CVAT 설치 및 데이터 연동 인수인계

작성일: 2025-12-12

대상: Ubuntu 환경(프로젝트 루트: healtheat_vision__)에서 Docker로 CVAT를 운영하고, YOLO 자동 라벨링 결과를 CVAT로 가져와 클래스/박스를 검수한 뒤, 원래의 카테고리 ID 사양(COCO ID / YOLO ID)을 복원하여 재학습/제출(Kaggle 등)에 활용하는 전체 절차.

문서 범위

1. Docker 설치
2. CVAT 설치 및 흔한 오류/해결
3. CVAT 라벨 생성(정규화 포함) + 코드
4. CVAT import용 COCO 생성(변환/병합) + 코드
5. CVAT export 후 원래 COCO ID / YOLO ID 복원(제출 사양 복구) + 코드

1. Docker 설치

CVAT은 다수의 서비스(PostgreSQL, Redis, ClickHouse, Traefik, CVAT server/UI 등)를 컨테이너로 띄우므로 Ubuntu에 Docker 엔진과 Compose를 먼저 준비해야 합니다.

1.1 설치 방식 선택

권장: Docker 공식 APT 저장소를 통해 docker-ce + docker compose plugin(v2)을 설치합니다. 배포판 기본 패키지(docker.io)는 버전/compose 지원이 부족해 CVAT docker-compose.yml과 충돌할 때가 있습니다.

1.2 빠른 설치(배포판 패키지) - 최소 구성

환경 제약으로 공식 저장소를 쓰기 어렵다면 아래 방식으로도 동작은 합니다. 단, compose 관련 이슈가 더 잦습니다.

```
# (Ubuntu) 배포판 패키지 설치
sudo apt update
sudo apt install -y docker.io

# 도커 데몬 시작/자동시작
sudo systemctl enable --now docker

# 동작 확인
docker --version
sudo docker ps
```

1.3 권장 설치(공식 저장소) - compose v2 포함

아래 설치는 docker compose(v2)까지 포함되어 CVAT 설치 시 오류를 줄입니다.

```
# 1) 의존 패키지
sudo apt update
sudo apt install -y ca-certificates curl gnupg

# 2) GPG 키 등록
sudo install -m 0755 -d /etc/apt/keyrings
curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo gpg --dearmor -o /etc/apt/keyrings/docker.gpg
sudo chmod a+r /etc/apt/keyrings/docker.gpg

# 3) 저장소 등록 (Ubuntu codename 자동 사용)
echo \
  "deb [arch=$(dpkg --print-architecture) signed-by=/etc/apt/keyrings/docker.gpg] \
  https://download.docker.com/linux/ubuntu \
  $(. /etc/os-release && echo \"$VERSION_CODENAME\") stable" | \
  sudo tee /etc/apt/sources.list.d/docker.list > /dev/null

# 4) 설치
sudo apt update
sudo apt install -y docker-ce docker-ce-cli containerd.io docker-buildx-plugin docker-compose-plugin

# 5) 확인
docker --version
docker compose version
```

1.4 권한/세션 관련 주의

도커 데몬 접속 오류 예: Cannot connect to the Docker daemon at unix:///var/run/docker.sock

해결 체크리스트:

- 도커 데몬 상태 확인: sudo systemctl status docker
- 미실행이면 시작: sudo systemctl start docker
- root가 아닌 사용자로 실행 시: sudo usermod -aG docker $USER 후 로그아웃/재로그인

2. CVAT 설치 및 문제 해결

CVAT는 공식적으로 Docker Compose 기반 설치를 제공합니다. 프로젝트 내에 cvat/ 디렉토리를 두고 해당 위치에서 compose를 올리는 방식으로 운영했습니다.

