Server based Recording

WebRTC Virtual Classroom PLatform

Table of Contents

[Recording Configuration options 2](#_Toc458757400)

[No recording 2](#_Toc458757401)

[Recording from start to end 2](#_Toc458757402)

[Recording starts automatically but 2](#_Toc458757403)

[Recording does not start automatically 2](#_Toc458757404)

[Flows 2](#_Toc458757405)

[Configuration 2](#_Toc458757406)

[Provisioning 2](#_Toc458757407)

[Actual Recording 3](#_Toc458757408)

[In-session operations 3](#_Toc458757409)

[Closure 3](#_Toc458757410)

[Authentication 3](#_Toc458757411)

[RecBot Design 4](#_Toc458757412)

[RecBot VM Instantiation 4](#_Toc458757413)

[Protocol Cluster-Recbot 4](#_Toc458757414)

[Discovery 4](#_Toc458757415)

[Start Recording 5](#_Toc458757416)

[Stop/Pause Recording 5](#_Toc458757417)

[Consolidate (or Save) Recording 5](#_Toc458757418)

[Recording Consolidation 6](#_Toc458757419)

[Keep the Session Cluster Alive 6](#_Toc458757420)

[RecBot directly informs the upstream entity 6](#_Toc458757421)

[Transcoding Workflow 7](#_Toc458757422)

[Self Destruct 7](#_Toc458757423)

[Cost Calculations 7](#_Toc458757424)

[Appendix A – installations 7](#_Toc458757425)

[Appendix B – Commands 8](#_Toc458757426)

# Recording Configuration options

The recording operation in a session can be of the following types:

### No recording

No recording controls should be visible to the presenter at all.

### Recording from start to end

The recording must happen from start to end of the class. The presenter has no control over it. The ‘start’ maybe defined by the entry of the presenter or the actually beginning of the session.

### Recording starts automatically but

… the presenter has controls to pause and resume

### Recording does not start automatically

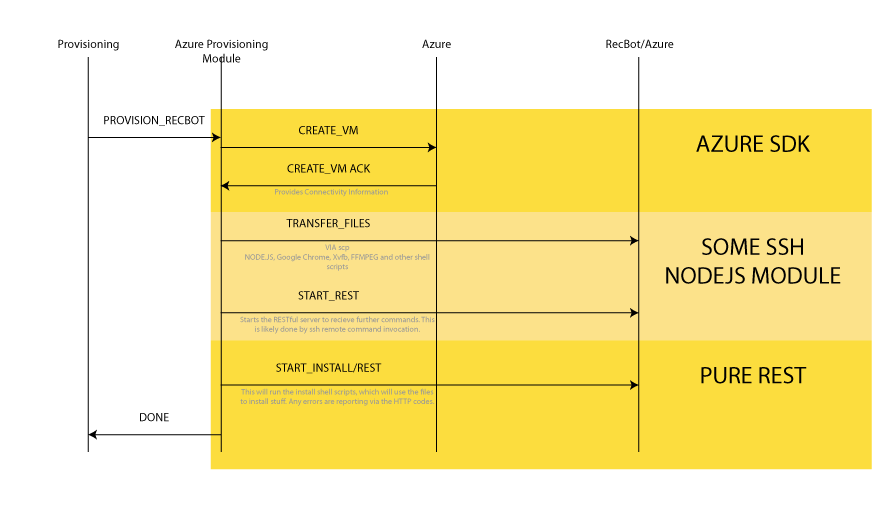
The presenter decides the best time to start the recording.

# Flows

### **Configuration**

The need for recording is indicated in the class configuration as one of the resources.

### **Provisioning**



Ideally, the recording bot (**RecBot**) should be instantiated upon the creation of the class. However, the provisioning of the VM could take time. One possibility is to have an array of RecBots pre-created and waiting. One way or the other, the following should happen:

* A VM should be provisioned for the session to act as a RecBot*.*
* The class meta-data should be provided to the RecBot, so it can join the correct class at the correct time.
  + Additionally, it needs to inform ‘someone’ once the recording has been uploaded to its final destination. This URL should also be supplied as a part of the meta-data.
* The IP Address of the RecBot should be added to the session info (which is passed on to the node to create a class)
* The RecBot waits for the class to start and then joins, just like any normal attendee would

The RecBotfor now will be provisioned on the Azure cloud. However, the code must be service provider independent.

TODO: What happens if the RecBot provisioning fails?

### **Actual Recording**

The actual act of recording is triggered by whoever has the recording permissions (typically the presenter). The recording START command is sent (by the act of pressing a record button in the UI) over the session control channel (WebSocket).

This message must reach the RecBot, which will also be connected to the session cluster via a control channel.

However, the RecBotwill have an additional **Recording Server Module** (which the other attendees will not have) to handle the incoming START recording messages and other messages of the Recording Protocol. The RecBot will attempt to start the recording of the session and will communicate its success of failure back to the originator of the request.

