is not possible to change the language mid-way through the session (it will be ignored if you do). It is possible to modify the following settings through a **SetRecognitionConfig** message:

- `max_delay`
- `enable_partials`
- `output_locale`
- `punctuation_overrides`
- `diarization` (`speaker_change` | `none`)
- `additional_vocab`

You should be aware that there is a performance penalty (latency degradation and memory increase) from using `additional_vocab`, especially if you intend to load in a large word list. When initialising a session that uses `additional_vocab` in the config you should expect a delay of up to 15 seconds, and an additional 800MB to 1700MB of memory (depending on the size of the list).

EndOfStream

This message is sent from the Client to the API to announce that it has finished sending all the audio that it intended to send. No more `AddAudio` message are accepted after this message. The Server will finish processing the audio it has received already and then send an EndOfTranscript message. This message is usually sent at the end of file or when the microphone input is stopped.

- `message: "EndOfStream"`
- `last_seq_no` (Int): Required. The total number of audio chunks sent (in the `AddAudio` messages).

EndOfTranscript

Sent from the API to the Client when the API has finished all the audio, as marked with the `EndOfStream` message. The API sends this only after it sends all the corresponding `AddTranscript` messages first. Upon receiving this message the Client can safely disconnect immediately because there will be no more messages coming from the API.

Supported audio types

An `AudioType` object always has one mandatory field `type`, and potentially more mandatory fields based on the value of `type`. The following types are supported:

`type: "raw"`

Raw audio samples, described by the following additional mandatory fields:

- `encoding` (String): Encoding used to store individual audio samples. Currently supported values:
  - `pcm_f32le` - Corresponds to 32 bit float PCM used in the WAV audio format, little-endian architecture.
  - `pcm_s16le` - Corresponds to 16 bit signed integer PCM used in the WAV audio format, little-endian architecture.
  - `mulaw` - Corresponds to 8 bit µ-law (mu-law) encoding.
- `sample_rate` (Int): Sample rate of the audio

`type: "file"`

Any audio/video format supported by GStreamer. The `AddAudio` messages have to provide all the file contents, including any headers. The file is usually not accepted all at once, but segmented into reasonably sized messages.

Example `audio_format` field value: `audio_format: {type: "raw", encoding: "pcm_s16le", sample_rate: 44100}`

Transcription config

A `TranscriptionConfig` object specifies various configuration values for the recognition engine. All the values are optional, using default values when not provided.

- `language` (String): Required. Language model to process the audio input, normally specified as an ISO language code e.g. 'en'.
- `additional_vocab` (List:AdditionalWord): Optional. Configure **additional words**. See Additional words. Default is an empty list.
- `diarization` (String): Optional. The speaker diarization method to apply to the audio. The default is "none" indicating that no diarization will be performed. An alternative option is "speaker_change" diarization in which the ASR system will attempt to detect any changes in speaker. Speaker changes are indicated in the results using an object with a `type` set to `speaker_change`.
- `enable_partials` (Boolean): Optional. Whether or not to send partials (i.e. `AddPartialTranscript` messages) as well as finals (i.e. `AddTranscript` messages). The default is `false`.
- `max_delay` (Number): Optional. Maximum delay in seconds between receiving input audio and returning partial transcription results. The default is 10. The minimum and maximum values are 2 and 20.
- `output_locale` (String): Optional. Configure **output locale**. See Output locale. Default is an empty string.
- `punctuation_overrides` (Object:PunctuationOverrides): Optional. Options for controlling punctuation in the output transcripts. See Punctuation overrides.
- `speaker_change_sensitivity` (Number): Optional.: Controls how responsive the system is for potential speaker changes. The value ranges between zero and one. High value indicates high sensitivity, i.e. prefer to indicate a speaker change if in doubt. The default is 0.4. This setting is only applicable when using `"diarization": "speaker_change"`.

