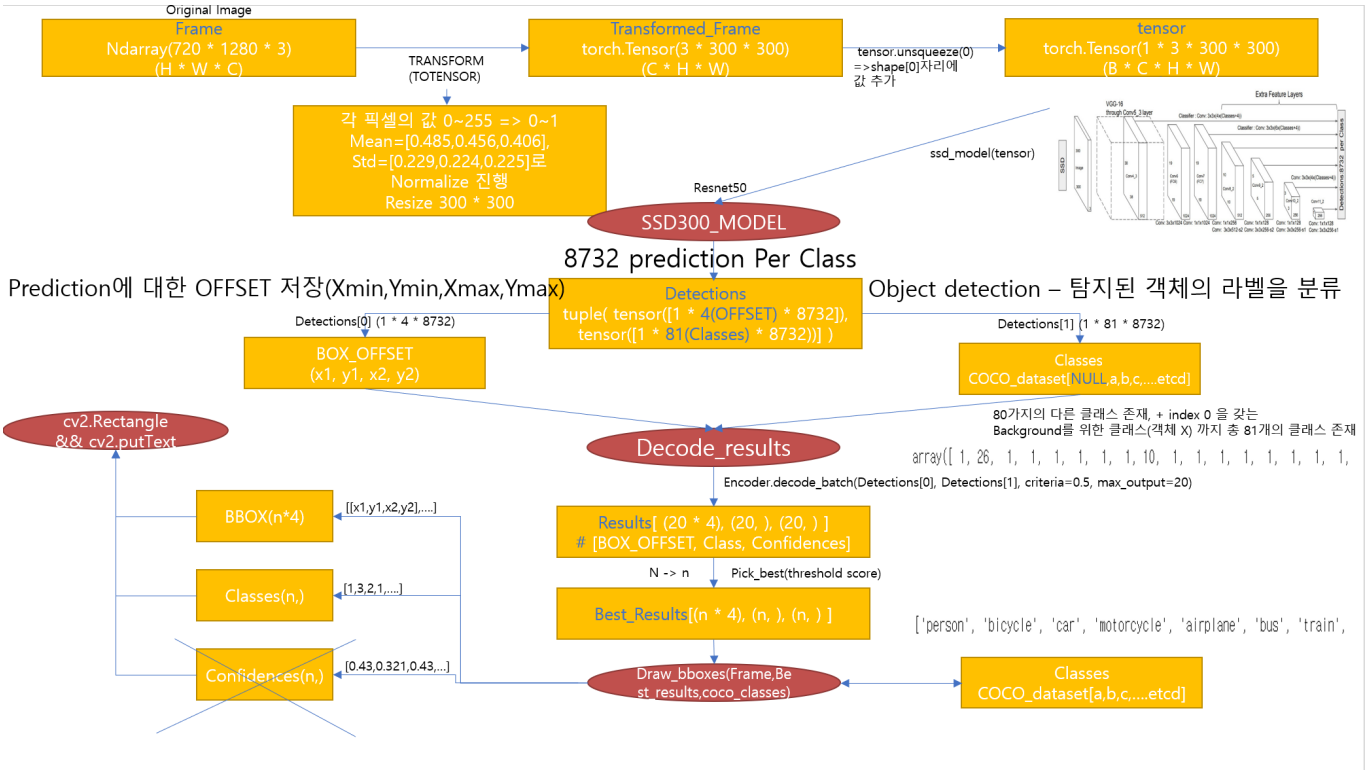
Pytorch Resnet Object Detection Microservices

Pytorch Resnet Object Detection Microservice 구조

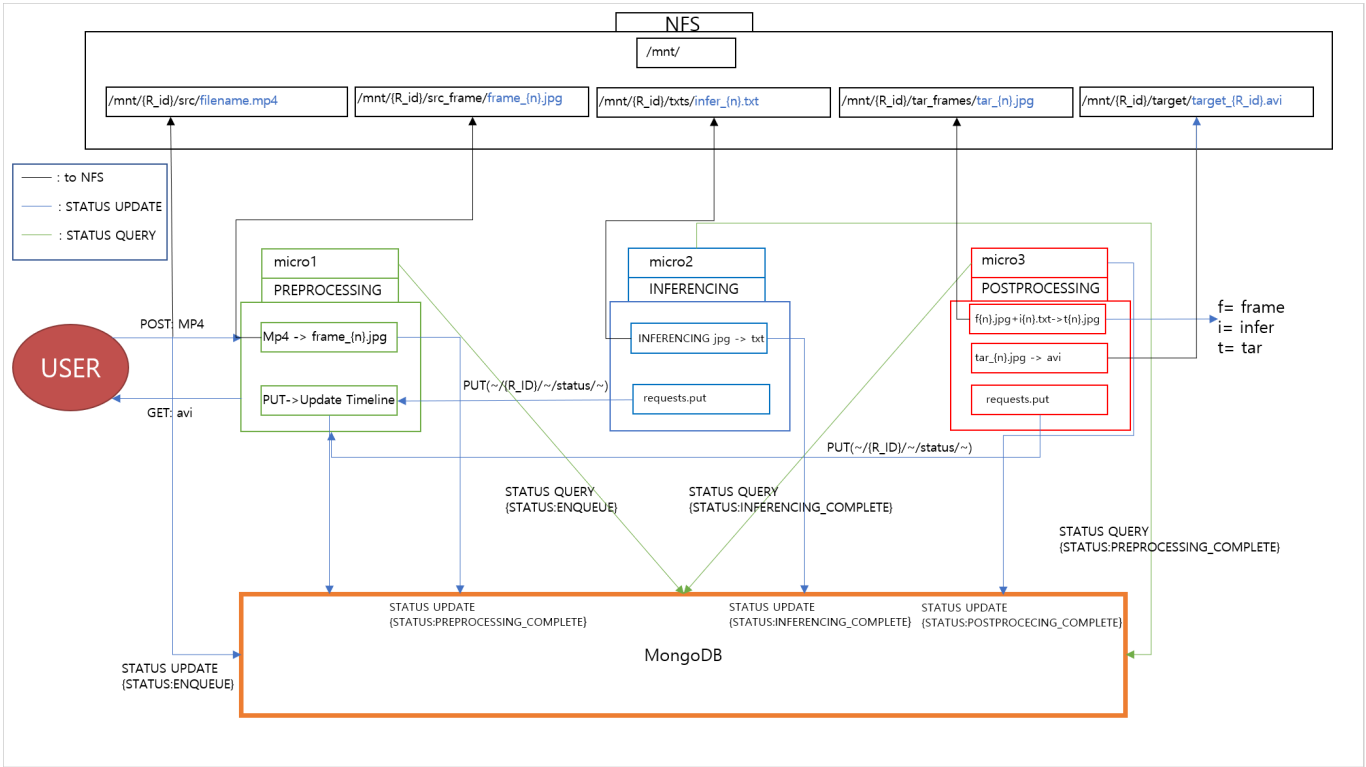
- 전체 구조

- 작업 상태 구조 및 DB 구조
- 폴더 구조
  - micro1
    - docker
    - yaml파일들 + (Service)
    - pre.py
    - requirements.txt
  - micro2
    - docker
    - yaml파일들
    - infer.py
    - requirements.txt
  - micro3
    - docker
    - yaml파일들
    - pody.py
    - requirements.txt
  - clusterrole.yaml
  - clusterrolebinding.yaml
- 시연 과정 및 화면
  - mp4파일을 프레임 단위로 저장하는 PreProcessing
  - 저장된 프레임 단위 이미지를 Resnet50 기반 SSD300 모델을 통해 Inferencing
  - 각각 Inferencing된 결과를 프레임 번호에 해당하는 txt파일에 저장
  - txt파일과 저장된 프레임을 결합해 탐지된 객체에 BOX가 BOUNDING된 새 프레임을 만들고 avi파일로 만드는 PostProcessing
  - Kubernetes Python Api를 사용해 Deployment의 Replicaset 조정

