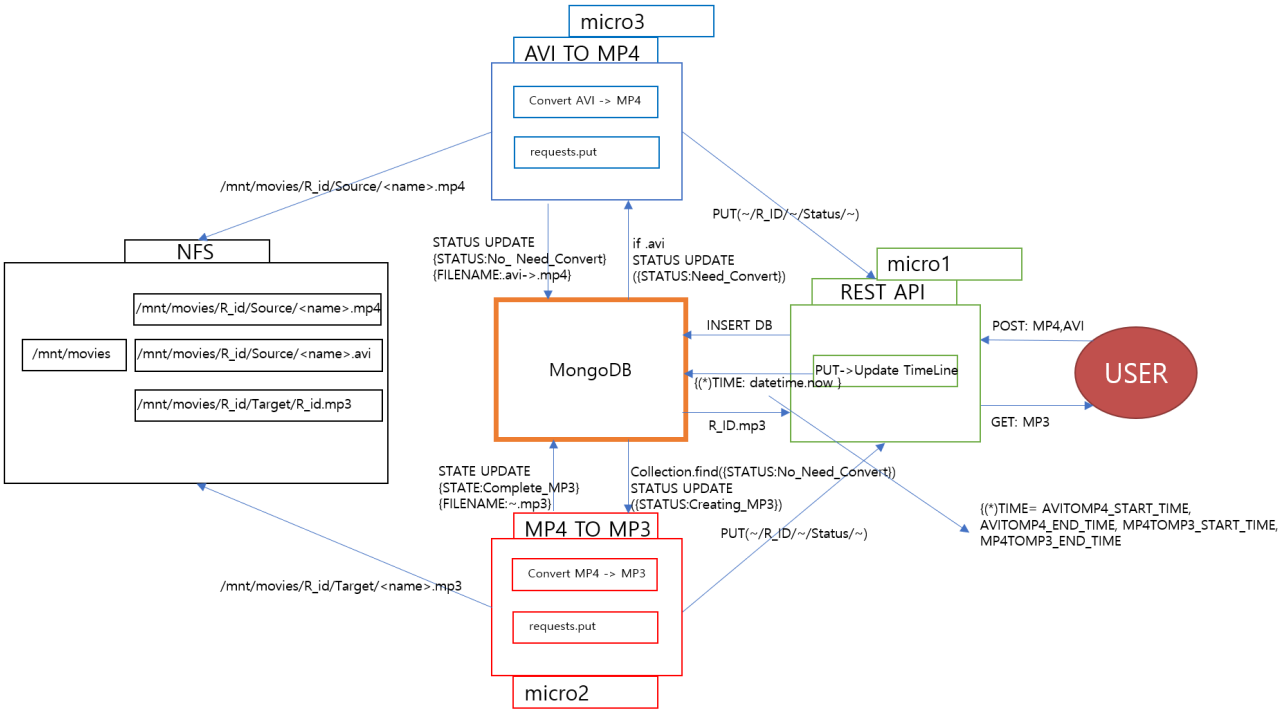
MP4AVI TO MP3 microservices

MP4AVI TO MP3 microservice 구조

- 전체 구조

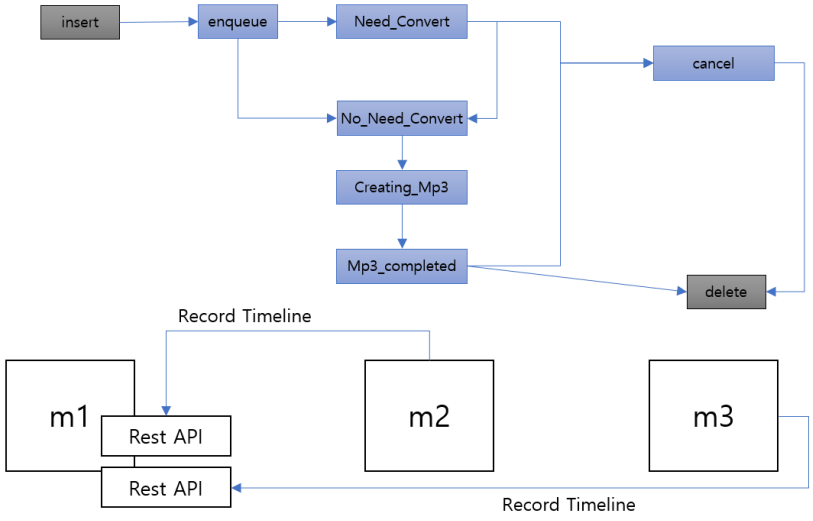
- 작업 상태 구조 및 DB 구조
- 폴더 구조
    - micro1
        - docker
        - yaml파일들 + (Service)
        - Request.py
        - requirements.txt
    - micro2
        - docker
        - yaml파일들
        - Convert.py
        - requirements.txt
    - micro3
        - docker
        - yaml파일들
        - display.py
        - requirements.txt
- 시연 과정 및 화면
    - avi파일을 mp4로 변환
    - mp4파일을 mp3로 변환
    - mp3파일 다운로드

MP4AVI TO MP3 microservice 프로젝트의 구조



작업 상태 구조 및 DB 구조

- lifecycle과 DB는 다음과 같은 상태로 구성되어 있습니다.