Additional words

Additional words expand the standard recognition dictionary with a list of words or phrases called **additional words**. An **additional word** can also be a phrase, as long as individual words in the phrase are separated by spaces. A pronunciation of those words is generated automatically or based on a provided `sounds_like` field. This is intended for adding new words and phrases, such as domain-specific terms or proper names. Better results for domain-specific words that contain common words can be achieved by using phrases rather than individual words (such as `action plan`).

`AdditionalWord` is either a `String` (the **additional word**) or an `Object`. The object form was introduced in 0.7.0. The object form has the following fields:

- `content` (String): The **additional word**.
- `sounds_like` (List:String): A list of words with similar pronunciation. Each word in this list is used as one alternative pronunciation for the additional word. Those don't have to be real words - only their pronunciation matters. This list must not be empty. When `sounds_like` is used, the pronunciation automatically obtained from the `content` field is not used.

The `String` form `"word"` corresponds with the following `Object` form: `{"content": "word", "sounds_like": ["word"]}`.

Full example of `additional_vocab`:

```
"additional_vocab": [
    "speechmatics",
    {"content": "gnocchi", "sounds_like": ["nyohki", "nokey", "nochi"]},
    {"content": "CEO", "sounds_like": ["C.E.O."]},
    "financial crisis"
]
```

To clarify, the following ways of adding the word `speechmatics` are equivalent with all using the default pronunciation:

1. `"additional_vocab": ["speechmatics"]`
2. `"additional_vocab": [{"content": "speechmatics"}]`
3. `"additional_vocab": [{"content": "speechmatics", "sounds_like": ["speechmatics"]}]`

Output locale

Change the spellings of the transcription according to the output locale language code. If the selected language pack supports a different output locale, this config value can be used to provide spelling for the transcription in one of these locales. For example, the English language pack currently supports locales: `en-GB`, `en-US` and `en-AU`. The default value for `output_locale` is an empty string that means the transcription will use its default configuration (without spellings being altered in the transcription).

Punctuation overrides

This object contains settings for configuring punctuation in the transcription output.

- `permitted_marks` (List:String) Optional. The punctuation marks which the client is prepared to accept in transcription output, or the special value 'all' (the default). Unsupported marks are ignored. This value is used to guide the transcription process.
- `sensitivity` (Number) Optional. Ranges between zero and one. Higher values will produce more punctuation. The default is 0.5.

Error messages

Error messages have the following fields:

- `message: "Error"`
- `code` (Int): Optional. A numerical code for the error. See below. TODO: This is not yet finalised.
- `type` (String): Required. A code for the error message. See the list of possible errors below.
- `reason` (String): Required. A human-readable reason for the error message.

Error types

- **`type: "invalid_message"`**
  - The message received was not understood.
- **`type: "invalid_model"`**
  - Unable to use the model for the recognition. This can happen if the language is not supported at all, or is not available for the user.
- **`type: "invalid_config"`**
  - The config received contains some wrong/unsupported fields.

- **`type: "invalid_audio_type"`**
  - Audio type is not supported, is deprecated, or the audio_type is malformed.
- **`type: "invalid_output_format"`**
  - Output format is not supported, is deprecated, or the output_format is malformed.
- **`type: "not_authorised"`**
  - User was not recognised, or the API key provided is not valid.
- **`type: "insufficient_funds"`**
  - User doesn't have enough credits or any other reason preventing the user to be charged for the job properly.
- **`type: "not_allowed"`**
  - User is not allowed to use this message (is not allowed to perform the action the message would invoke).
- **`type: "job_error"`**
  - Unable to do any work on this job, the **Worker** might have timed out etc.
- **`type: "data_error"`**
  - Unable to accept the data specified - usually because there is too much data being sent at once
- **`type: "buffer_error"`**
  - Unable to fit the data in a corresponding buffer. This can happen for clients sending the input data faster then real-time.
- **`type: "protocol_error"`**
  - Message received was syntactically correct, but could not be accepted due to protocol limitations. This is usually caused by messages sent in the wrong order.
- **`type: "unknown_error"`**
  - An error that did not fit any of the types above.

