**STREAM RECORD APP**

*(for pexip api)*

**ANALYSIS**

Based on the previous report, we can setup a nginx server to use as a re-stream proxy and record all the meeting stream (read NGINX AS STREAMING AND STORAGE for the simple setup detail).

On this nginx Server, we need to build a record app so we can adapt these main features:

* Receive the record command from the Main App (via API) and do the record for selected live stream.
* Manage the recorded streams
* List streams
* Return recorded videos info based on stream id so the Main App user can view/download or do other commands (delete, etc.)

Beside of those main features above, we still need to have a scalable plan for the app on those things below:

* Security: IP restriction, firewall, token
* Logging the record state
* Planning for the performance or limitation of each server node:
  + Hardware: CPUs (threads), Storage, Bandwidth
  + Software: Load Balancing, multi threads process, multi servers node, message queue,…

We are going to use Nodejs & related libraries to build the application. For the database, we will use MySQL.

**FIRST VERSION REQUIREMENTS & OTHER NOTES**

**Record API**

*Input*

* Stream source: Record API will need the live stream as the input. This information should be provided by the Main App as the API’s body. Pexip can provide the stream URI through the *participant* *calls* API with *type = RTMP* (<https://docs.pexip.com/api_client/api_rest.htm#calls>)
* Meeting uuid: this will be used as the ID of the recorded video and also used to build the video file name.

*Process*

* Build shell file: The uuid & stream source will be used to build a shell (.sh) file which contains the *ffmpeg* code to do the re-stream via the Live stream application which is configured on nginx RTMP.  
  *NOTE:* There is a small chance that in one meeting, the pexip will devide into multi streams based on the *participant uuid*. We still need to test to have the confirmation.
* Run shell file: The API will use *child\_process* from Nodejs to run the created .sh file in the background. With this, many .sh file can be run at the same time. The *worker\_thread* or *cluster* can be applied here so we can optimize the app. Need more research on using *pool* library for *worker\_thread*.
* Save meeting info: The API should save recorded stream information into the DB:
  + Meeting id (stream id)
  + Status
  + Video Path
  + Log path (or log content)
  + Started time / End time
  + Hash (optional)
* Stop command (mandatory) With current configuration, we only get all and record all. We still haven’t applied the *stop* command. This *Stop* command is mandatory so the record process will be stopped when:
  + Security issues
  + There is any error with the stream source
  + The meeting is stopped (stream source is stopped)
  + Out of hardware resources (CPU, memory, storage,...)
* Resume the meeting record (optional, later): The feature can be used when the record participant by any chance is out of the meeting and re-join. We will use the meeting id (uuid) so we can know the recorded videos belong to one meeting.