Pytorch로 Resnet Object Detection 을 이용해 Bounding Box를 진행하는 프로젝트의 흐름도

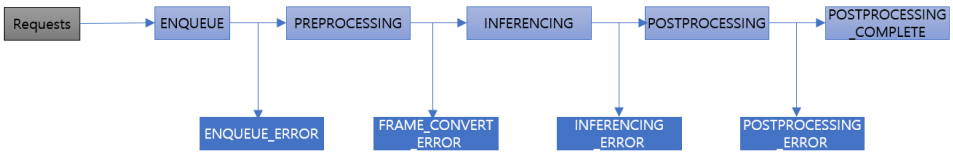


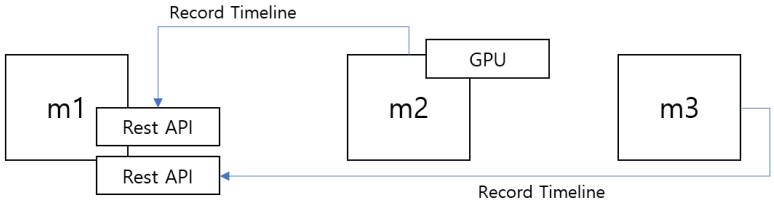
Pytorch Resnet Object Detection Microservice 프로젝트의 구조



작업 상태 구조 및 DB 구조

- lifecycle과 DB 및 마이크로 서비스 간 구조는 다음과 같은 상태로 구성되어 있습니다.





ObjectId	FILENAME	STATUS	PROCESS_START	PREPROCESSING_START	PREPROCESSING_END	INFERENCING_START	INFERENCING_END	POSTPROCESSING_START	POSTPROCESSING_END	PROCESS_END	TOTAL

- ObjectId: MongoDB에 데이터를 저장할 때 별도로 지정하지 않은 경우 자동으로 부여되는 값
- FILENAME: 가장 초기에 Request로 들어온 File의 이름
- STATUS: 6개의 정상 상태(ENQUEUE, PREPROCESSING, PREPROCESSING_COMPLETE, INFERENCING, INFERENCING_COMPLETE,POSTPROCESSING, POSTPROCESSING_COMPLETE), 4개의 비정상 상태 (ENQUEUE_ERROR, FRAME_CONVERT_ERROR, INFERENCING_ERROR, POSTPROCESSING_ERROR)로 총 10개가 존재
- PROCESS_START: Request가 DB에 들어갔을 때의 Pod1에서의 시간
- PREPROCESSING_START: PREPROCESSING 작업을 시작할 때의 시간
- PREPROCESSING_END: PREPROCESSING 작업을 완료했을 때의 시간
- INFERENCING_START: INFERENCING 작업을 시작할 때의 시간
- INFERENCING_END: INFERENCING 작업을 완료했을 때의 시간
- POSTPROCESSING_START: POSTPROCESSING 작업을 시작할 때의 시간
- POSTPROCESSING_END: POSTPROCESSING 작업을 완료했을 때의 시간
- PROCESS_END: 전체적인 작업을 완료했을 때의 시간
- TOTAL: 전체 작업을 완료하는데 소요된 시간 (PROCESS_END - PROCESS_START)

폴더구조

- hs_res
  - hsdef
    - hsdef.py
  - micro1
    - docker
      - Dockerfile
    - pre-deploy.yaml
    - pre-PV.yaml
    - pre-PVC.yaml
    - pre-SVC.yaml
    - pre.py
    - micro1_python_k8s_api.py
    - requirements.txt
  - micro2
    - docker

        - Dockerfile
    - infer-deploy.yaml
    - infer.py
    - requirements.txt
- micro3
    - docker
        - Dockerfile
    - post-deploy.yaml
    - post.py
    - micro3_python_k8s_api.py
    - requirements.txt
- templates
    - resdelete.html
    - resorback.html
    - result.html
    - upload.html
- clusterrole.yaml
- clusterrolebinding.yaml
- micro1_restart.sh
- micro2_restart.sh
- micro3_restart.sh

resdef

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

micro1

- docker/Dockerfile: docker 컨테이너를 빌드할 수 있는 Dockerfile이 있는 디렉토리 입니다.
- pre-deploy.yaml: mp4에서 각 프레임을 넘버링 된 jpg로 변환하는 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.
- pre-SVC.yaml: label selector로 접근이 가능하며 app: hs-pre를 가진 Pod를 대상으로 서비스 합니다. type은 NodePort로, 30232 포트를 사용합니다.
- micro1_python_k8s_api.py: Python에서 k8s API를 접근하기 위해 Pod 내에서 실행할 수 있는 파일입니다.
- requirements.txt: Flask,Pymongo 등 Dockerfile에서 RUN할 모듈들이 txt파일로 저장되어 있습니다.

micro2

- docker/Dockerfile: docker 컨테이너를 빌드할 수 있는 Dockerfile이 있는 디렉토리 입니다.
- infer-deploy.yaml: 넘버링 된 각 프레임을 대상으로 Resnet 기반 SSD300_MODEL을 통해 INFERENCING 하여 txt파일로 추출하는 Pod를 실행하는 yaml파일 입니다.
- SSD300_MODEL은 CUDA Available Device에서만 작동할 수 있으므로, GPU가 있는 WorkerNode에 label을 추가해 nodeSelector에서 해당 노드를 지정하도록 해야합니다. 혹은 Python Code를 통해 적절한 디바이스를 찾아도 문제 없습니다.
- requirements.txt: Flask,Pymongo 등 Dockerfile에서 RUN할 모듈들이 txt파일로 저장되어 있습니다.

micro3

- docker/Dockerfile: docker 컨테이너를 빌드할 수 있는 Dockerfile이 있는 디렉토리 입니다.
- post-deploy.yaml: 넘버링 된 각 프레임과 INFERENCING된 txt파일을 결합하여 넘버링 된 tar_frame을 생성하고, tar_frame들을 순차적으로 결합하여 avi파일로 만드는 Pod를 실행하는 yaml파일 입니다.
- micro3_python_k8s_api.py: Python에서 k8s API를 접근하기 위해 Pod 내에서 실행할 수 있는 파일입니다.
- 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.

clusterrole.yaml

- Python에서 k8s API 접근을 허용하기 위해 클러스터 범위 리소스의 권한을 정의하는 clusterrole을 생성하는 파일 입니다.

clusterrolebinding.yaml

- RBAC 인증을 통해 clusterrole에 정의된 권한을 Python에서 k8s API 접근을 하려는 사용자에게 부여합니다.
- 본 과정에서는 사용자 대상으로 ServiceAccount에 name: default, namespace: default로 설정하였습니다.