Note that `invalid_message`, `protocol_error` and `unknown_error` can be triggered as a response to any type of messages.

The transcription is terminated and the connection is closed after any error.

Warning messages

Warning messages have the following fields:

- `message: "Warning"`
- `code` (Int): Optional. A numerical code for the warning. See below. TODO: This is not yet finalised.
- `type` (String): Required. A code for the warning message. See the list of possible warnings below.
- `reason` (String): Required. A human-readable reason for the warning message.

Warning types

- **`type: "duration_limit_exceeded"`**
  - The maximum allowed duration of a single utterance to process has been exceeded. Any AddAudio messages received that exceed this limit are confirmed with AudioAdded, but are ignored by the transcription engine. Exceeding the limit triggers the same mechanism as receiving an `EndOfStream` message, so the Server will eventually send an `EndOfTranscript` message and suspend.
  - It has the following extra field:
    - `duration_limit` (Number): The limit that was exceeded (in seconds).

Info messages

Info messages denote additional information sent form the **Server** to the **Client**. Those are similar to `Error` and `Warning` messages in syntax, but don't actually denote any problem. The **Client** can safely ignore these messages or use them for additional client-side logging.

- `message: "Info"`
- `code` (Int): Optional. A numerical code for the informational message. See below. TODO: This is not yet finalised.
- `type` (String): Required. A code for the info message. See the list of possible info messages below.
- `reason` (String): Required. A human-readable reason for the informational message.

Info message types

- **`type: "recognition_quality"`**
  - Informs the client what particular quality-based model is used to handle the recognition.
  - It has the following extra field:
    - `quality` (String): Quality-based model name. It is one of `"telephony"`, `"broadcast"`. The model is selected automatically, for high-quality audio (12kHz+) the broadcast model is used, for lower quality audio the telephony model is used.
- ** `type: "model_redirect"`
  - Informs the client that a deprecated language code has been specified, and will be handled with a different model. For example, if the `model` parameter is set to one of en-US, en-GB, or en-AU, then the request may be internally redirected to the Global English model (en).
- ** `type: "deprecated"`
  - Informs about using a feature that is going to be removed in a future release.

Example communication

The communication consists of 3 stages - initialization, transcription and a disconnect handshake.

On **initialization**, the `StartRecognition` message is sent from the Client to the API and the Client must block and wait until it receives a `RecognitionStarted` message.

