Recording sessions in Agora require a server-side call via their REST API to “acquire” resources for recording, start/stop recording, and query and ongoing recording. Agora will dump the media files into cloud storage (AWS S3 in our case) during the recording process, and also call a “webhook” (REST-based callback) to our server to notify it of events throughout the recording process.

I have some concern that the callback to our servers may not be completely reliable (they had missteps enabling it). There are also some complexities because recording will be stopped if the session is not joined within 5 minutes, and may need to be restarted if everybody drops off from the session. For these reasons, it might be necessary to create AWS Lambda functions to trigger on new S3 files and make callbacks to our server, as a backup to the Agora webhooks. Server-side functionality could then be used to detect stopped recordings (e.g. if media files stop appearing). Another approach could be to poll the recording status server-side using the Agora REST API.

There is no UI to see existing recording sessions, download or manage media, or indicate that recording is successfully in progress. In the short run, we can use off-the-shelf tools (e.g. CyberDuck) to view, download, and manage media files. We can tie into the existing Global Events websockets mechanism to send recording state information to the browser for recording status.

Recording can be done in a “composite” fashion; all of the streams are combined into a single A/V stream, or as individual streams from each participant. These are mutually exclusive: you can do it as a composite or as individual streams, but not both.

Since the solution is WebRTC based, the resolution of streams is controlled by the actual resolution of the video streams produced by the participants, which is determined by the real-time constraints of the session, so we will not have much control of the recorded resolution.

Also, the recording process gives us identifiers of recording resources, which in turn are used to identify files from the recording, so we will have to store those identifiers in our database session object to be able to locate the files by name.

A final issue: the Agora recording mechanism requires that participant’s user IDs are integers, whereas most identity systems do not use integers to identify them (we are currently using string-based login names to identify users to Agora). So, we will need a mechanism to assign integers to users (across sessions) so that we can record. We will also need that mapping to determine the user associated with a particular recorded stream, and to locate the files associated with a specific user.

### Resources

[Agora Cloud Recording Overview](https://docs.agora.io/en/cloud-recording/product_cloud_recording?platform=Linux)

[Agora Cloud Recording RESTful API](https://docs.agora.io/en/cloud-recording/cloud_recording_api_rest)

[Cloud Recording RESTful API](https://docs.agora.io/en/cloud-recording/restfulapi/#/Cloud%20Recording/query)

[Agora Cloud Recording Postman Collection](https://documenter.getpostman.com/view/6319646/SVSLr9AM?version=latest#917f58a8-d6ef-411b-94b6-39dd6a020e20)

[How to Serve HLS Video from an S3 Bucket](https://hlsbook.net/how-to-serve-hls-video-from-an-s3-bucket/)