ObjectId	DIR	FILENAME	STATUS	CURPOD	STARTTIME	AVITOMP4_START_TIME	AVITOMP4_END_TIME	MP4TOMP3_START_TIME	MP4TOMP3_END_TIME

- ObjecId: MongoDB에 데이터를 저장할 때 별도로 지정하지 않은 경우 자동으로 부여되는 값
- DIR: 현재 처리하고 있는 mp4,avi의 경로 혹은 처리가 완료된 mp3의 경로를 나타냄
- FILENAME: 현재 처리하고 있는 mp4,avi의 이름 혹은 처리가 완료된 mp3의 이름을 나타냄
- STATUS: 총 6개의 상태로 이루어짐
  (Enqeue,Need_Convert,No_Need_Convert,Creating_MP3,MP3_Completed,Deleted)

- CURPOD: 현재 어떤 Pod에서 작업이 진행되고 있는지를 나타내는 값이며 총 3개(WEBPOD, MP4TOMP3POD, AVITOMP4POD)로 이루어짐
- STARTTIME: Request가 DB에 들어갔을 때의 Pod1에서의 시간
- AVITOMP4_START_TIME: Pod3에서 AVI를 MP4로 바꾸는 작업이 시작됐을 때 REST API를 이용해 Pod1에 기록된 시간
- AVITOMP4_END_TIME: Pod3에서 AVI를 MP4로 바꾸는 작업이 완료됐을 때 REST API를 이용해 Pod1에 기록된 시간
- MP4TOMP3_START_TIME: Pod2에서 MP4를 MP3로 바꾸는 작업이 시작됐을 때 REST API를 이용해 Pod1에 기록된 시간
- MP4TOMP3_START_TIME: Pod2에서 MP4를 MP3로 바꾸는 작업이 완료됐을 때 REST API를 이용해 Pod1에 기록된 시간

폴더구조

- hyeonseong
  - hsdef
    - hsdef.py
  - micro1
    - docker
      - Dockerfile
    - hyeonseong-reqDeploy.yaml
    - hyeonseong-reqPV.yaml
    - hyeonseong-reqPVC.yaml
    - hyeonseong-reqSVC.yaml
    - Request.py
    - requirements.txt
  - micro2
    - docker
      - Dockerfile
    - hyeonseong-mp4-to-mp3-deploy.yaml
    - mp4tomp3.py
    - requirements.txt
  - micro3
    - docker
      - Dockerfile
    - hyeonseong-avi-to-mp4-deploy.yaml
    - avitomp4.py
    - requirements.txt
  - templates
    - resdelete.html
    - resorback.html
    - result.html
    - upload.html
  - micro1_restart.sh
  - micro2_restart.sh
  - micro3_restart.sh

hsdef

- hsdef.py: 자주 사용되는 변수들을 모듈 형태로 저장한 파일입니다.

micro1

- docker/Dockerfile: docker 컨테이너를 빌드할 수 있는 Dockerfile이 있는 디렉토리 입니다.
- hyeonseong-reqDeploy.yaml.yaml: mp4에서 mp3로의 변환을 실행하는 Pod를 실행할 수 있는 yaml파일 입니다. 타입은 Deployment이며, /mnt/ 위치에 Persistent Volume(PV)를 생성합니다. 해당 PV는 hyeonseong-nfs-pvc라는 이름을 가진 Persistent Volume Claim(PVC)를 통해 각 Pod에 연결됩니다.
- hyeonseong-reqSvc.yaml: label selector로 접근이 가능하며 app: hyeonseong-req를 가진 Pod를 대상으로 서비스 합니다. type은 NodePort로, 30111 포트를 사용합니다.
- requirements.txt: Flask,Pymongo 등 Dockerfile에서 RUN할 모듈들이 txt파일로 저장되어 있습니다.