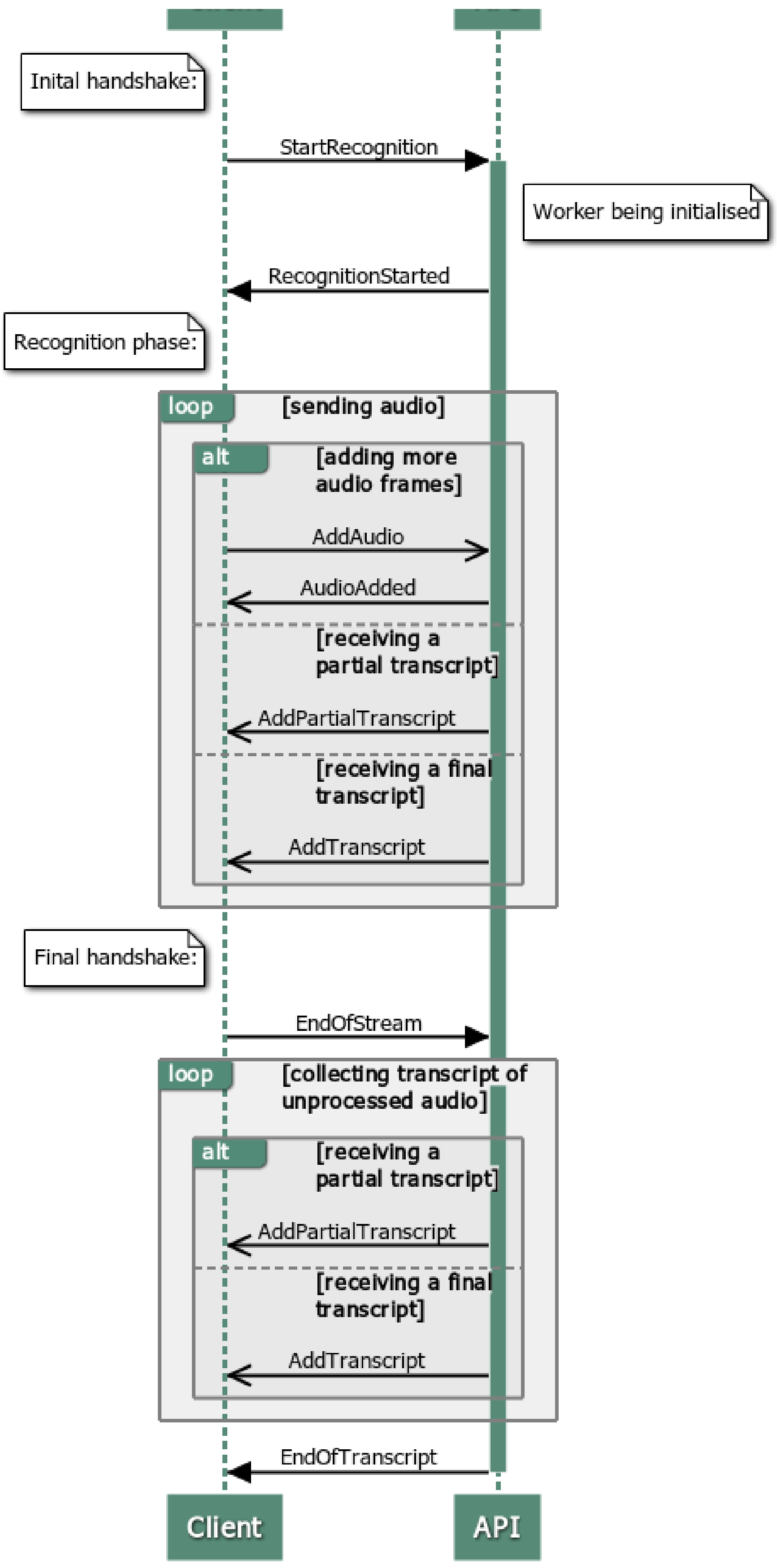
Afterwards, the **transcription** stage happens. The client keeps sending `AddAudio` messages. The API asynchronously replies with `AudioAdded` messages. The API also asynchronously sends `AddPartialTranscript` and `AddTranscript` messages.

Once the client doesn't want to send any more audio, the **disconnect handshake** is performed. The Client sends an `EndOfStream` message as it's last message. No more messages are handled by the API afterwards. The API processes whatever audio it has buffered at that point and sends all the `AddTranscript` and `AddPartialTranscript` messages accordingly. Once the API processes all the buffered audio, it sends an `EndOfTranscript` message and the Client can then safely disconnect.