2.1 설치 준비

```
# (예시) 프로젝트 루트
cd ~/mission/proj-1/healtheat_vision__/cvat

# 컨테이너 기동 (백그라운드)
docker compose up -d
# 또는 legacy compose만 있는 환경:
docker-compose up -d
```

2.2 정상 기동 확인

```
# 컨테이너 상태 확인
docker ps

# 로그 확인 (문제 발생 시)
docker compose logs -f --tail=200 cvat_server
docker compose logs -f --tail=200 traefik
```

정상이라면 traefik 컨테이너가 0.0.0.0:8080, 8090 등의 포트를 바인딩하고, cvat_server / cvat_ui / db(redis, postgres 등)가 Up 상태로 표시됩니다.

2.3 자주 발생한 오류와 해결

(A) docker compose 명령이 없음: docker: unknown command: docker compose

- 환경이 docker.io(배포판 패키지)만 설치된 경우 compose plugin이 빠져 있을 수 있습니다.
- 해결 1: 공식 저장소 설치로 docker-compose-plugin 추가(1.3 참고)
- 해결 2: 임시로 legacy docker-compose 사용(패키지 docker-compose 또는 pip 설치)

(B) Docker 데몬 미실행: Cannot connect to the Docker daemon at unix:///var/run/docker.sock

```
sudo systemctl status docker
sudo systemctl start docker
# 권한 문제면 docker 그룹 추가 후 로그아웃/로그인
sudo usermod -aG docker $USER
```

(C) docker-compose.yml 버전/스키마 불일치

에러 예: Compose file is invalid... 'name' does not match '^x-' 또는 environment ... contains false, invalid type

- 원인: 매우 오래된 docker-compose(v1) 또는 구버전 파서가 최신 Compose 스펙을 제대로 처리하지 못함.
- 해결: docker compose v2(플러그인)로 실행하거나, CVAT 레포/compose 파일과 도커/컴포즈 버전을 맞춤.
- 추가 팁: YAML의 boolean(false/true)을 문자열("false")로 바꾸면 통과하는 경우가 있으나, 근본 해결은 compose 업데이트.

(D) 초기 계정

CVAT는 배포 구성에 따라 초기 계정이 없을 수 있습니다. 로그인 불가 시 server 컨테이너에서 슈퍼유저를 생성합니다.

```
# 컨테이너 이름은 환경에 따라 다를 수 있음(cvat_server 등)
docker exec -it cvat_server bash
python manage.py createsuperuser
exit
```

2.4 데이터 업로드 방식

CVAT Task는 기본적으로 웹 UI에서 이미지 업로드 후, 별도로 어노테이션(import) 파일을 업로드하는 흐름입니다. 대용량 데이터는 호스트 디렉토리를 CVAT 컨테이너에 마운트하는 방식도 가능하나, 팀 인수인계 관점에서는 UI 업로드 + COCO import를 표준으로 두는 편이 재현성이 좋습니다.

3. CVAT용 라벨 생성(정규화 포함)

핵심 결론: 라벨 문자열 정규화(normalize)가 최우선(0번 단계)입니다. CVAT은 라벨 매칭을 문자열 완전 일치로 처리하므로, 보이지 않는 공백(NBSP, ₩xa0)이나 앞뒤 공백이 섞이면 Import가 즉시 실패합니다.

3.1 정규화가 필요한 이유(실제 증상)

실제 에러 예: Label '...₩xa0' is not registered for this task (₩xa0 = Non Breaking Space). 사람 눈에는 같아 보이지만 시스템 비교에서는 다른 문자열입니다.

3.2 라벨 정규화 스크립트

입력: raw_processed/train_annotations/category_mapping.json

출력: raw_processed/train_annotations/category_mapping.normalized.json

```
# src/utils/normalize_labels.py
import json
import re
from pathlib import Path
from src.config import Config

IN_PATH = Config.DATA_DIR / "raw_processed/train_annotations/category_mapping.json"
OUT_PATH = Config.DATA_DIR / "raw_processed/train_annotations/category_mapping.normalized.json"

def normalize(text: str) -> str:
    # 1) NBSP 제거/치환
    text = text.replace("\xa0", " ")
    # 2) 연속 공백 정리
    text = re.sub(r"\s+", " ", text)
    # 3) 앞뒤 공백 제거
    return text.strip()

mapping = json.loads(IN_PATH.read_text(encoding="utf-8"))
for _, v in mapping.items():
    v["name"] = normalize(v["name"])

OUT_PATH.write_text(json.dumps(mapping, indent=2, ensure_ascii=False), encoding="utf-8")
print(f"[DONE] normalized mapping saved -> {OUT_PATH}")
```

3.3 CVAT Labels 입력 포맷(웹 UI - Raw)

CVAT Task 생성 시 Labels 입력란은 JSON 배열을 기대합니다.