micro2

- docker/Dockerfile: docker 컨테이너를 빌드할 수 있는 Dockerfile이 있는 디렉토리 입니다.
- hyeonseong-mp4-to-mp3-deploy.yaml: mp4에서 mp3로의 변환을 실행하는 Pod를 실행할 수 있는 yaml 파일 입니다.
- requirements.txt: Flask,Pymongo 등 Dockerfile에서 RUN할 모듈들이 txt파일로 저장되어 있습니다.

micro3

- docker/Dockerfile: docker 컨테이너를 빌드할 수 있는 Dockerfile이 있는 디렉토리 입니다.
- hyeonseong-avi-to-mp4-deploy.yaml: mp4에서 mp3로의 변환을 실행하는 Pod를 실행할 수 있는 yaml 파일 입니다.
- requirements.txt: Flask,Pymongo 등 Dockerfile에서 RUN할 모듈들이 txt파일로 저장되어 있습니다.

templates

- 해당하는 REST API가 호출 되었을 때 가져올 html문서들을 관리하는 디렉토리 입니다.

micro{n}_restart.sh

- 초기에 700 권한을 준 후 실행하면 현재 running중인 Pod를 삭제하고, Dockerfile을 Build하고, tag를 설정해 Pull하여 Pod에서 해당 이미지를 사용할 수 있는 상태로 만든 뒤, yaml파일을 실행해 Pod 및 Deployment를 실행합니다. Setting
- mongoDB가 잘 연결되었는지 확인

```
$ service mongod status
$ sudo systemctl start mongod
```

- microservice를 진행하기 전 PV와 PVC 생성

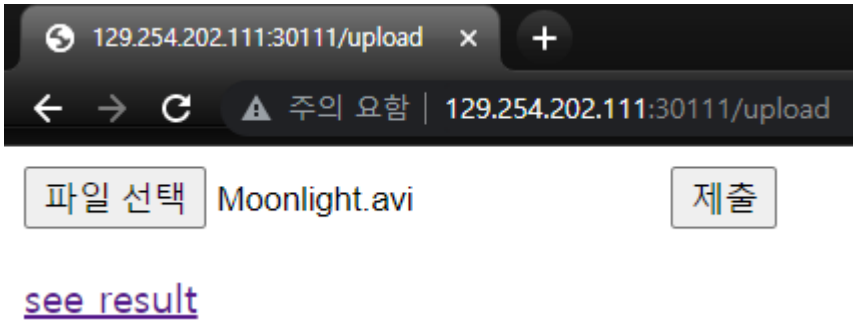
```
$ kubectl apply -f hyeonseong-reqPV.yaml
$ kubectl apply -f hyeonseong-reqPVC.yaml
```

- Docker Container 이미지 생성 및 Pod 생성

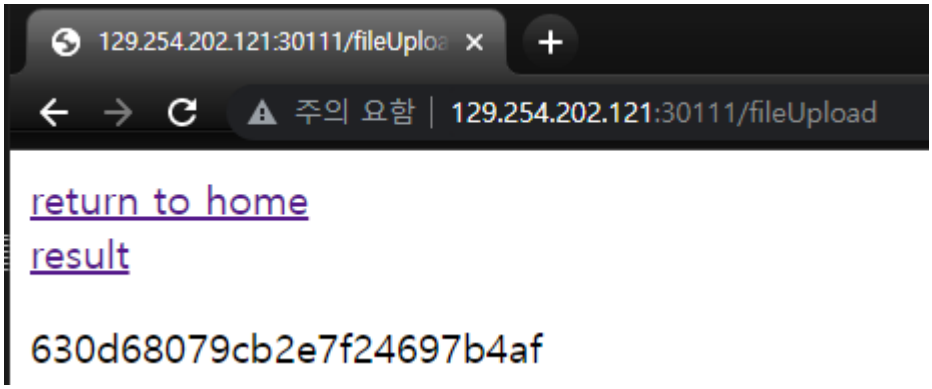
```
$ sudo docker build -t hsr -f ./micro1/docker/Dockerfile . &&
sudo docker tag hsr hyeonseong0917/hsr &&
sudo docker push hyeonseong0917/hsr &&
kubectl apply -f ./micro1/hyeonseong-reqDeploy.yaml
$ kubectl apply -f hyeonseong-reqSvc.yaml # Pod를 생성한 후에 Service를 적용해야 함
$ sudo docker build -t hsc -f ./micro2/docker/Dockerfile . &&
sudo docker tag hsc hyeonseong0917/hsc &&
sudo docker push hyeonseong0917/hsc &&
kubectl apply -f ./micro2/hyeonseong-mp4-to-mp3-deploy.yaml
$ sudo docker build -t hsd -f ./micro3/docker/Dockerfile . &&
sudo docker tag hsd hyeonseong0917/hsd &&
sudo docker push hyeonseong0917/hsd &&
kubectl apply -f ./micro3/hyeonseong-avi-to-mp4-deploy.yaml
```

시연 과정 및 화면

- 인터넷 브라우저를 통해 Workernode 혹은 Masternode의 IP를 입력하고 hyeonseong-reqSvc.yaml에서 정의한 nodeport 번호를 입력 합니다. 그 후 /upload로 이동해 avi파일을 선택했습니다.