Note: In the example below, `->` denotes a message sent by the Client to the API, `<-` denotes a message send by the API to the Client. Any comments are denoted `"[like this]"`.

```
-> {"message": "StartRecognition", "audio_format": {"type": "file"},
    "transcription_config": {"language": "en", "enable_partials": true}}

 <- {"message": "RecognitionStarted", "id": "807670e9-14af-4fa2-9e8f-5d525c22156e"}

->  "[binary message - AddAudio 1]"
->  "[binary message - AddAudio 2]"

 <- {"message": "AudioAdded", "seq_no": 1}
 <- {"message": "Info", "type": "recognition_quality", "quality": "broadcast", "reason": "Running
recognition using a broadcast model quality."}
 <- {"message": "AudioAdded", "seq_no": 2}
```

```
->  "[binary message - AddAudio 3]"

 <- {"message": "AudioAdded", "seq_no": 3}

"[asynchronously received transcripts:]"

 <- {"message": "AddPartialTranscript", "metadata": {"start_time": 0.0, "end_time":
0.5399999618530273, "transcript": "One"},
     "results": [{"alternatives": [{"confidence": 0.0, "content": "One"}],
                  "start_time": 0.47999998927116394, "end_time": 0.5399999618530273, "type":
"word"}
                ]}
 <- {"message": "AddPartialTranscript", "metadata": {"start_time": 0.0, "end_time":
0.7498992613545260, "transcript": "One to"},
     "results": [{"alternatives": [{"confidence": 0.0, "content": "One"}],
                  "start_time": 0.47999998927116394, "end_time": 0.5399999618530273, "type":
"word"},
                 {"alternatives": [{"confidence": 0.0, "content": "to"}],
                  "start_time": 0.6091238623430891, "end_time": 0.7498992613545260, "type":
"word"}
                ]}
 <- {"message": "AddPartialTranscript", "metadata": {"start_time": 0.0, "end_time":
0.9488123643240011, "transcript": "One to three"},
     "results": [{"alternatives": [{"confidence": 0.0, "content": "One"}],
                  "start_time": 0.47999998927116394, "end_time": 0.5399999618530273, "type":
"word"},
                 {"alternatives": [{"confidence": 0.0, "content": "to"}],
                  "start_time": 0.6091238623430891, "end_time": 0.7498992613545260, "type":
"word"}
                 {"alternatives": [{"confidence": 0.0, "content": "three"}],
                  "start_time": 0.8022338627780892, "end_time": 0.9488123643240011, "type":
"word"}
                ]}
 <- {"message": "AddTranscript", "metadata": {"start_time": 0.0, "end_time": 0.9488123643240011,
"transcript": "One two three."},
     "results": [{"alternatives": [{"confidence": 1.0, "content": "One"}],
                  "start_time": 0.47999998927116394, "end_time": 0.5399999618530273, "type":
"word"},
                 {"alternatives": [{"confidence": 1.0, "content": "to"}],
                  "start_time": 0.6091238623430891, "end_time": 0.7498992613545260, "type":
"word"}
                 {"alternatives": [{"confidence": 0.96, "content": "three"}],
                  "start_time": 0.8022338627780892, "end_time": 0.9488123643240011, "type":
"word"}
                 {"alternatives": [{"confidence": 1.0, "content": "."}],
                  "start_time": 0.9488123643240011, "end_time": 0.9488123643240011, "type":
"punctuation", "is_eos": true}
                ]}

"[closing handshake]"

->  {"message":"EndOfStream","last_seq_no":3}

 <- {"message": "EndOfTranscript"}
```


Inital handshake:
StartRecognition
Worker being initialised
RecognitionStarted
Recognition phase:
loop
[sending audio]
alt
[adding more audio frames]
AddAudio
AudioAdded
[receiving a partial transcript]
AddPartialTranscript
[receiving a final transcript]
AddTranscript
Final handshake:
EndOfStream
loop
[collecting transcript of unprocessed audio]
alt
[receiving a partial transcript]
AddPartialTranscript
[receiving a final transcript]
AddTranscript
EndOfTranscript
Client
API

Examples how to use the V2 API

The V2 Websocket Speech API aligns with other Speechmatics platforms such as the Batch Virtual Appliane and Speechmatics SaaS.

Websocket URI

To use the V2 API you use the '/v2' endpoint for the URI, for example:

```
wss://rt-asr.example.com:9000/v2
```

Session Configuration

The V2 API is configured by sending a `StartRecognition` message initially when the WebSocket connection begins. We have designed the format of this message to be very similar to the `config.json` object that has been used for a while now with the Speechmatics batch mode platforms (Batch Virtual Appliance, Batch Container and SaaS). The `transcription_config` section of the message should be almost identical between the two modes. There are some minor differences (for example batch features a different set of diarization options, and real-time features some settings which don't apply to batch such as `max_delay`).