최소 포맷(권장):

```
[
  { "name": "보령부스파정 5mg", "attributes": [] },
  { "name": "뮤테란캡슐 100mg", "attributes": [] }
]
```

만약 CVAT 버전/빌드에 따라 unknown label type "undefined"가 발생하면, 라벨 객체에 아래처럼 type을 명시하는 방식으로 우회할 수 있습니다(주로 구형 UI/스키마에서 관측).

```
[
  { "name": "보령부스파정 5mg", "attributes": [], "type": "rectangle" }
]
```

3.4 라벨 JSON 자동 생성 스크립트

정규화된 mapping 파일을 읽어 CVAT Raw Labels JSON을 생성합니다.

```
# src/utils/cvat_labels_gen.py
import json
from src.config import Config

MAPPING = Config.DATA_DIR / "raw_processed/train_annotations/category_mapping.normalized.json"
OUT = Config.DATA_DIR / "cvat_import/labels.json"

mapping = json.loads(MAPPING.read_text(encoding="utf-8"))

labels = []
# mapping의 key(=YOLO id) 순서를 유지하거나, coco_id 기준 정렬 중 선택 가능
items = sorted(mapping.values(), key=lambda x: int(x["coco_id"]))  # coco_id 기준 정렬
for v in items:
    labels.append({"name": v["name"], "attributes": []})

OUT.parent.mkdir(parents=True, exist_ok=True)
OUT.write_text(json.dumps(labels, ensure_ascii=False, indent=2), encoding="utf-8")
print(f"[DONE] labels.json -> {OUT}")
```

사용법:

```
uv run -m src.utils.normalize_labels
uv run -m src.utils.cvat_labels_gen

# labels.json 내용을 CVAT Task 생성 화면 Labels(Raw)에 그대로 붙여넣기
```

4. CVAT용 COCO 어노테이션 생성(코드 포함)

현재 데이터는 이미지당 JSON 1개(per-image COCO) 형태로 저장되어 있습니다 (예: unified_dataset_v2/annotations/*.json). CVAT의 COCO import는 일반적으로 dataset-level COCO 1개 파일(coco_instances.json)을 기대하므로, per-image JSON을 변환/병합하는 단계가 필요합니다.

4.1 전제 조건(중요)

- Task에 업로드된 이미지 파일명과 COCO의 images[].file_name이 완전 동일해야 함(경로/대소문자/확장자 포함).
- Task에 등록된 Labels(name)과 COCO categories[].name이 완전 동일해야 함(₩xa0/공백 포함).
- images[].id는 유일해야 하고, annotations[].image_id는 images[].id를 참조해야 함.

4.2 (권장) CVAT 내부용 category_id 재매핑

권장: CVAT에 import할 때는 category_id를 0..N-1처럼 연속 ID(cvat_id)로 두고, 검수 후 export 결과를 다시 원래 COCO ID로 복원합니다.

4.3 mapping 파일 사양

category_mapping.normalized.json에 cvat_id가 없으면, coco_id 기준 정렬로 자동 부여합니다.

