ISSN: 2831-6789

Research Article

Journal of Robotics and Automation Research

High-Performance Network Application for Creating Personalized Videos and Audios Using MLT

Marius Heinrich*

Heinrich Software Solutions Department of Computer Science North Rhine-Westphalia Germany

***Corresponding Author**
Marius Heinrich, Heinrich Software Solutions Department of Computer Science North Rhine-Westphalia Germany.

Submitted: 2023, Jun 13; **Accepted:** 2023, July 11; **Published:** 2023, July 20

Citation: Heinrich, M. (2023). High-Performance Network Application for Creating Personalized Videos and Audios Using MLT. *J Robot Auto Res, 4*(2), 384-386.

Abstract

Nowadays, many marketing campaigns are based on personalized marketing, so a solution should be found for the core problem, namely the creation of this content. More and more software solutions are based on proprietary products where the licenses for commercial use are unclear. A solution has now been found for this, which is made available to all people as open source technology. The concept is based on the Media Lovin' Toolkit, which is a framework written in C++ that uses XML files to create finished videos that have previously been designed in a conventional editor. In the XML file, however, the most diverse parameters can be changed before processing, which makes this framework, a perfect problem solver.

Keywords: Autoscaling, Rendering, Network

1. Media Lovin' Toolkit

The Media Lovin' Toolkit, a framework written in C++, uses XML files to create finished videos that have previously been designed in a conventional editor. In the XML file, however, the most diverse parameters can be changed before processing.

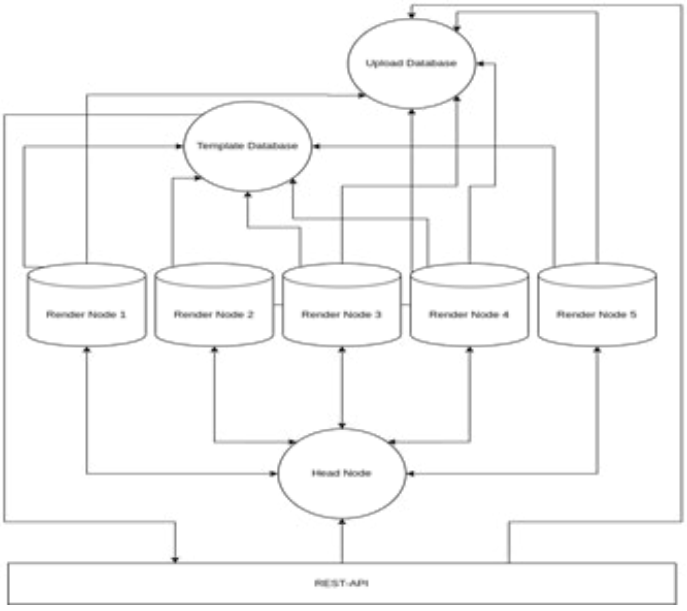
1.1. Possible modifications

- Text replacement.
- Image replacement.
- Audio replacement.
- Video replacement.
- Path replacement for dynamic environments.

2. Example Commands

- Project preview without render
- melt {project}.mlt
- Render project without settings (default settings are used) melt {project}.mlt -consumer avformat:{filename}.mp4
- Render project with settings (default settings are overwritten)
- melt {project}.mlt -consumer avformat:{filename}.mp4 f=mp4 movflags=+faststart vcodec=libx264 progressive=1 g=15 bf=2 crf=15 acodec=aac ab=256k