TranscriptionConfig

A `transcription_config` structure is used to specify various configuration values for the recognition engine when the `StartRecognition` message is sent to the server. All values apart from `language` are optional. Here's an example of calling the `StartRecognition` message with this structure:

```
{
        "message": "StartRecognition",
        "transcription_config": {
            "language": "en"
        },
        "audio_format": {
            "type": "raw",
            "encoding": "pcm_f32le",
            "sample_rate": 16000
        }
    }
}
```

AddAudio

Once the websocket session is setup and you've sucessfully called `StartRecognition` you'll receive a `RecognitionStarted` message from server. You can then just to send the binary audio chunks, which we refer to as `AddAudio` messages.

You would replace this in the V2 API with much simpler code:

```
// NEW V2 EXAMPLE
function addAudio(audioData) {
    ws.send(audioData);
   seqNoIn++;
}
```

We recommend that you do not send more than 10 seconds of audio data or 500 individual AddAudio messages ahead of time.

Final and Partial Transcripts

The `AddTranscript` and `AddPartialTranscript` messages from the server output a JSON format which aligns with the JSON output format used by other Speechmatics products. There is a now a `results` list which contains the transcribed words and punctuation marks along with timings and confidence scores. Here's an example of a final transcript output:

```
{
   "message":"AddTranscript",
   "results":[
      {
         "start_time":0.11670026928186417,
         "end_time":0.4049381613731384,
         "alternatives":[
            {
               "content":"gale",
               "confidence":0.7034434080123901
            }
         ],
         "type":"word"
      },
      {
         "start_time":0.410246878862381,
         "end_time":0.6299981474876404,
         "alternatives":[
            {
               "content":"eight",
               "confidence":0.670033872127533
            }
         ],
         "type":"word"
      },
      {
         "start_time":0.6599999666213989,
         "end_time":1.0799999237060547,
         "alternatives":[
            {
               "content":"becoming",
               "confidence":1.0
            }
         ],
         "type":"word"
      },
      {
         "start_time":1.0799999237060547,
         "end_time":1.6154180765151978,
         "alternatives":[
            {
               "content":"cyclonic",
               "confidence":1.0
            }
         ],
         "type":"word"
      },
      {
         "start_time":1.6154180765151978,
         "is_eos":true,
         "end_time":1.6154180765151978,
         "alternatives":[
            {
               "content":".",
               "confidence":1.0
            }
         ],
         "type":"punctuation"
      }
   ],
   "metadata":{
      "transcript":"gale eight becoming cyclonic.",
      "start_time":190.65994262695312,
      "end_time":194.46994256973267
   },
   "format":"2.5"
}
```

You can use the `metadata.transcript` property to get the complete final transcript as a chunk of plain text. The `format` property describes the exact version of the transcription output format, which is currently 2.5. This may change in future releases if the output format is updated.

Advanced Punctuation

Some language models (English, French, German and Spanish currently) support advanced punctuation. This uses machine learning techniques to add in more naturalistic punctuation, improving the readability of your transcripts. As well as putting punctuation marks in more naturalistic positions in the output, additional punctuation marks such as commas (,) exclamation marks (!) and question marks (?) will also appear.

There is no need to explicitly enable this in the 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` setting within the `transcription_config` object, for example:

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

Note that changing the punctuation setting from the default can take a couple of seconds, which means if the user is using non-default neural punctuation sensitivity, after they send the `StartRecognition` message, there will be a slight delay (2-3 seconds) before the `RecognitionStarted` message is sent back.

The 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.

Example Usage

This section provides some client code samples that show simple usage of the V2 WebSockets Speech API. It shows how you can test your Real-Time Appliance using a minimal WebSocket client.