4.4 per-image COCO -> CVAT import용 per-image COCO 변환 스크립트

```
# src/utils/cvat_coco_gen.py
"""
per-image COCO JSON -> CVAT import용 per-image COCO 변환
- annotation.category_id: coco_id -> cvat_id
- categories.name: normalized mapping의 name 사용(문자열 불일치 방지)
- category_mapping.normalized.json 기반으로 cvat_id를 자동 부여 가능
"""

import json
from src.config import Config

IN_DIR = Config.DATA_DIR / "unified_dataset_v2/annotations"
OUT_DIR = Config.DATA_DIR / "cvat_import/annotations"
MAP_PATH = Config.DATA_DIR / "raw_processed/train_annotations/category_mapping.normalized.json"

OUT_DIR.mkdir(parents=True, exist_ok=True)

mapping = json.loads(MAP_PATH.read_text(encoding="utf-8"))

# (1) cvat_id 자동 부여(없을 때): coco_id 기준 정렬 후 0..N-1
sorted_items = sorted(mapping.items(), key=lambda kv: int(kv[1]["coco_id"]))
for new_cvat_id, (k, v) in enumerate(sorted_items):
    v.setdefault("yolo_id", int(k))        # 원래 YOLO class index
    v["cvat_id"] = int(v.get("cvat_id", new_cvat_id))

# (2) 변환용 lookup
coco_to_cvat = {int(v["coco_id"]): int(v["cvat_id"]) for v in mapping.values()}
cvat_to_name = {int(v["cvat_id"]): v["name"] for v in mapping.values()}

for json_path in sorted(IN_DIR.glob("*.json")):
    if json_path.name == "category_mapping.json":
        continue
```

```
    coco = json.loads(json_path.read_text(encoding="utf-8"))

    # 방어: per-image COCO 최소 키 확인
    if "images" not in coco or "annotations" not in coco:
        continue

    # annotations 변환
    for ann in coco.get("annotations", []):
        coco_id = int(ann["category_id"])
        ann["category_id"] = coco_to_cvat[coco_id]

    # categories 재작성(이 이미지에 등장한 cvat_id만)
    used_cvat_ids = sorted({int(a["category_id"]) for a in coco.get("annotations", [])})
    coco["categories"] = [
        {"id": cid, "name": cvat_to_name[cid], "supercategory": "pill"}
        for cid in used_cvat_ids
    ]

    out_path = OUT_DIR / json_path.name
    out_path.write_text(json.dumps(coco, indent=2, ensure_ascii=False), encoding="utf-8")

# mapping 업데이트 저장(옵션)
MAP_PATH.write_text(json.dumps(mapping, indent=2, ensure_ascii=False), encoding="utf-8")
print("[DONE] cvat_import per-image COCO 생성 완료")
```

4.5 per-image COCO 병합 -> coco_instances.json 생성 스크립트

```
# src/utils/cvat_merge.py
"""
per-image COCO -> dataset-level COCO(coco_instances.json)
- images.id / annotations.id / annotations.image_id 재부여
- COCO 키가 없는 파일은 skip + 목록 출력
"""

import json
from src.config import Config

IN_DIR = Config.DATA_DIR / "cvat_import/annotations"
OUT_FILE = Config.DATA_DIR / "cvat_import/coco_instances.json"

images = []
annotations = []
categories = {}

next_img_id = 1
next_ann_id = 1
skipped = []

for json_path in sorted(IN_DIR.glob("*.json")):
    coco = json.loads(json_path.read_text(encoding="utf-8"))

    if "images" not in coco or "annotations" not in coco or not coco.get("images"):
        skipped.append(json_path.name)
        continue

    img = coco["images"][0]
    img["id"] = next_img_id
    images.append(img)

    for c in coco.get("categories", []):
```

```
            categories[int(c["id"])] = c

        for ann in coco.get("annotations", []):
            ann["id"] = next_ann_id
            ann["image_id"] = next_img_id
            annotations.append(ann)
            next_ann_id += 1

        next_img_id += 1

merged = {
    "images": images,
    "annotations": annotations,
    "categories": sorted(categories.values(), key=lambda x: int(x["id"])),
}

OUT_FILE.parent.mkdir(parents=True, exist_ok=True)
OUT_FILE.write_text(json.dumps(merged, indent=2, ensure_ascii=False), encoding="utf-8")

print(f"[DONE] {OUT_FILE}")
print(f"images={len(images)}, annotations={len(annotations)}")
if skipped:
    print("[SKIPPED]")
    for s in skipped:
        print(" -", s)
```