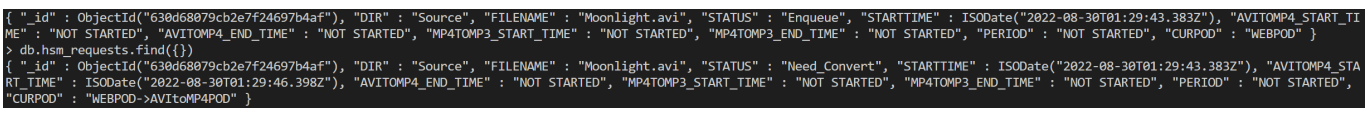


- 제출 버튼을 누르면 그 순간부터 microservice를 진행하게 됩니다.



- /fileUpload로 라우팅되며 Object_Id가 표시되고, /upload 로 돌아가는 return, 결과를 볼 수 있는 result 하이퍼링크가 표시됩니다. 결과를 보기 위해 result를 누를 수 있습니다.

- DB의 STATUS 및 FILENAME 변환 과정을 보겠습니다.



- FILENAME이 처음에는 .avi에 CURPOD가 WEBPOD이고, STATUS가 Enqueue인 초기 상태 였지만 곧 AVItoMP4POD(microservice3)로 이동하여 변환 과정을 거치게 되고, 변환의 시작에서 Timeline이 기록되

며, FILENAME이 .mp4로 바뀌었을 뿐만 아니라 .avi파일이기 때문에 STATUS가 Need_Convert로 바뀐 것을 확인할 수 있습니다.

만약 .mp4 파일이라면 이 순간에 STATUS가 No_Need_Convert로 바뀔 것입니다.


```
{ "_id" : ObjectId("630d68079cb2e7f24697b4af"), "DIR" : "Source", "FILENAME" : "Moonlight.avi", "STATUS" : "Need_Convert", "STARTTIME" : ISODate("2022-08-30T01:29:43.383Z"), "AVITOMP4_START_TIME" : ISODate("2022-08-30T01:29:46.398Z"), "AVITOMP4_END_TIME" : "NOT STARTED", "MP4TOMP3_START_TIME" : "NOT STARTED", "MP4TOMP3_END_TIME" : "NOT STARTED", "PERIOD" : "NOT STARTED", "CURPOD" : "WEBPOD->AVItoMP4POD" }
> db.hsm_requests.find({})
{ "_id" : ObjectId("630d68079cb2e7f24697b4af"), "DIR" : "Source", "FILENAME" : "Moonlight.mp4", "STATUS" : "No_Need_Convert", "STARTTIME" : ISODate("2022-08-30T01:29:43.383Z"), "AVITOMP4_START_TIME" : ISODate("2022-08-30T01:29:46.398Z"), "AVITOMP4_END_TIME" : ISODate("2022-08-30T01:29:51.418Z"), "MP4TOMP3_START_TIME" : "NOT STARTED", "MP4TOMP3_END_TIME" : "NOT STARTED", "PERIOD" : "NOT STARTED", "CURPOD" : "WEBPOD->AVItoMP4POD->AVItoMP4POD" }
```