JavaScript

The basic usage of the WebSockets interface from a JavaScript client is shown in this section. In the first instance you setup the connection to the server and define the various event handlers that are required:

```
var ws = new WebSocket('wss://rta:9000/v2');
ws.binaryType = "arraybuffer";
ws.onopen = function(event) { onOpen(event) };
ws.onmessage = function(event) { onMessage(event) };
ws.onclose = function(event) { onClose(event) };
ws.onerror = function(event) { onError(event) };
```

In the above example, the hostname of the Real-Time Appliance or Container is **rta** – change this to match the IP address or hostname of your Real-Time Appliance or Container. The port used is 9000 and you need to make sure that you add '/v2' to the WebSocket URI. Note that only the secure WebSocket (wss) protocol is supported. You should also ensure that the binaryType property of the WebSocket object is set to `"arraybuffer"`.

In the `onopen` handler you initiate the session by sending the **StartRecognition** message to the server, for example:

```
function onOpen(evt) {
    var msg = {
        "message": "StartRecognition",
        "transcription_config": {
            "language": "en",
            "output_locale": "en-GB"
        },
        "audio_format": {
            "type": "raw",
            "encoding": "pcm_s16le",
            "sample_rate": 16000
        }
    };

    ws.send(JSON.stringify(msg));
}
```

An `onmessage` handler is where you will respond to the *server-initiated messages* sent by the appliance, and decide how to handle them. Typically, this involves implementing functions to display or process data that you get back from the appliance.

```
function onMessage(evt) {
    var objMsg = JSON.parse(evt.data);

    switch (objMsg.message) {
        case "RecognitionStarted":
            recognitionStarted(objMsg);
            break;

        case "AudioAdded":
            audioAdded(objMsg);
            break;

        case "AddPartialTranscript":
        case "AddTranscript":
            transcriptOutput(objMsg);
            break;

        case "EndOfTranscript":
            endTranscript();
            break;

        case "Info":
        case "Warning":
```

```
        case "Error":
            showMessage(objMsg);
            break;

        default:
            console.log("UNKNOWN MESSAGE: " + objMsg.message);
    }
}
```

Once the WebSocket is initialized, the **StartRecognition** message is sent to the appliance to setup the audio input. It is then a matter of sending audio data periodically using the **AddAudio** message.

Your **AddAudio** message will take audio from a source (for example microphone input, or an audio stream) and pass it to the Real-Time Appliance.

```
// Send audio data to the API using the AddData message.
function addAudio(audioData) {
   ws.send(audioData);
   seqNoIn++;
}
```

In this example we use a counter `seqNoIn` to keep track of the addAudio messages we've sent.

A set of server-initiated transcript messages are triggered to indicate the availability of transcripted text:

- `AddTranscript`
- `AddPartialTranscript`

See above for changes to the JSON output schema in the V2 API. For full details of the output schema refer to the AddTranscript section in the API reference.

Finally, the client should send an **EndOfStream** message and close the WebSocket when it terminates. This should be done in order to release resources on the appliance and allow other clients to connect and use resources.

The Mozilla developer network provides a useful reference to the WebSocket API.

Python

Speechmatics provides a Python library called `smwebsocket-py` which is a wrapper to the WebSockets API, making it easy to incorporate Speechmatics real-time transcription into your Python program. Please contact support@speechmatics.com if you require this library.

The `smwebsocket-py` library can be incorporated into your own applications, used as a reference for your own client library, or called directly from the command line (CLI) like this (to pass a test audio file to the appliance):

```
python -m smwebsocket.cli --url wss://rta:9000/v2 --max-delay 3 --lang en test.mp3
```

Note that configuration options are specified on the command-line as parameters, with a '_' character in the configuration option being replaced by a '-'. The CLI option accepts an audio stream on standard input, meaning that you can stream in a live microphone feed. To get help on the CLI use the following command:

```
python -m smwebsocket.cli --help
```

The library depends on Python 3.7 or above, since it makes use of some of the newer `asyncio` features introduced with Python 3.7.