Setting

- mongoDB가 잘 연결되었는지 확인

```
$ service mongod status
$ sudo systemctl start mongod
> use hs_resnet
> db.hs_requests.find({})
```

- microservice를 진행하기 전 PV와 PVC 생성

```
$ kubectl apply -f pre-PV.yaml
$ kubectl apply -f pre-PVC.yaml
```

- microservice를 진행하기 전 clusterrole과 clusterrolevinding 생성 (name과 namespace default인 ServiceAccount는 생성되어 있음)

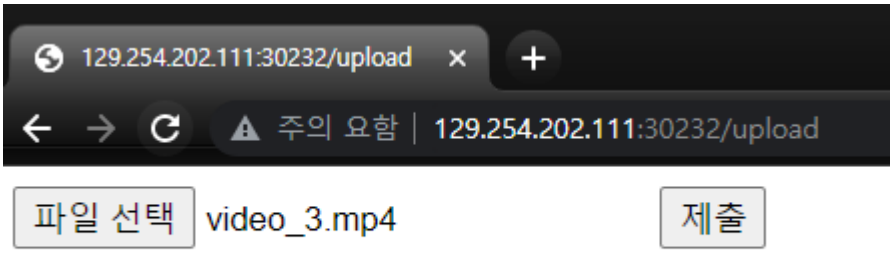
```
$ kubectl apply -f clusterrole.yaml
$ kubectl apply -f clusterrolebinding.yaml
```

- Docker Container 이미지 생성 및 Pod 생성

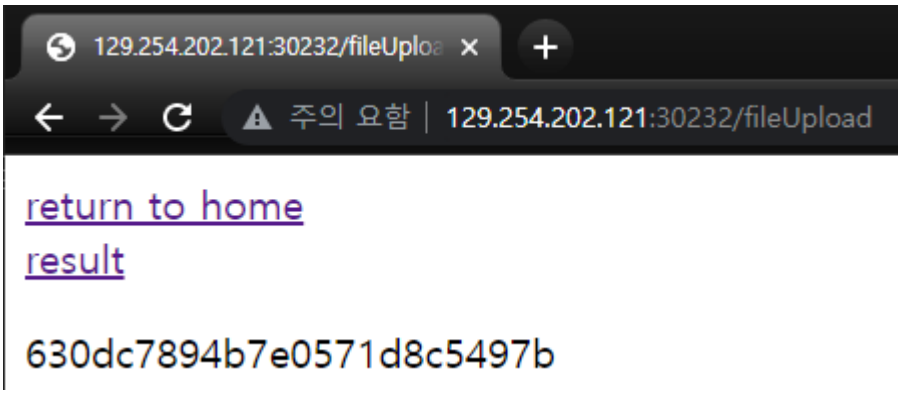
```
$ sudo docker build -t hs_pre -f ./micro1/docker/Dockerfile . &&
sudo docker tag hs_pre hyeonseong0917/hs_pre &&
sudo docker push hyeonseong0917/hs_pre &&
kubectl apply -f ./micro1/pre-deploy.yaml
$ kubectl apply -f pre-SVC.yaml # Pod를 생성한 후에 Service를 적용해야 함
$ sudo docker build -t hs_infer -f ./micro2/docker/Dockerfile . &&
sudo docker tag hs_infer hyeonseong0917/hs_infer &&
sudo docker push hyeonseong0917/hs_infer &&
kubectl apply -f ./micro2/infer-deploy.yaml
$ sudo docker build -t hs_post -f ./micro3/docker/Dockerfile . &&
sudo docker tag hs_post hyeonseong0917/hs_post &&
sudo docker push hyeonseong0917/hs_post &&
kubectl apply -f ./micro3/post-deploy.yaml

# 혹시 마이크로 서비스 실행에 문제가 생겼다면 해당 마이크로 서비스의 Dockerfile로 가셔서, 주석 처리 돼있는 # CMD ["python3",  "/hsapp/main.py"] 를 주석을 풀고, 기존에 있었던 CMD명령어 줄을 주석처리 한 다음에, restart를 합니다. 그 후 pod가 생성되면
1. kubectl exec -it <pod_name> /bin/bash 를 통해 pod 안쪽으로 접근
2. pod 안에서 기존에 실행하려는 Python파일 실행
과정을 통해 TroubleShooting 하시면 될 것 같습니다.
# main함수는 컨테이너가 종료되지 않게 시간 간격을 두고(time.sleep), while(1) 내에서 무한으로 순회하는 코드입니다.
pods에 접근 후 기존 Python 파일에 cnt를 찍어본다거나 print문을 적절히 사용하여 원활한 TroubleShooting 하시길 바랍니다.
# microservice 2  실행 시 SSD_MODEL을 불러오는데 오류가 생기면 pod 내에서
/root/.cache/ 에 있는 checkpoints 디렉토리를 삭제하시면 됩니다.
```

시연 과정 및 화면

- 인터넷 브라우저를 통해 Workernode 혹은 Masternode의 IP를 입력하고 pre-SVC.yaml에서 정의한 nodeport 번호를 입력 합니다. 그 후 /upload로 이동해 mp4파일을 선택했습니다.



- 제출 버튼을 누르면 그 순간부터 microservice를 진행하게 됩니다.



- /fileUpload로 라우팅되며 Object_Id가 표시되고, /upload 로 돌아가는 return, 결과를 볼 수 있는 result 하이러링크가 표시됩니다. 결과를 보기 위해 result를 누를 수 있습니다.
- DB의 STATUS 및 FILENAME 변환 과정을 보겠습니다.

```
> db.hs_requests.find({})
{ "_id" : ObjectId("630dc7894b7e0571d8c5497b"), "FILENAME" : "video_3.mp4", "STATUS" : "PREPROCESSING", "PROCESS_START" : ISODate("2022-08-30T08:17:13.663Z"), "PREPROCESSING_START" : ISOD
ate("2022-08-30T08:17:13.663Z"), "PREPROCESSING_END" : "WAITING..", "INFERENCING_START" : "WAITING..", "INFERENCING_END" : "WAITING..", "POSTPROCESSING_START" : "WAITING..", "POSTPROCESSI
NG_END" : "WAITING..", "PROCESS_END" : "WAITING..", "TOTAL" : "WAITING..." }
```

