<EC500 Case Study: PeerTube>

Byoungsul Lee

With the rise of peer to peer technology and its advantages over having a central server, a lot of softwares such as Bitcoin's transaction and Skype's (previous) VoIP successfully uses this technology to carry out tasks over the internet. PeerTube, which was recently published to the public in 2018 aims to build a "Decentralized Youtube" using peer to peer technology in its core and create a federated network of small servers.

PeerTube is written in a programming language that is a superset of Javascript, which is called Typescript. Being an decentralized video software application, compatibility over variety of environment is important to provide a seamless service across all user bases. Typescript offers access to ES6 and ES7 features which are supported to all major browsers and since it compiles down to a version of Javascript in compilation, it also becomes adaptable in all browsers. In terms of scalability, Typescript's "Interface oriented development" offloads the onboarding time of new developers to the project. This is especially crucial for an open source software that has multiples and a growing team of people contributing to the project. Another benefit is the ES-next compliance giving futureproof to the scaling software as Javascript develops further with new features. Due to these benefits of using Typescript as a core language for PeerTube and the fact that PeerTube itself is relatively new, I believe that Typescript would still be used even if it was to be rebuilt today.

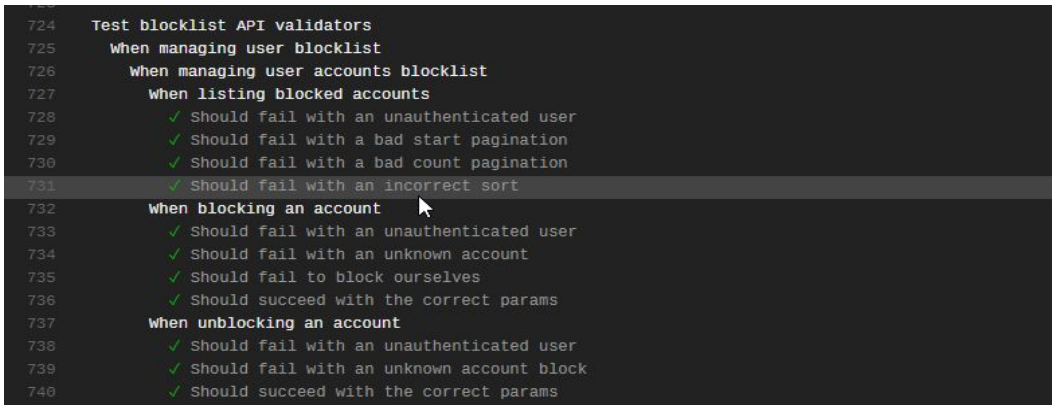
// Content on Build systems

The core libraries (Dependencies) for PeerTube consists of the list below :

1) nginx
2) PostgreSQL >= 9.6
3) Redis >= 2.8.18
4) NodeJS >= 8.x

Since the local PeerTube server is never to be exposed to the public web directly, PeerTube uses nginx as a reverse proxy to make up for its vulnerability. PostgreSQL is used to store long-term data such as user accounts, video metadata, comments or channels are stored in a PostgreSQL database. Unlike the long-term data, a caching system, Redis, is used to store temporary data to reduce load from the database and to increase loading speed. NodeJS serves as a core library hosting the small servers within each community local nodes.

As PeerTube is a scalable project with more than 100 contributors, it is using Travis CI as their continuous integration platform. Travis CI is ideal for this open-source project hosted in github due to the fact that it provides the developers free cloud-based hosting that requires no maintenance or administration on their part. When a contributor makes a commit to the PeerTube project Travis CI automatically checks if the update complies with all the core functionalities and edge cases that is crucial to the service and security.



The automated system checks from validating user listings, searching videos, subscribing to channels and countless more features. It measures the response time of certain time-sensitive features to make sure that they function within a reasonable timeframe. In addition to the time measure, the PeerTube's testing mechanics check for result values to their testing framework so that the value of the output matches the expected result and behavior of the ideal working system.