The recording is now in progress.

### **In-session operations**

The RecBotmust handle multiple sequential START/STOPS/PAUSE in a particular session.

### **Closure**

The RecBot remains active till the session is active. The process of consolidation of the video recording starts once the session has triggered its termination. The video file is uploaded, likely to an Azure Blob. The session cluster has, meanwhile, most likely terminated itself.

Once the upload is complete, the RecBot informs the concerned party by invoking the URL supplied via the meta-data and then terminates itself.

# Authentication

The RecBot needs to join the session just like all other participants, and therefore needs to authenticate itself to the session cluster. This authentication needs to be treated specially, since we might want to hide the presence of the RecBot in the attendee list, amongst other things.

One possible way is a direct authentication, where the credentials are encrypted in the URL query parameters as follows:

[ url of the session ] ? e=[ encrypted information ]

The encrypted information could contain the following parameters:

* Validity time stamp
* Some identifier (equivalent to a display name)
* Maybe, visible or invisible flag
* Additional modules to load (this could be outside of the encrypted set as well)

This could look like this:

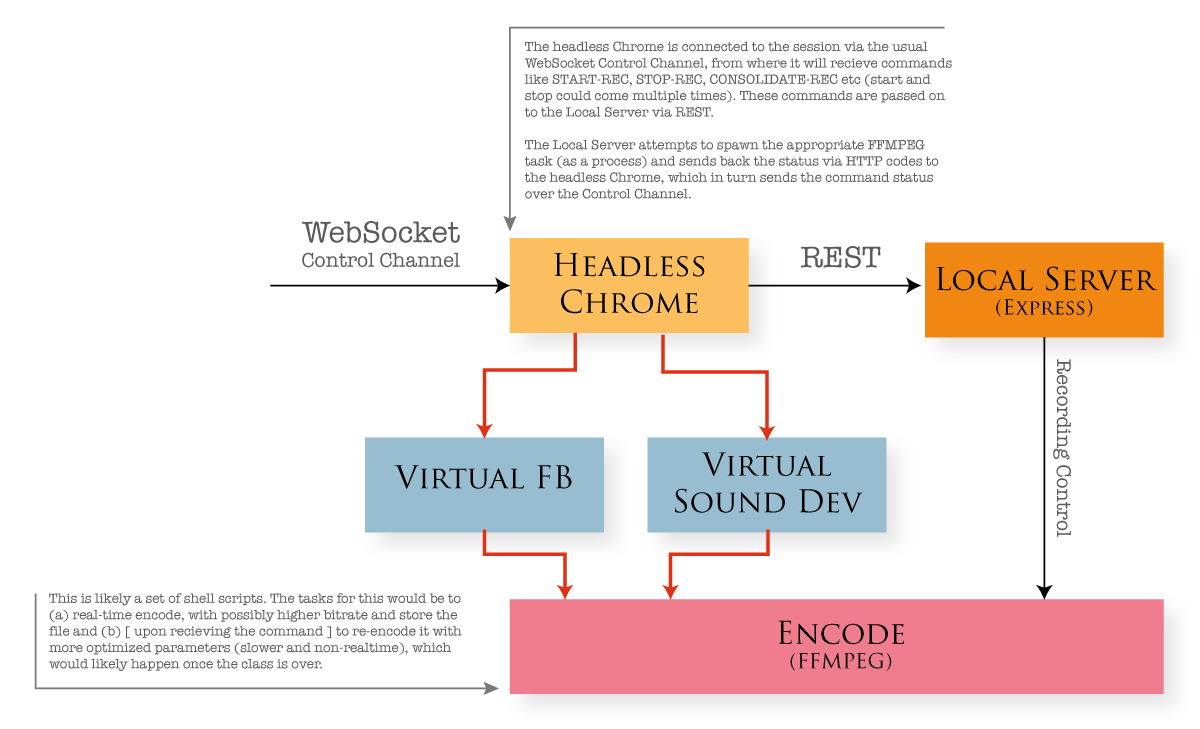
https://**[ landing host** ]/landing/session/v1/**[ class id ]**?e=**[ encrypted params ]**&auth\_via=private

/\* Decrypted URL would look like this:

\* name=recbot&D=YYMMddHHmm

\*/

# RecBot Design



# RecBot VM Instantiation

TODO

# Protocol Cluster-Recbot

### **Discovery**

The RecBot will take some time to come up and when it does it must announce its availability to the Session Cluster.

/\* Message type : ‘info’ \*/

{

info\_id : ‘REC\_IAMALIVE’,

info : { /\* Some information \*/

}

Upon receiving this message from the RecBot, the Session Cluster will inform all participants with Recording Permissions (defined elsewhere) that the recording facility is now available. This message be used by the client side code to enable the recording button.

### **Start Recording**

When the user presses the *Start Recording* button, it in turn sends a command to the Session Cluster, which in turn relays the command to the RecBot. The RecBot acks or nacks the command, which is relayed back to the user and an appropriate status is shown on the user UI.