- 초기 STATUS가 ENQUEUE에서 mp4 파일이 /mnt/src/ 디렉토리에 저장되면 STATUS가 PREPROCESSING으로 바뀌게 됩니다.

```
{ "_id" : ObjectId("630dc7894b7e0571d8c5497b"), "FILENAME" : "video_3.mp4", "STATUS" : "PREPROCESSING_COMPLETE", "PROCESS_START" : ISODate("2022-08-30T08:17:13.663Z"), "PREPROCESSING_STAR
T" : ISODate("2022-08-30T08:17:13.663Z"), "PREPROCESSING_END" : ISODate("2022-08-30T08:17:27.065Z"), "INFERENCING_START" : "WAITING..", "INFERENCING_END" : "WAITING..", "POSTPROCESSING_ST
ART" : "WAITING..", "POSTPROCESSING_END" : "WAITING..", "PROCESS_END" : "WAITING..", "TOTAL" : "WAITING..." }
> db.hs_requests.find({})
{ "_id" : ObjectId("630dc7894b7e0571d8c5497b"), "FILENAME" : "video_3.mp4", "STATUS" : "INFERENCING", "PROCESS_START" : ISODate("2022-08-30T08:17:13.663Z"), "PREPROCESSING_START" : ISODat
e("2022-08-30T08:17:13.663Z"), "PREPROCESSING_END" : ISODate("2022-08-30T08:17:27.065Z"), "INFERENCING_START" : ISODate("2022-08-30T08:17:27.388Z"), "INFERENCING_END" : "WAITING..", "POST
PROCESSING_START" : "WAITING..", "POSTPROCESSING_END" : "WAITING..", "PROCESS_END" : "WAITING..", "TOTAL" : "WAITING..." }
```

- microservice1에서 PreProcessing 작업이 완료되면 STATUS가 PREPROCESSING_COMPLETE로 바뀌게 되며, 이로 인해 microservice2가 작동하게 되어 STATUS가 INFERENCING으로 바뀌게 됩니다.
- PreProcessing 과정을 완료한 후 결과물의 일부는 다음과 같습니다.

```
root@hs-pre-deploy-b77488b9-sgrdn:/mnt/630dc7894b7e0571d8c5497b/src_frame# ls
frame_0.jpg    frame_193.jpg  frame_288.jpg  frame_382.jpg  frame_477.jpg
frame_1.jpg    frame_194.jpg  frame_289.jpg  frame_383.jpg  frame_478.jpg
frame_10.jpg   frame_195.jpg  frame_29.jpg   frame_384.jpg  frame_479.jpg
frame_100.jpg  frame_196.jpg  frame_290.jpg  frame_385.jpg  frame_48.jpg
frame_101.jpg  frame_197.jpg  frame_291.jpg  frame_386.jpg  frame_480.jpg
frame_102.jpg  frame_198.jpg  frame_292.jpg  frame_387.jpg  frame_481.jpg
frame_103.jpg  frame_199.jpg  frame_293.jpg  frame_388.jpg  frame_482.jpg
frame_104.jpg  frame_2.jpg    frame_294.jpg  frame_389.jpg  frame_483.jpg
frame_105.jpg  frame_20.jpg   frame_295.jpg  frame_39.jpg   frame_484.jpg
frame_106.jpg  frame_200.jpg  frame_296.jpg  frame_390.jpg  frame_485.jpg
frame_107.jpg  frame_201.jpg  frame_297.jpg  frame_391.jpg  frame_486.jpg
frame_108.jpg  frame_202.jpg  frame_298.jpg  frame_392.jpg  frame_487.jpg
frame_109.jpg  frame_203.jpg  frame_299.jpg  frame_393.jpg  frame_488.jpg
frame_11.jpg   frame_204.jpg  frame_3.jpg    frame_394.jpg  frame_489.jpg
frame_110.jpg  frame_205.jpg  frame_30.jpg   frame_395.jpg  frame_49.jpg
frame_111.jpg  frame_206.jpg  frame_300.jpg  frame_396.jpg  frame_490.jpg
frame_112.jpg  frame_207.jpg  frame_301.jpg  frame_397.jpg  frame_491.jpg
frame_113.jpg  frame_208.jpg  frame_302.jpg  frame_398.jpg  frame_492.jpg
frame_114.jpg  frame_209.jpg  frame_303.jpg  frame_399.jpg  frame_493.jpg
frame_115.jpg  frame_21.jpg   frame_304.jpg  frame_4.jpg    frame_494.jpg
frame_116.jpg  frame_210.jpg  frame_305.jpg  frame_40.jpg   frame_495.jpg
frame_117.jpg  frame_211.jpg  frame_306.jpg  frame_400.jpg  frame_496.jpg
frame_118.jpg  frame_212.jpg  frame_307.jpg  frame_401.jpg  frame_497.jpg
frame_119.jpg  frame_213.jpg  frame_308.jpg  frame_402.jpg  frame_498.jpg
frame_12.jpg   frame_214.jpg  frame_309.jpg  frame_403.jpg  frame_499.jpg
frame_120.jpg  frame_215.jpg  frame_31.jpg   frame_404.jpg  frame_5.jpg
frame_121.jpg  frame_216.jpg  frame_310.jpg  frame_405.jpg  frame_50.jpg
frame_122.jpg  frame_217.jpg  frame_311.jpg  frame_406.jpg  frame_500.jpg
frame_123.jpg  frame_218.jpg  frame_312.jpg  frame_407.jpg  frame_501.jpg
frame_124.jpg  frame_219.jpg  frame_313.jpg  frame_408.jpg  frame_502.jpg
```