3. Network



3.1. Components

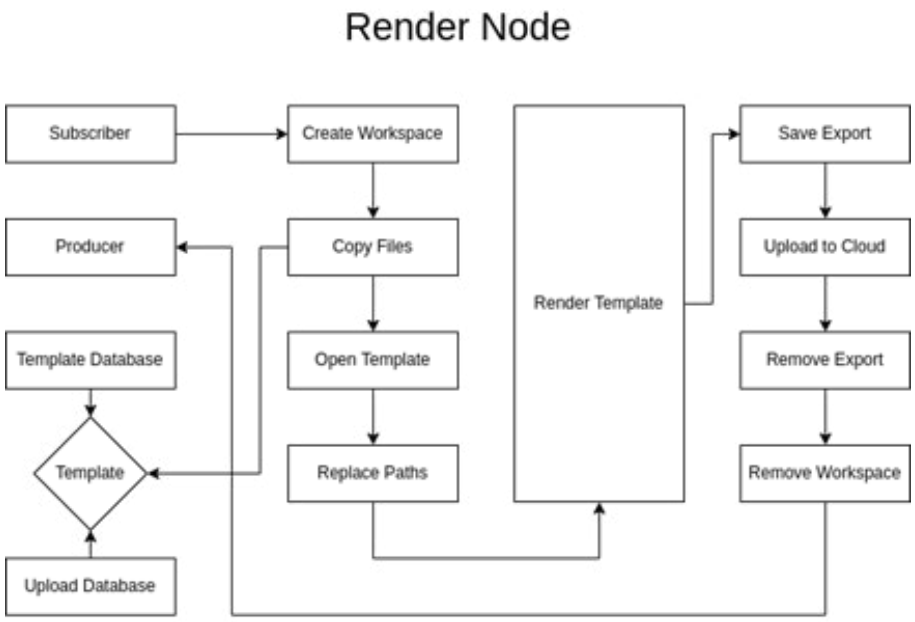
Our network consists of 4 components

1. Databases.
2. Rendering nodes.
3. Head node.
4. RESTful Web-Services.

3.2. Explanation of the Individual Components

1. The template database stores all the available projects with their associated files. (placeholders, audios, videos, images)
2. The upload database stores all files uploaded by the user, for later processing. (Audios, Videos, Images)
3. The render nodes are responsible for processing the final product. They take over all relevant core tasks.
4. The head node is responsible for job distribution, it throws the jobs into the queues and ensures load balancing among the individual nodes.
5. The API is responsible for receiving the individual jobs, and can of course also be relieved separately via a load balancer.

4. Render Node

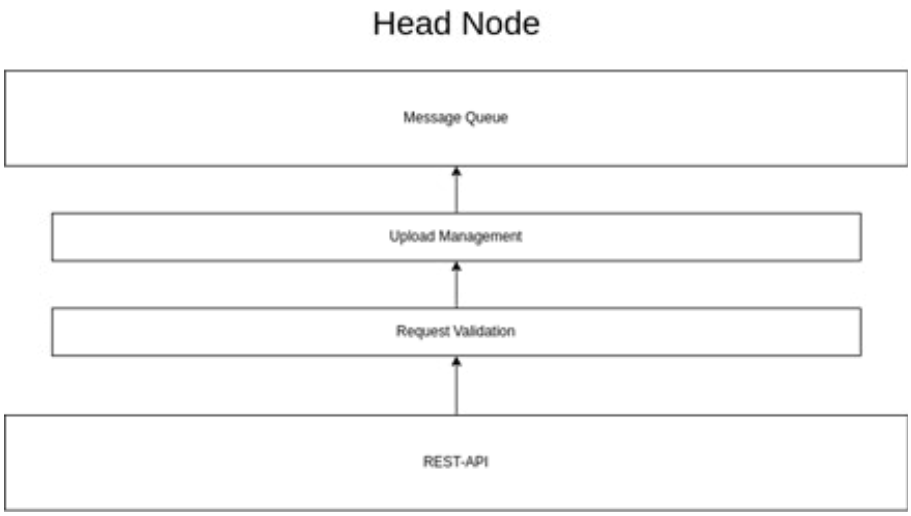


The render nodes are responsible for processing the final product. They perform all relevant core tasks, including processing the data from the receipt of the job to the upload to the cloud storage.

4.1. Processing Sequence

1. First we create a working environment, this can be a folder with a unique ID.
2. Next, we load all files into our working environment, including the project file.
3. Now we open the project file and replace the Static paths with the Dynamic paths, for this reason we also initially assigned a unique ID to reference to our working environment later.
4. When this step is done, all previously determined placeholders are replaced.
5. Now the project file can be rendered.
6. We will then get a file, in the desired output format. This file can now be saved in a previously determined temporary location.
7. This file will now be uploaded to the cloud storage.
8. Finally, we will now delete the render product, including the working environment.
9. Now we have our finished video in the cloud storage.

5. Head Node



The main node is responsible for the job distribution, it throws the jobs into the queues and takes care of the load balancing between the individual nodes. In addition, it takes care of the validation of the individual requests and ensures a correct upload of all data to be processed later. The main node takes care of the modularization of the individual render nodes and serves as a central hub. Thus, the render nodes can be expanded as needed to handle load peaks more easily and quickly. Message Brokers are a common tool employed to fixture communication in a distributed environment. This system provides with asynchronous communication between producers and consumers, and handles some of the challenges that are common within distributed and concurrent data processing [1].

5.1. High Load Peaks

To get the highest performance out of the system, please pay special attention to the following factors

1. Make sure you have enough system resources in both the head node and the render node.
2. Make sure they have enough cloud storage to avoid a mistake. In addition, you should pay attention to the latency of the provider.
3. Set up a redundant system and use a load balancer in front of the API.
4. Make sure that the uploaded files are validated properly and that there is no vulnerability in the system.

References

1. Landau, D., Andrade, X., & Barbosa, J. G. (2022). Kafka Consumer Group Autoscaler. arXiv preprint arXiv:2206.11170.

Copyright: *©2023 Marius Heinrichl. This is an open-access article distributed under the terms of the Creative Commons Attribution License, which permits unrestricted use, distribution, and reproduction in any medium, provided the original author and source are credited.*