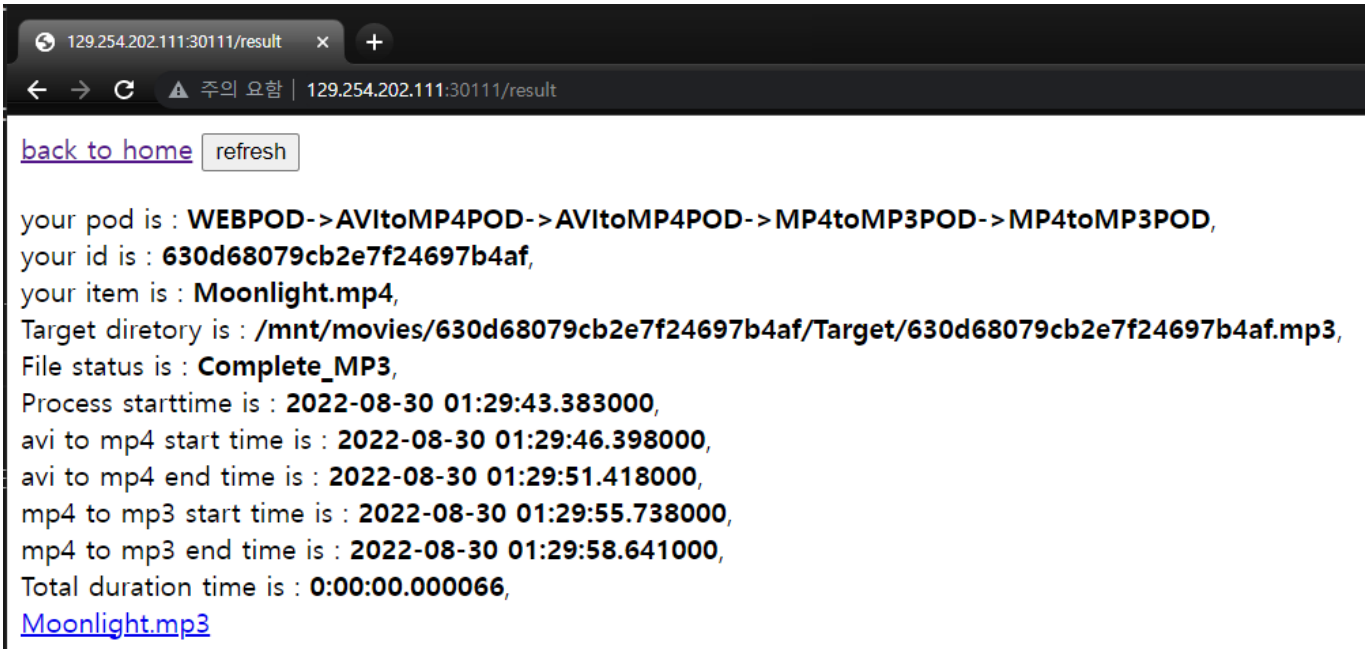

- AVI를 MP4로 변환하는 작업을 완료하면 변환의 끝에서 Timeline이 기록되며, STATUS가 Need_Convert에서 No_Need_Convert로 바뀐 것을 확인할 수 있습니다.


```
{ "_id" : ObjectId("630d68079cb2e7f24697b4af"), "DIR" : "Source", "FILENAME" : "Moonlight.mp4", "STATUS" : "No_Need_Convert", "STARTTIME" : ISODate("2022-08-30T01:29:43.383Z"), "AVITOMP4_START_TIME" : ISODate("2022-08-30T01:29:46.398Z"), "AVITOMP4_END_TIME" : ISODate("2022-08-30T01:29:51.418Z"), "MP4TOMP3_START_TIME" : "NOT STARTED", "MP4TOMP3_END_TIME" : "NOT STARTED", "PERIOD" : "NOT STARTED", "CURPOD" : "WEBPOD->AVItoMP4POD->AVItoMP4POD" }
> db.hsm_requests.find({})
{ "_id" : ObjectId("630d68079cb2e7f24697b4af"), "DIR" : "Source", "FILENAME" : "Moonlight.mp4", "STATUS" : "Creating_MP3", "STARTTIME" : ISODate("2022-08-30T01:29:43.383Z"), "AVITOMP4_START_TIME" : ISODate("2022-08-30T01:29:46.398Z"), "AVITOMP4_END_TIME" : ISODate("2022-08-30T01:29:51.418Z"), "MP4TOMP3_START_TIME" : ISODate("2022-08-30T01:29:55.738Z"), "MP4TOMP3_END_TIME" : "NOT STARTED", "PERIOD" : "NOT STARTED", "CURPOD" : "WEBPOD->AVItoMP4POD->AVItoMP4POD->MP4toMP3POD" }
```