- frame들이 순차적으로 넘버링 되어 존재하는 모습입니다.

```
> db.hs_requests.find({})
{ "_id" : ObjectId("630dc7894b7e0571d8c5497b"), "FILENAME" : "video_3.mp4", "STATUS" : "INFERENCING", "PROCESS_START" : ISODate("2022-08-30T08:17:13.663Z"), "PREPROCESSING_START" : ISODate("2022-08-30T08:17:13.663Z"), "PREPROCESSING_END" : ISODate("2022-08-30T08:17:27.065Z"), "INFERENCING_START" : ISODate("2022-08-30T08:17:27.388Z"), "INFERENCING_END" : "WAITING..", "POSTPROCESSING_START" : "WAITING..", "POSTPROCESSING_END" : "WAITING..", "PROCESS_END" : "WAITING..", "TOTAL" : "WAITING..." }
> db.hs_requests.find({})
{ "_id" : ObjectId("630dc7894b7e0571d8c5497b"), "FILENAME" : "video_3.mp4", "STATUS" : "INFERENCING_COMPLETE", "PROCESS_START" : ISODate("2022-08-30T08:17:13.663Z"), "PREPROCESSING_START" : ISODate("2022-08-30T08:17:13.663Z"), "PREPROCESSING_END" : ISODate("2022-08-30T08:17:27.065Z"), "INFERENCING_START" : ISODate("2022-08-30T08:17:27.388Z"), "INFERENCING_END" : ISODate("2022-08-30T08:20:26.308Z"), "POSTPROCESSING_START" : "WAITING..", "POSTPROCESSING_END" : "WAITING..", "PROCESS_END" : "WAITING..", "TOTAL" : "WAITING..." }
> db.hs_requests.find({})
{ "_id" : ObjectId("630dc7894b7e0571d8c5497b"), "FILENAME" : "video_3.mp4", "STATUS" : "INFERENCING_COMPLETE", "PROCESS_START" : ISODate("2022-08-30T08:17:13.663Z"), "PREPROCESSING_START" : ISODate("2022-08-30T08:17:13.663Z"), "PREPROCESSING_END" : ISODate("2022-08-30T08:17:27.065Z"), "INFERENCING_START" : ISODate("2022-08-30T08:17:27.388Z"), "INFERENCING_END" : ISODate("2022-08-30T08:20:26.308Z"), "POSTPROCESSING_START" : "WAITING..", "POSTPROCESSING_END" : "WAITING..", "PROCESS_END" : "WAITING..", "TOTAL" : "WAITING..." }
> db.hs_requests.find({})
{ "_id" : ObjectId("630dc7894b7e0571d8c5497b"), "FILENAME" : "video_3.mp4", "STATUS" : "INFERENCING_COMPLETE", "PROCESS_START" : ISODate("2022-08-30T08:17:13.663Z"), "PREPROCESSING_START" : ISODate("2022-08-30T08:17:13.663Z"), "PREPROCESSING_END" : ISODate("2022-08-30T08:17:27.065Z"), "INFERENCING_START" : ISODate("2022-08-30T08:17:27.388Z"), "INFERENCING_END" : ISODate("2022-08-30T08:20:26.308Z"), "POSTPROCESSING_START" : "WAITING..", "POSTPROCESSING_END" : "WAITING..", "PROCESS_END" : "WAITING..", "TOTAL" : "WAITING..." }
> db.hs_requests.find({})
{ "_id" : ObjectId("630dc7894b7e0571d8c5497b"), "FILENAME" : "video_3.mp4", "STATUS" : "POSTPROCESSING", "PROCESS_START" : ISODate("2022-08-30T08:17:13.663Z"), "PREPROCESSING_START" : ISODate("2022-08-30T08:17:13.663Z"), "PREPROCESSING_END" : ISODate("2022-08-30T08:17:27.065Z"), "INFERENCING_START" : ISODate("2022-08-30T08:17:27.388Z"), "INFERENCING_END" : ISODate("2022-08-30T08:20:26.308Z"), "POSTPROCESSING_START" : ISODate("2022-08-30T08:20:27.755Z"), "POSTPROCESSING_END" : "WAITING..", "PROCESS_END" : "WAITING..", "TOTAL" : "WAITING..." }
```