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). Using 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. 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`, `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:

```
{
  "message": "StartRecognition",
  "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 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 converted to numbers separated by "/"	three sixteenths	3/16
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

```
[{
 "message": "AddTranscript",
 "format": 2.7,
 "results": [{
     "alternatives": [{
         "confidence": 1,
         "content": "17th of January 2022",
         "language": "en"
     }],
     "end_time": 3.0899999141693115,
     "entity_class": "date",
     "spoken_form": [{
             "alternatives": [{
                 "confidence": 1,
                 "content": "Seventeenth",
                 "language": "en"
             }],
             "end_time": 1.3799999952316284,
             "start_time": 0.8399999737739563,
             "type": "word"
         },
         {
             "alternatives": [{
                 "confidence": 1,
                 "content": "of",
                 "language": "en"
             }],
             "end_time": 1.4399999380111694,
             "start_time": 1.3799999952316284,
             "type": "word"
         },
         {
             "alternatives": [{
                 "confidence": 1,
                 "content": "January",
                 "language": "en"
```

```
                }],
                "end_time": 1.9199999570846558,
                "start_time": 1.4399999380111694,
                "type": "word"
            },
            {
                "alternatives": [{
                    "confidence": 1,
                    "content": "twenty",
                    "language": "en"
                }],
                "end_time": 2.25,
                "start_time": 1.9199999570846558,
                "type": "word"
            },
            {
                "alternatives": [{
                    "confidence": 1,
                    "content": "twenty",
                    "language": "en"
                }],
                "end_time": 2.549999952316284,
                "start_time": 2.25,
                "type": "word"
            },
            {
                "alternatives": [{
                    "confidence": 0.9504331946372986,
                    "content": "two",
                    "language": "en"
                }],
                "end_time": 3.0899999141693115,
                "start_time": 2.549999952316284,
                "type": "word"
            }
        ],
        "start_time": 0.8399999737739563,
        "type": "entity",
        "written_form": [{
                "alternatives": [{
                    "confidence": 1,
                    "content": "17th",
                    "language": "en"
                }],
                "end_time": 1.1999999682108562,
                "start_time": 0.8399999737739563,
                "type": "word"
            },
            {
                "alternatives": [{
                    "confidence": 1,
                    "content": "of",
                    "language": "en"
                }],
                "end_time": 1.559999962647756,
                "start_time": 1.1999999682108562,
                "type": "word"
            },
            {
                "alternatives": [{
                    "confidence": 1,
                    "content": "January",
                    "language": "en"
                }],
                "end_time": 1.9199999570846558,
                "start_time": 1.559999962647756,
                "type": "word"
            },
            {
                "alternatives": [{
                    "confidence": 1,
                    "content": "2022",
                    "language": "en"
                }],
                "end_time": 3.0899999141693115,
                "start_time": 1.9199999570846558,
                "type": "word"
            }
        ]
    }],
    "metadata": {
        "end_time": 5.16,
        "start_time": 0,
        "transcript": "17th of January 2022 "
    }
}]
```

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

```
  [{
    "message": "AddTranscript",
    "format": 2.7,
    "results": [{
            "alternatives": [{
                "confidence": 1,
                "content": "17th",
                "language": "en"
            }],
            "end_time": 1.1999999682108562,
            "start_time": 0.8399999737739563,
            "type": "word"
        },
        {
            "alternatives": [{
                "confidence": 1,
                "content": "of",
                "language": "en"
            }],
            "end_time": 1.559999962647756,
            "start_time": 1.1999999682108562,
            "type": "word"
        },
        {
            "alternatives": [{
                "confidence": 1,
                "content": "January",
                "language": "en"
            }],
            "end_time": 1.9199999570846558,
            "start_time": 1.559999962647756,
            "type": "word"
        },
        {
            "alternatives": [{
                "confidence": 1,
                "content": "2022",
                "language": "en"
            }],
            "end_time": 3.0899999141693115,
            "start_time": 1.9199999570846558,
            "type": "word"
        }
    ],
    "metadata": {
        "end_time": 5.16,
        "start_time": 0,
        "transcript": "17th of January 2022 "
    }
}]
```