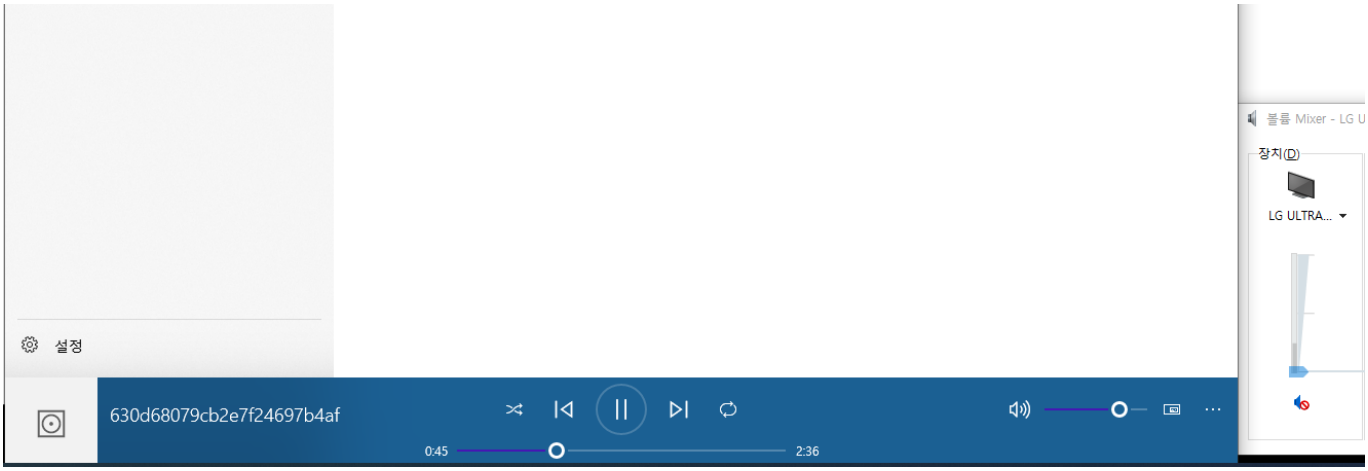

- 그 후 MP4TOMP3 POD로 이동하여 Timeline 기록 및 mp4->mp3 변환을 진행합니다.
- 변환을 시작할 때 STATUS가 NO_NEED_CONVERT에서 Creating_MP3로 바뀐 것을 확인할 수 있습니다.


```
{ "_id" : ObjectId("630d68079cb2e7f24697b4af"), "DIR" : "Source", "FILENAME" : "Moonlight.mp4", "STATUS" : "Creating_MP3", "STARTTIME" : ISODate("2022-08-30T01:29:43.383Z"), "AVITOMP4_START_TIME" : ISODate("2022-08-30T01:29:46.398Z"), "AVITOMP4_END_TIME" : ISODate("2022-08-30T01:29:51.418Z"), "MP4TOMP3_START_TIME" : ISODate("2022-08-30T01:29:55.738Z"), "MP4TOMP3_END_TIME" : "NOT STARTED", "PERIOD" : "NOT STARTED", "CURPOD" : "WEBPOD->AVItoMP4POD->AVItoMP4POD->MP4toMP3POD" }
> db.hsm_requests.find({})
{ "_id" : ObjectId("630d68079cb2e7f24697b4af"), "DIR" : "/mnt/movies/630d68079cb2e7f24697b4af/Target/630d68079cb2e7f24697b4af.mp3", "FILENAME" : "Moonlight.mp4", "STATUS" : "Complete_MP3", "STARTTIME" : ISODate("2022-08-30T01:29:43.383Z"), "AVITOMP4_START_TIME" : ISODate("2022-08-30T01:29:46.398Z"), "AVITOMP4_END_TIME" : ISODate("2022-08-30T01:29:51.418Z"), "MP4TOMP3_START_TIME" : ISODate("2022-08-30T01:29:55.738Z"), "MP4TOMP3_END_TIME" : ISODate("2022-08-30T01:29:58.641Z"), "PERIOD" : "0:00:00.000066", "CURPOD" : "WEBPOD->AVItoMP4POD->AVItoMP4POD->MP4toMP3POD->MP4toMP3POD" }
```


- mp3로의 변환이 끝나면 STATUS가 Creating_MP3에서 ComPlete_MP3로 바뀌고, 마찬가지로 시작과 끝에서 Timeline이 기록되며 총 시간의 합계가 PERIOD에 나타나게 됩니다.



- 이제 /result로 이동하여 웹 페이지를 보면 MongoDB에 있는 내용들이 표시되며 하이퍼링크를 통해 mp3 파일을 다운로드 받을 수 있습니다.



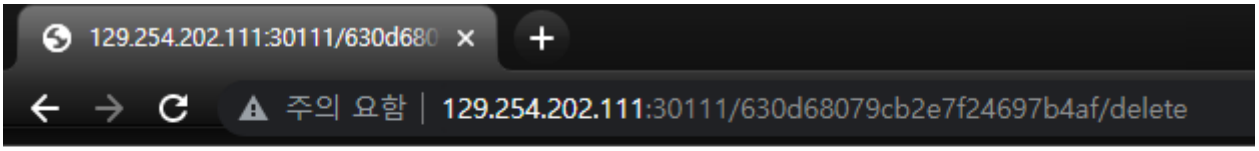
- 다운로드를 한 뒤에 삭제할 수 있는 링크가 생깁니다.