- microservice2에서 Inferencing 작업이 완료되면 STATUS가 INFERENCING_COMPLETE로 바뀌게 되며, 이로 인해 microservice3가 작동하게 되어 STATUS가 POSTPROCESSING으로 바뀌게 됩니다.
- INFERENCING 과정을 완료한 후 결과물은 다음과 같습니다.

```
root@hs-pre-deploy-b77488b9-sgrdn:/mnt/630dc7894b7e0571d8c5497b/txts# ls
infer0.txt    infer135.txt  infer172.txt  infer209.txt  infer246.txt  infer283.txt  infer32.txt   infer357.txt  infer394.txt  infer430.txt  infer468.txt  infer504.txt  infer7.txt
infer1.txt    infer136.txt  infer173.txt  infer21.txt   infer247.txt  infer284.txt  infer320.txt  infer358.txt  infer395.txt  infer431.txt  infer469.txt  infer505.txt  infer70.txt
infer10.txt   infer137.txt  infer174.txt  infer210.txt  infer248.txt  infer285.txt  infer321.txt  infer359.txt  infer396.txt  infer432.txt  infer47.txt   infer506.txt  infer71.txt
infer100.txt  infer138.txt  infer175.txt  infer211.txt  infer249.txt  infer286.txt  infer322.txt  infer36.txt   infer397.txt  infer433.txt  infer470.txt  infer507.txt  infer72.txt
infer101.txt  infer139.txt  infer176.txt  infer212.txt  infer25.txt   infer287.txt  infer323.txt  infer360.txt  infer398.txt  infer434.txt  infer471.txt  infer508.txt  infer73.txt
infer102.txt  infer14.txt   infer177.txt  infer213.txt  infer250.txt  infer288.txt  infer324.txt  infer361.txt  infer399.txt  infer435.txt  infer472.txt  infer509.txt  infer74.txt
infer103.txt  infer140.txt  infer178.txt  infer214.txt  infer251.txt  infer289.txt  infer325.txt  infer362.txt  infer4.txt    infer436.txt  infer473.txt  infer51.txt   infer75.txt
infer104.txt  infer141.txt  infer179.txt  infer215.txt  infer252.txt  infer29.txt   infer326.txt  infer363.txt  infer40.txt   infer437.txt  infer474.txt  infer510.txt  infer76.txt
infer105.txt  infer142.txt  infer18.txt   infer216.txt  infer253.txt  infer290.txt  infer327.txt  infer364.txt  infer400.txt  infer438.txt  infer475.txt  infer511.txt  infer77.txt
infer106.txt  infer143.txt  infer180.txt  infer217.txt  infer254.txt  infer291.txt  infer328.txt  infer365.txt  infer401.txt  infer439.txt  infer476.txt  infer512.txt  infer78.txt
infer107.txt  infer144.txt  infer181.txt  infer218.txt  infer255.txt  infer292.txt  infer329.txt  infer366.txt  infer402.txt  infer44.txt   infer477.txt  infer513.txt  infer79.txt
infer108.txt  infer145.txt  infer182.txt  infer219.txt  infer256.txt  infer293.txt  infer33.txt   infer367.txt  infer403.txt  infer440.txt  infer478.txt  infer514.txt  infer8.txt
infer109.txt  infer146.txt  infer183.txt  infer22.txt   infer257.txt  infer294.txt  infer330.txt  infer368.txt  infer404.txt  infer441.txt  infer479.txt  infer515.txt  infer80.txt
infer11.txt   infer147.txt  infer184.txt  infer220.txt  infer258.txt  infer295.txt  infer331.txt  infer369.txt  infer405.txt  infer442.txt  infer48.txt   infer516.txt  infer81.txt
infer110.txt  infer148.txt  infer185.txt  infer221.txt  infer259.txt  infer296.txt  infer332.txt  infer37.txt   infer406.txt  infer443.txt  infer480.txt  infer517.txt  infer82.txt
infer111.txt  infer149.txt  infer186.txt  infer222.txt  infer26.txt   infer297.txt  infer333.txt  infer370.txt  infer407.txt  infer444.txt  infer481.txt  infer518.txt  infer83.txt
infer112.txt  infer15.txt   infer187.txt  infer223.txt  infer260.txt  infer298.txt  infer334.txt  infer371.txt  infer408.txt  infer445.txt  infer482.txt  infer519.txt  infer84.txt
infer113.txt  infer150.txt  infer188.txt  infer224.txt  infer261.txt  infer299.txt  infer335.txt  infer372.txt  infer409.txt  infer446.txt  infer483.txt  infer52.txt   infer85.txt
infer114.txt  infer151.txt  infer189.txt  infer225.txt  infer262.txt  infer3.txt    infer336.txt  infer373.txt  infer41.txt   infer447.txt  infer484.txt  infer520.txt  infer86.txt
infer115.txt  infer152.txt  infer19.txt   infer226.txt  infer263.txt  infer30.txt   infer337.txt  infer374.txt  infer410.txt  infer448.txt  infer485.txt  infer521.txt  infer87.txt
infer116.txt  infer153.txt  infer190.txt  infer227.txt  infer264.txt  infer300.txt  infer338.txt  infer375.txt  infer411.txt  infer449.txt  infer486.txt  infer522.txt  infer88.txt
infer117.txt  infer154.txt  infer191.txt  infer228.txt  infer265.txt  infer301.txt  infer339.txt  infer376.txt  infer412.txt  infer45.txt   infer487.txt  infer523.txt  infer89.txt
infer118.txt  infer155.txt  infer192.txt  infer229.txt  infer266.txt  infer302.txt  infer34.txt   infer377.txt  infer413.txt  infer450.txt  infer488.txt  infer524.txt  infer9.txt
infer119.txt  infer156.txt  infer193.txt  infer23.txt   infer267.txt  infer303.txt  infer340.txt  infer378.txt  infer414.txt  infer451.txt  infer489.txt  infer53.txt   infer90.txt
infer12.txt   infer157.txt  infer194.txt  infer230.txt  infer268.txt  infer304.txt  infer341.txt  infer379.txt  infer415.txt  infer452.txt  infer49.txt   infer54.txt   infer91.txt
infer120.txt  infer158.txt  infer195.txt  infer231.txt  infer269.txt  infer305.txt  infer342.txt  infer38.txt   infer416.txt  infer453.txt  infer490.txt  infer55.txt   infer92.txt
infer121.txt  infer159.txt  infer196.txt  infer232.txt  infer27.txt   infer306.txt  infer343.txt  infer380.txt  infer417.txt  infer454.txt  infer491.txt  infer56.txt   infer93.txt
infer122.txt  infer16.txt   infer197.txt  infer233.txt  infer270.txt  infer307.txt  infer344.txt  infer381.txt  infer418.txt  infer455.txt  infer492.txt  infer57.txt   infer94.txt
infer123.txt  infer160.txt  infer198.txt  infer234.txt  infer271.txt  infer308.txt  infer345.txt  infer382.txt  infer419.txt  infer456.txt  infer493.txt  infer58.txt   infer95.txt
infer124.txt  infer161.txt  infer199.txt  infer235.txt  infer272.txt  infer309.txt  infer346.txt  infer383.txt  infer42.txt   infer457.txt  infer494.txt  infer59.txt   infer96.txt
infer125.txt  infer162.txt  infer2.txt    infer236.txt  infer273.txt  infer31.txt   infer347.txt  infer384.txt  infer420.txt  infer458.txt  infer495.txt  infer6.txt    infer97.txt
infer126.txt  infer163.txt  infer20.txt   infer237.txt  infer274.txt  infer310.txt  infer348.txt  infer385.txt  infer421.txt  infer459.txt  infer496.txt  infer60.txt   infer98.txt
infer127.txt  infer164.txt  infer200.txt  infer238.txt  infer275.txt  infer311.txt  infer349.txt  infer386.txt  infer422.txt  infer46.txt   infer497.txt  infer61.txt   infer99.txt
infer128.txt  infer165.txt  infer201.txt  infer239.txt  infer276.txt  infer312.txt  infer35.txt   infer387.txt  infer423.txt  infer460.txt  infer498.txt  infer62.txt
infer129.txt  infer166.txt  infer202.txt  infer24.txt   infer277.txt  infer313.txt  infer350.txt  infer388.txt  infer424.txt  infer461.txt  infer499.txt  infer63.txt
infer13.txt   infer167.txt  infer203.txt  infer240.txt  infer278.txt  infer314.txt  infer351.txt  infer389.txt  infer425.txt  infer462.txt  infer5.txt    infer64.txt
infer130.txt  infer168.txt  infer204.txt  infer241.txt  infer279.txt  infer315.txt  infer352.txt  infer39.txt   infer426.txt  infer463.txt  infer50.txt   infer65.txt
infer131.txt  infer169.txt  infer205.txt  infer242.txt  infer28.txt   infer316.txt  infer353.txt  infer390.txt  infer427.txt  infer464.txt  infer500.txt  infer66.txt
infer132.txt  infer17.txt   infer206.txt  infer243.txt  infer280.txt  infer317.txt  infer354.txt  infer391.txt  infer428.txt  infer465.txt  infer501.txt  infer67.txt
infer133.txt  infer170.txt  infer207.txt  infer244.txt  infer281.txt  infer318.txt  infer355.txt  infer392.txt  infer429.txt  infer466.txt  infer502.txt  infer68.txt
infer134.txt  infer171.txt  infer208.txt  infer245.txt  infer282.txt  infer319.txt  infer356.txt  infer393.txt  infer43.txt   infer467.txt  infer503.txt  infer69.txt
```

- txt들이 순차적으로 넘버링 되어 존재하는 모습입니다.
- txt들이 어떤 정보들을 가지고 있는지 한 개만 확인해 보겠습니다.

```
root@hs-pre-deploy-b77488b9-sgrdn:/mnt/630dc7894b7e0571d8c5497b/txts# cat infer232.txt
0.58806956
0.4054076
0.8640084
0.6594328
2
0.17521656
0.3989436
0.22637817
0.4794193
2
0.49387497
0.032108888
0.9473652
0.31782967
6
0.15249959
0.39721596
0.19364426
0.4815235
0
```

- 해당 프레임에서 탐지된 객체들에 대한 정보로, 직사각형을 그리는 데 있어 minX,minY,maxX,maxY 그리고 COCO Dataset 기반 label의 idx가 출력되는 것을 확인 할 수 있습니다.