/\* Message type : ‘req’ \*/

{

command : ‘REC\_START’,

data : { /\* Some data \*/

}

### **Stop/Pause Recording**

Similar to *Start.* This could be interpreted as either *Pause* or *Stop*. A *Start* could be received again after this command (hence it would be interpreted as *Pause*). If a REC\_CONSOLIDATE command is received next, then it would be interpreted as a *Stop*.

/\* Message type : ‘req’ \*/

{

command : ‘REC\_STOP’,

data : { /\* Some data \*/

}

### **Consolidate (or Save) Recording**

The *Consolidate* command should accomplish the following:

1. Re-encode the recording for better compression
2. Transfer it to the configured persistent storage (Azure Blob or Amazon S3)
3. Inform *someone* that the stuff is done (see **Recording Consolidation**)

/\* Message type : ‘req’ \*/

{

command : ‘REC\_CONSOLIDATE,

data : { /\* Some data \*/

}

# **Recording Consolidation**

The consolidation of recording can take significant time. This section discusses some ideas around that. The problem statement is: once a class is over, the presenter expects to know the status of the recording and how she can access it.

However, since the consolidation could literally take hours, we can’t expect her to keep sitting with her session open, waiting for the intimation.

There are possibly multiple ways to handle this.

## Keep the Session Cluster Alive

One way is to keep the Session Cluster alive till it receives the confirmation from the RecBot. It may receive additional information regarding the URL of the recording etc. The presenter does not need to be logged on all the while, as the Session Cluster will do this in its DEPROVISIONING state.

The Session Cluster can then inform some upstream entity (likely a WizIQ endpoint) using a REST API. The final intimation to the presenter(s) is left as an exercise to the upstream entity ☺.

But this is resource consuming, since the Session Cluster is alive and doing almost nothing.

## RecBot directly informs the upstream entity

The Session Cluster merges into the benign and indifferent Operating System, but informs the RecBot, possibly via the REC\_CONSOLIDATE command of the upstream URL. The RecBot does the intimation rather than the Session Cluster.

# Transcoding Workflow

A session can range in duration from ½ hour to 2 or even 3 hours. Potentially, a recording could be massive in size. A very ballpark calculation is as follows:

# Self Destruct

The RecBot must be configured to, and be able to self-destruct. The class duration will be known to the RecBot. However, it will require additional time post the class getting finished to process the recording. The self-destruct feature is a safe guard against the RecBot getting stuck in the post processing indefinitely.

The RecBot will self-destruct automatically once it’s job is done (and inform the relevant services). However, there should be a hardcoded upper limit on the lifetime of the RecBot – possibly something like 5 hours, upon which the RecBot will *unconditionally* self-destruct.

# Cost Calculations

The recording will use up VMs runtime and therefore incur costs. A module which keeps and estimated cost is must be implemented, likely at the Provisioning server end. It should be possible to get a current estimate of costs:

1. Per Session
2. Overall Cost on a daily or monthly basis

# Appendix A – installations

Installing Virtual Frame Buffer

sudo apt-get install Xvfb

Installing Chrome

wget -q -O - https://dl-ssl.google.com/linux/linux\_signing\_key.pub | sudo apt-key add -

sudo sh -c 'echo "deb [arch=amd64] http://dl.google.com/linux/chrome/deb/ stable main" >> /etc/apt/sources.list.d/google-chrome.list'

sudo apt-get update

sudo apt-get install google-chrome-stable

Installing FFMPEG

sudo add-apt-repository ppa:mc3man/trusty-media  
sudo apt-get update  
sudo apt-get dist-upgrade

sudo apt-get install ffmpeg

# Appendix B – Commands

To start the virtual frame buffer

Xvfb :99 -screen 0 1920x1080x24

To start Chrome

export DISPLAY=:99

google-chrome --window-position=0,0 --window-size=1920,1080 –kiosk <site-addr>

The exporting of DISPLAY ensures that the Chrome will use the virtual frame buffer.

To capture video via FFMPEG

ffmpeg -video\_size 1920x1080 -framerate 29 -f x11grab -i :99.0 -c:v libx264 -qp 0 -preset ultrafast ~/capture.mkv

# APPENDIX C – SOUND SETUP

To check and restart pulseaudio daemon

pulseaudio –check

# a zero return value implies the daemon is running

pulseaudio –k

# to see the instances

pulseaudio –D

# restart the daemon

Setup audio recording

#

# Create a sink

#

pactl load-module module-null-sink sink\_name=MySink

#

# Export it for Chrome to use

#

export PULSE\_SINK=MySink

# Now start Google Chrome

#

# To record pure audio into an mp3, run the following

#

ffmpeg -loglevel verbose -f pulse -ac 2 -i MySink.monitor -acodec libmp3lame -ar 44100 -f mp3 out.mp3