back to home refresh

your pod is : **WEBPOD->AVItoMP4POD->AVItoMP4POD->MP4toMP3POD->MP4toMP3POD**,
your id is : **630d68079cb2e7f24697b4af**,
your item is : **Moonlight.mp4**,
Target diretory is : **/mnt/movies/630d68079cb2e7f24697b4af/Target/630d68079cb2e7f24697b4af.mp3**,
File status is : **Deleted**,
Process starttime is : **2022-08-30 01:29:43.383000**,
avi to mp4 start time is : **2022-08-30 01:29:46.398000**,
avi to mp4 end time is : **2022-08-30 01:29:51.418000**,
mp4 to mp3 start time is : **2022-08-30 01:29:55.738000**,
mp4 to mp3 end time is : **2022-08-30 01:29:58.641000**,
Total duration time is : **0:00:00.000066**,

630d68079cb2e7f24697b4af delete

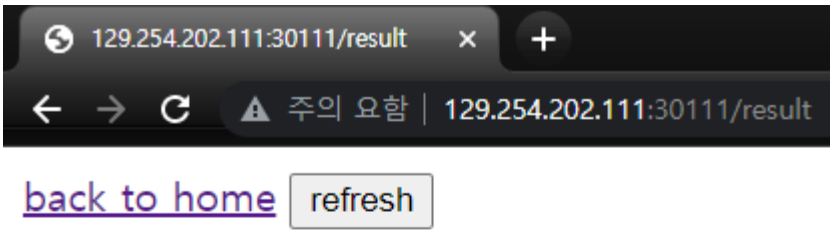
- delete를 누르면



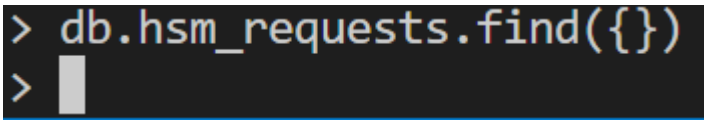
return to home
result

630d68079cb2e7f24697b4af Deleted

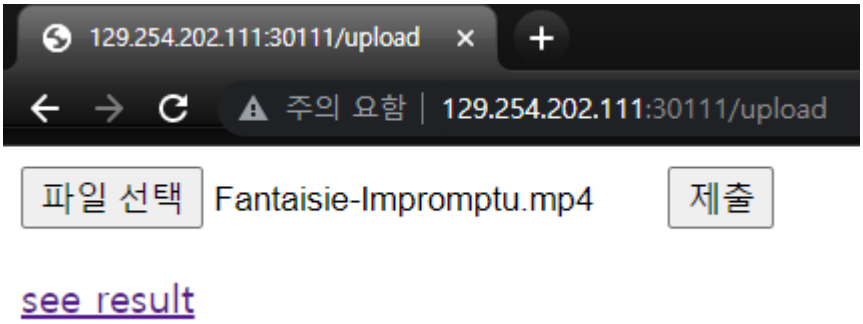
- deleted 되었다는 문구가 뜹니다.



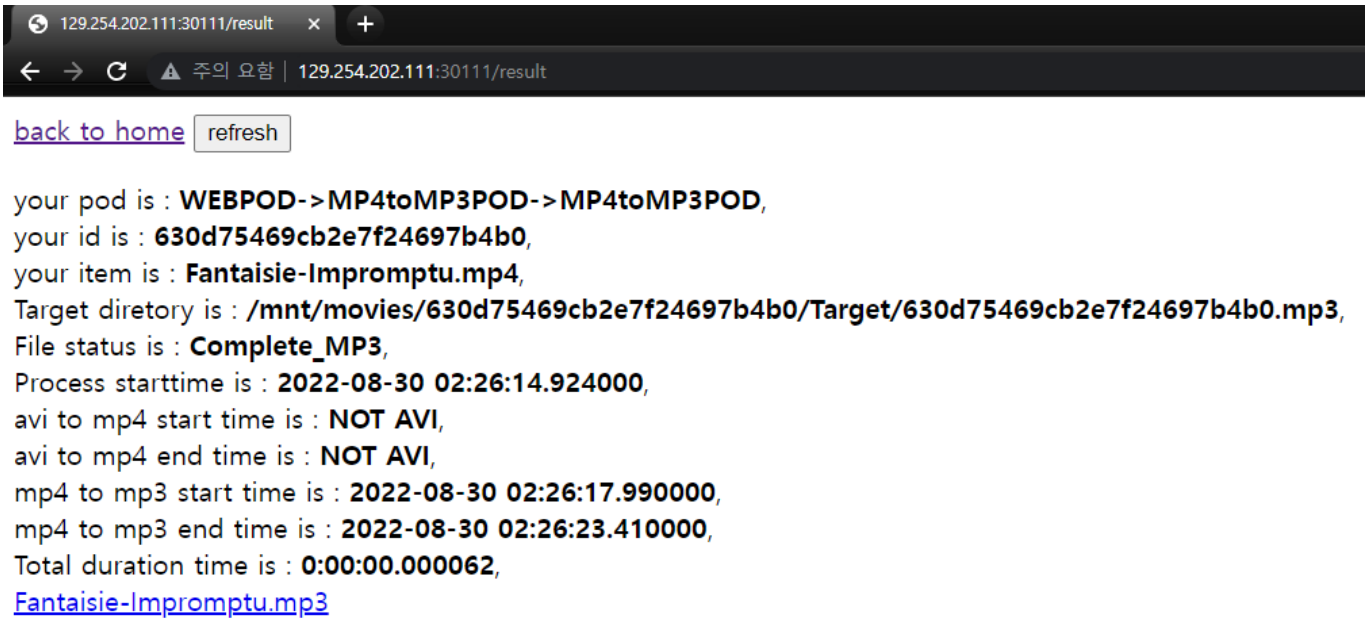
- /result로 다시 가보면 모든 내용들이 사라진 것을 확인할 수 있습니다.