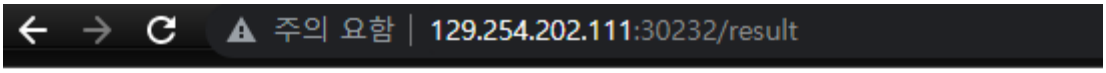
```
> db.hs_requests.find({})
{ "_id" : ObjectId("630dc7894b7e0571d8c5497b"), "FILENAME" : "video_3.mp4", "STATUS" : "POSTPROCESSING", "PROCESS_START" : ISODate("2022-08-30T08:17:13.663Z"), "PREPROCESSING_START" : ISODate("2022-08-30T08:17:13.663Z"), "PREPROCESSING_END" : ISODate("2022-08-30T08:17:27.065Z"), "INFERENCING_START" : ISODate("2022-08-30T08:17:27.388Z"), "INFERENCING_END" : ISODate("2022-08-30T08:20:26.308Z"), "POSTPROCESSING_START" : ISODate("2022-08-30T08:20:27.755Z"), "POSTPROCESSING_END" : "WAITING..", "PROCESS_END" : "WAITING..", "TOTAL" : "WAITING..." }
> db.hs_requests.find({})
{ "_id" : ObjectId("630dc7894b7e0571d8c5497b"), "FILENAME" : "video_3.mp4", "STATUS" : "POSTPROCESSING_COMPLETE", "PROCESS_START" : ISODate("2022-08-30T08:17:13.663Z"), "PREPROCESSING_START" : ISODate("2022-08-30T08:17:13.663Z"), "PREPROCESSING_END" : ISODate("2022-08-30T08:17:27.065Z"), "INFERENCING_START" : ISODate("2022-08-30T08:17:27.388Z"), "INFERENCING_END" : ISODate("2022-08-30T08:20:26.308Z"), "POSTPROCESSING_START" : ISODate("2022-08-30T08:20:27.755Z"), "POSTPROCESSING_END" : ISODate("2022-08-30T08:20:52.576Z"), "PROCESS_END" : ISODate("2022-08-30T08:20:52.578Z"), "TOTAL" : "0:03:38.918805" }
```

- microservice3에서 PostProcessing 작업이 완료되면 STATUS가 POSTPROCESSING_COMPLETE로 바뀌게 되며, 총 소요된 시간이 TOTAL에 나타나게 됩니다.
- POSTPROCESSING의 결과물은 src_frame에 txt가 합쳐진 tar_frame(tar_{n}.jpg)과 tar_frame을 연속적으로 이어 만든 avi(target_{R_id}.avi) 입니다.
- POSTPROCESSING 결과물들의 일부를 확인해 보겠습니다.

```
root@hs-pre-deploy-b77488b9-sgrdn:/mnt/630dc7894b7e0571d8c5497b/tar_frame# ls
tar_0.jpg    tar_151.jpg  tar_204.jpg  tar_258.jpg  tar_310.jpg  tar_364.jpg  tar_417.jpg  tar_470.jpg  tar_523.jpg
tar_1.jpg    tar_152.jpg  tar_205.jpg  tar_259.jpg  tar_311.jpg  tar_365.jpg  tar_418.jpg  tar_471.jpg  tar_524.jpg
tar_10.jpg   tar_153.jpg  tar_206.jpg  tar_26.jpg   tar_312.jpg  tar_366.jpg  tar_419.jpg  tar_472.jpg  tar_53.jpg
tar_100.jpg  tar_154.jpg  tar_207.jpg  tar_260.jpg  tar_313.jpg  tar_367.jpg  tar_42.jpg   tar_473.jpg  tar_54.jpg
tar_101.jpg  tar_155.jpg  tar_208.jpg  tar_261.jpg  tar_314.jpg  tar_368.jpg  tar_420.jpg  tar_474.jpg  tar_55.jpg
tar_102.jpg  tar_156.jpg  tar_209.jpg  tar_262.jpg  tar_315.jpg  tar_369.jpg  tar_421.jpg  tar_475.jpg  tar_56.jpg
tar_103.jpg  tar_157.jpg  tar_21.jpg   tar_263.jpg  tar_316.jpg  tar_37.jpg   tar_422.jpg  tar_476.jpg  tar_57.jpg
tar_104.jpg  tar_158.jpg  tar_210.jpg  tar_264.jpg  tar_317.jpg  tar_370.jpg  tar_423.jpg  tar_477.jpg  tar_58.jpg
tar_105.jpg  tar_159.jpg  tar_211.jpg  tar_265.jpg  tar_318.jpg  tar_371.jpg  tar_424.jpg  tar_478.jpg  tar_59.jpg
tar_106.jpg  tar_16.jpg   tar_212.jpg  tar_266.jpg  tar_319.jpg  tar_372.jpg  tar_425.jpg  tar_479.jpg  tar_6.jpg
tar_107.jpg  tar_160.jpg  tar_213.jpg  tar_267.jpg  tar_32.jpg   tar_373.jpg  tar_426.jpg  tar_48.jpg   tar_60.jpg
tar_108.jpg  tar_161.jpg  tar_214.jpg  tar_268.jpg  tar_320.jpg  tar_374.jpg  tar_427.jpg  tar_480.jpg  tar_61.jpg
tar_109.jpg  tar_162.jpg  tar_215.jpg  tar_269.jpg  tar_321.jpg  tar_375.jpg  tar_428.jpg  tar_481.jpg  tar_62.jpg
tar_11.jpg   tar_163.jpg  tar_216.jpg  tar_27.jpg   tar_322.jpg  tar_376.jpg  tar_429.jpg  tar_482.jpg  tar_63.jpg
```

```
root@hs-pre-deploy-b77488b9-sgrdn:/mnt/630dc7894b7e0571d8c5497b/target# ls
target_630dc7894b7e0571d8c5497b.avi
```

- tar_frame들과 target avi가 생성된 것을 확인할 수 있습니다.