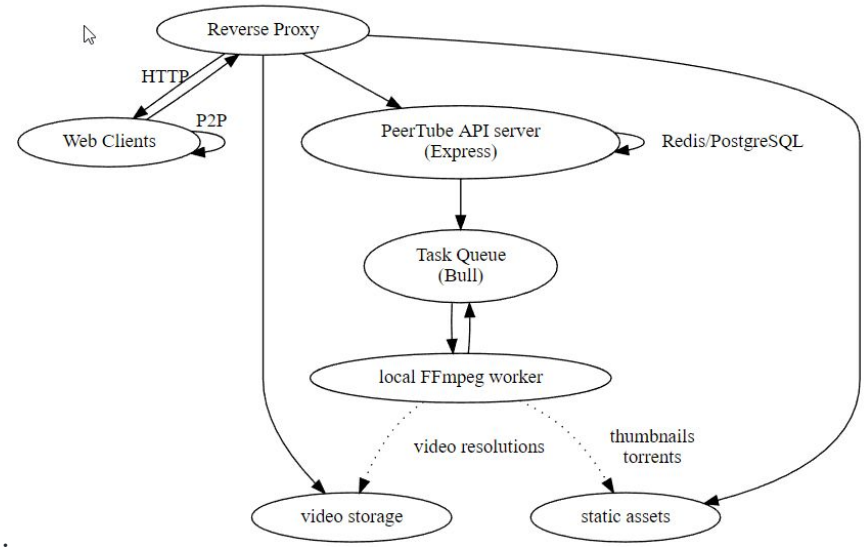
```
it('Should create a thread in this video', async function () {
  const text = 'my super first comment'

  const res = await addVideoCommentThread(server.url, server.accessToken, videoUUID, text)
  const comment = res.body.comment

  expect(comment.inReplyToCommentId).to.be.null
  expect(comment.text).equal('my super first comment')
```

For this example given above, the testing code invokes a function to add a video comment ("my super first comment") by hard coding the text value. Then it checks to to see if the value equals from the result given by the addVideoCommentThread function. The testing environment for this Travis CI is Linux, and Windows platform is not supported yet.

The architecture for PeerTube is broadly divided into 2 parts, client application (frontend) and the server (backend). The client application is made up with the web interface, which is written in Angular. This module takes care of all the user interaction and interacts with the server via HTTP protocol. Once the reverse proxy receives the request from the client, it proxies them to the PeerTube's API (Express). Once the request reaches the API server, it determines if the request is within the Redis/PostgreSQL and enqueues the information into the task queue. When the video is within the local system, the video gets passed back into the reverse proxy for the peer to peer streaming



Since PeerTube's API is designed to by modular, it is easy to add new functionalities if needed. If I were to add a functionality to the system, I would add more focus to the existing video buffering function. PeerTube's video streaming is greatly affected by the other instance of PeerTube since the video is often being loaded from other people's local system. The video streaming of one's system might lag behind even if it doesn't have any connectivity issues on its part. By adding a better or longer buffer to the video depending on the speed of the other instances. Adding to the code design of PeerTube, the design of the PeerTube's system leans towards the functional component design, mainly separated by each functionality of the video platform. If we look at the github code, we can observe that most of the components are named and coded by their functionality rather than passing an object and calling their class methods. However, PeerTube also has model objects that maps search queries and user account object for creation and deletion. The project is also usable in other programs by disabling the front end portion of the architecture and embedding the videos into them. Also some of the architecture is implemented asynchronously so that

it maximizes the streaming speed of the video. One part that I found that was asynchronous was the Redis caching system. The middleware that handles the caching system is running asynchronously so that the PeerTube program does not have to wait until Redis replies back with an cached video or "No Cached Result".

One of the issues that was posted in Github that I believed it could be improved was the information of the uploaded video. Right now, PeerTube does not show a lot of features to the uploaded video. For example, it does not show what resolution are available nor what kind of format/codec it is using. For this issue, it's more of an addition of a feature rather than a whole architecture overhaul. When we look at the "video-upload.component.ts" in Github, it has the following code.

```
const formData = new FormData()
formData.append('name', name)
// Put the video "private" -> we are waiting the user validation of the second step
formData.append('privacy', VideoPrivacy.PRIVATE.toString())
formData.append('nsfw', '' + nsfw)
formData.append('commentsEnabled', '' + commentsEnabled)
formData.append('downloadEnabled', '' + downloadEnabled)
formData.append('waitTranscoding', '' + waitTranscoding)
formData.append('channelId', '' + channelId)
formData.append('videofile', videofile)
```

When we append additional form data to the video such as available resolution or types of codecs, it would be easier to pull that data in the front end and make it visible for the Uploaders/Viewers to use.

Another issue that was posted in Github is recommending title related videos. Right now the recommendations are made by upload date format which gives you videos that are less relevant. However, this would require some significant amount of modification. If we just look for a title related video by searching, it would take a lot of time due to the amount of videos it needs to go though. Adding a preprocessing step where it clusters related videos might be better in terms of runtime optimization. So when a new video gets uploaded, it gets placed within a cluster with similar titles. When the user requests for a related video, it could simply look pick within the cluster rather than searching through all the videos known to your instance.

→ Additional information on Running the project to be added soon