- MongoDB의 Collection에서도 더이상 검색되지 않습니다.
- mp4를 input file로 넣었을 때도 큰 틀은 다르지 않습니다. 위와 다른 부분만을 짚어서 보겠습니다.



- AVITOMP4 Timeline들이 "NOT AVI"라고 표시되며 오직 MP4TOMP3POD만 거치는 것을 확인할 수 있습니다.



- 그 외에는 크게 다르지 않게 표시되고 있습니다.

Timeline을 설정했던 방법

- 현재 작업 환경이 Masternode 1대, Workernode 2대로, 총 3대가 있지만 각각의 Pod들이 다른 Workernode에 할당된다면 Timeline이 의도치 않게 표시 될 수 있습니다. 각각 node들의 시간이 같다고 보장할 수 없기 때문이죠. 이를 해결하기 위해 microservice1(REST API)를 실행하는 yaml파일에 nodeSelector를 통해 해당 Pod는 scale에 관계없이 오직 한 노드에만 할당되도록 하였습니다. 그 다음, microservice 2 혹은 3에서 각각의 convert작업이 시작되거나 끝났을 때,

  (microservice 3를 예로 들었습니다.)

```
firsturi=os.path.join("http://129.254.202.111:30111","requestids",R_id,"statuses",hsdef.NEED_CONVERT)
r=requests.put(firsturi)
print(r)
time.sleep(5)
avi_file_path=os.path.join(hsdef.MOVIE_DIR,R_id,hsdef.SRC,filename) #avi path
output_name=os.path.join(hsdef.MOVIE_DIR,R_id,hsdef.SRC,filename[:-4]) #mp4 path
# avi->mp4

os.popen("ffmpeg -fflags +genpts -i '{input}' -c:v copy -c:a copy '{output}.mp4'".format(input = avi_file_

hs_movie.hsm_requests.update_one({"_id":q["_id"]},{"$set":{"STATUS":hsdef.NO_NEED_CONVERT}})
seconduri=os.path.join("http://129.254.202.111:30111","requestids",R_id,"statuses",hsdef.NO_NEED_CONVERT)
requests.put(seconduri)
```

  이와 같이 put method를 이용해 R_id,status를 바탕으로 microservice 1의 API에 접근하여

```
elif status == hsdef.NEED_CONVERT:
    curPod=""
    for i in hs_doc:
        curPod=(i["CURPOD"]+"->AVItoMP4POD")
    hs_movie.hsm_requests.update_one({"_id":ObjectId(request_id)},{"$set":{"CURPOD":curPod}})
    hs_movie.hsm_requests.update_one({"_id":ObjectId(request_id)},{"$set":{"AVITOMP4_START_TIME":datetime.datetime.now()}})
elif status==hsdef.NO_NEED_CONVERT:
    curPod=""
    for i in hs_doc:
        curPod=(i["CURPOD"]+"->AVItoMP4POD")
    hs_movie.hsm_requests.update_one({"_id":ObjectId(request_id)},{"$set":{"CURPOD":curPod}})
    hs_movie.hsm_requests.update_one({"_id":ObjectId(request_id)},{"$set":{"AVITOMP4_END_TIME":datetime.datetime.now()}})
```

API에 접근했을 때의 timeline을 DB에 기록하는 방식으로 각각의 Timeline을 설정할 수 있었습니다.