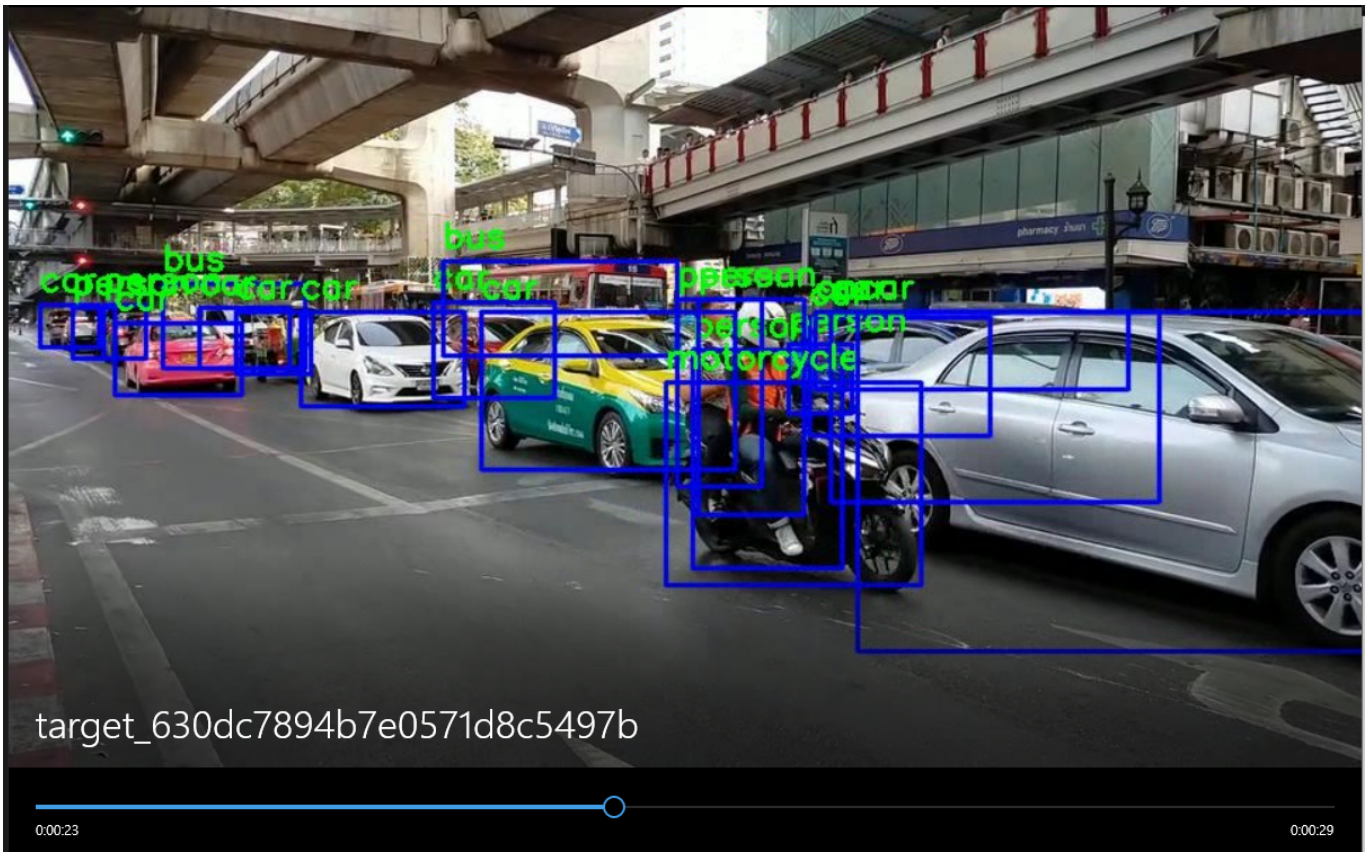


back to home refresh

your id is : **630dc7894b7e0571d8c5497b**,
your filename is : **video_3.mp4**,
process start time : **2022-08-30 08:17:13.663000**,
preprocessing start time : **2022-08-30 08:17:13.663000**,
preprocessing end time : **2022-08-30 08:17:27.065000**,
inferencing start time : **2022-08-30 08:17:27.388000**,
inferencing end time : **2022-08-30 08:20:26.308000**,
postprocessing start time : **2022-08-30 08:20:27.755000**,
postprocessing end time : **2022-08-30 08:20:52.576000**,
total duration time : **0:03:38.918805**,
video_3.avi

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

- 다운로드 받은 avi 파일을 엽니다.



- Box 형태로 Object Detection을 수행해 해당 객체를 표시한 모습입니다.

Python 으로 Kubernetes Api 활용

- Python 코드에서 현재 실행되고 있는 deployment의 Replicaset을 조정하는 방법입니다.
- ServiceAccount가 생성 되어 있고 자격 증명을 받았으며, clusterrole, clusterrolebinding이 생성 되어있는 상태여야 합니다.
- microservice 1 혹은 3 의 pod 안쪽으로 접근하여 (kubectl exec 사용) micro{n}_python_k8s_api.py파일을 살펴봅니다.
- 다음은 해당 ServiceAccount의 자격 증명을 받아오는 Python 코드입니다.

```
config = client.Configuration()

config.api_key['authorization'] = open('/var/run/secrets/kubernetes.io/serviceaccount/token').read()
config.api_key_prefix['authorization'] = 'Bearer'
config.host = 'https://kubernetes.default'
config.ssl_ca_cert = '/var/run/secrets/kubernetes.io/serviceaccount/ca.crt'
config.verify_ssl=True
```

- 다음은 API Instance를 생성해 Deployment의 Replicaset을 조정하는 코드입니다.

```
with kubernetes.client.ApiClient(config) as api_client:
    # Create an instance of the API class
    api_instance = kubernetes.client.AppsV1Api(api_client)
    name = 'hs-pre-deploy' # str | name of the deployment

namespace='default'

api_response = api_instance.patch_namespaced_deployment_scale(
   name, namespace,
   [{'op': 'replace', 'path': '/spec/replicas', 'value': 1}])
```

- 현재 value값이 1로 설정되어 있고

```
NAMESPACE     NAME                                READY   UP-TO-DATE   AVAILABLE
default       deployment.apps/hs-avi-to-mp4       1/1     1            1
default       deployment.apps/hs-mp4-to-mp3       1/1     1            1
default       deployment.apps/hs-post-deploy      1/1     1            1
default       deployment.apps/hs-pre-deploy       1/1     1            1
```

- Type이 Deployment인 hs-pre-deploy 가 1/1인 것을 확인할 수 있습니다.

```
api_response = api_instance.patch_namespaced_deployment_scale(
   name, namespace,
   [{'op': 'replace', 'path': '/spec/replicas', 'value': 3}])
```

- 이제 Replicaset의 value값을 3으로 조정하고 실행해 보겠습니다.

```
NAMESPACE     NAME                                READY   UP-TO-DATE   AVAILABLE
default       deployment.apps/hs-avi-to-mp4       1/1     1            1
default       deployment.apps/hs-mp4-to-mp3       1/1     1            1
default       deployment.apps/hs-post-deploy      1/1     1            1
default       deployment.apps/hs-pre-deploy       3/3     3            3
```

- Type이 Deployment인 hs-pre-deploy가 3/3이 된 것을 확인할 수 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 및 PreProcessing)를 실행하는 yaml파일에 nodeSelector를 통해 해당 Pod는 scale에 관계없이 오직 한 노드에만 할당되도록 하였습니다. 그 다음, microservice 2 혹은 3에서 각각의 작업이 시작되거나 끝났을 때,

  (microservice 3를 예로 들었습니다.)

```
R_id=doc["_id"]
hs_resnet.hs_requests.update_one({"_id":R_id},{"$set":{"STATUS":"POSTPROCESSING"}})

firsturi=os.path.join("http://129.254.202.111:30232","requestid",str(R_id),"status","POSTPROCESSING")
print(firsturi)
requests.put(firsturi)
```

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

```
@app.route('/requestid/<request_id>/status/<status>',methods=['PUT','GET','POST'])
def update_timeline(request_id, status):
    print("rid: ", request_id, "status: ",status)
    if status=="INFERENCING":
        hs_resnet.hs_requests.update_one({"_id":ObjectId(request_id)},{"$set":{"INFERENCING_START":datetime.datetime.now()}})
    elif status=="INFERENCING_COMPLETE":
        hs_resnet.hs_requests.update_one({"_id":ObjectId(request_id)},{"$set":{"INFERENCING_END":datetime.datetime.now()}})
    elif status=="POSTPROCESSING":
        hs_resnet.hs_requests.update_one({"_id":ObjectId(request_id)},{"$set":{"POSTPROCESSING_START":datetime.datetime.now()}})
```

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