4.6 실행 순서

```
uv run -m src.utils.normalize_labels
uv run -m src.utils.cvat_labels_gen
uv run -m src.utils.cvat_coco_gen
uv run -m src.utils.cvat_merge
```

4.7 Import 실패 시 1차 점검 포인트

- 에러: 'Label ... is not registered' -> Task Labels와 COCO categories.name 불일치(₩xa0/공백 포함).
- 에러: 'Failed to import dataset coco_instances' -> COCO 구조 이상(필수 키 누락, image_id 참조 불일치, 파일명 불일치).
- 에러: per-image JSON에 images 키 없음(KeyError) -> 병합 대상 디렉토리에 mapping.json 등 비COCO 파일이 섞임. skip 처리 필요.

5. Export 후 원래 카테고리 ID / YOLO ID 복구(제출/재학습 사양 복원)

CVAT에서 검수 완료 후 Export를 수행하면, 기본적으로 Task Labels에 등록된 ID(cvat_id) 체계로 category_id가 기록될 수 있습니다. 따라서 원래 사양인 COCO ID(예: 1899, 2482 ...) 또는 YOLO class index(0..N-1)로 복원하는 후처리가 필요합니다.

5.1 원칙(데이터 레이어를 분리)

- CVAT 내부: cvat_id(연속 ID) + name(정규화된 문자열)로 운영
- 학습/제출: coco_id(원래 COCO 카테고리 ID) 또는 yolo_id(클래스 인덱스)로 복원
- 항상 mapping 파일(category_mapping.normalized.json)을 단일 진실 소스로 유지

5.2 CVAT Export COCO -> coco_id 복원 스크립트

입력: cvat_export/coco_instances.json (category_id=cvat_id)

출력: restored_dataset/coco_instances.json (category_id=coco_id)

```
# src/utils/restore_coco_ids.py
"""
CVAT export COCO -> 원래 coco_id 복원
- annotation.category_id: cvat_id -> coco_id
- categories.id/name 재작성
"""

import json
from src.config import Config

IN_PATH = Config.DATA_DIR / "cvat_export/coco_instances.json"
OUT_PATH = Config.DATA_DIR / "restored_dataset/coco_instances.json"
MAP_PATH = Config.DATA_DIR / "raw_processed/train_annotations/category_mapping.normalized.json"

OUT_PATH.parent.mkdir(parents=True, exist_ok=True)

mapping = json.loads(MAP_PATH.read_text(encoding="utf-8"))

# cvat_id -> coco_id, name
cvat_to_coco = {int(v["cvat_id"]): int(v["coco_id"]) for v in mapping.values()}
coco_to_name = {int(v["coco_id"]): v["name"] for v in mapping.values()}

coco = json.loads(IN_PATH.read_text(encoding="utf-8"))

for ann in coco.get("annotations", []):
    ann["category_id"] = cvat_to_coco[int(ann["category_id"])]

used_coco_ids = sorted({int(a["category_id"]) for a in coco.get("annotations", [])})
coco["categories"] = [
    {"id": cid, "name": coco_to_name[cid], "supercategory": "pill"}
    for cid in used_coco_ids
]

OUT_PATH.write_text(json.dumps(coco, indent=2, ensure_ascii=False), encoding="utf-8")
print(f"[DONE] restored coco_id -> {OUT_PATH}")
```

5.3 CVAT Export COCO -> YOLO txt(label) 생성(제출/재학습)

YOLO 포맷은 이미지당 1개 txt가 필요하며, 각 줄은: class x_center y_center width height (모두 0~1 정규화)입니다.

입력: restored_dataset/coco_instances.json (category_id=coco_id) 또는 CVAT export(복원 전)도 가능

출력: restored_dataset/labels_yolo/*.txt (category_id=yolo_id)

```
# src/utils/coco_to_yolo.py
"""
COCO instances -> YOLO txt 라벨 생성
- category_id: coco_id -> yolo_id
- bbox: COCO xywh(px) -> YOLO xcycwh(norm)
"""

import json
from pathlib import Path
from src.config import Config

IN_PATH = Config.DATA_DIR / "restored_dataset/coco_instances.json"
OUT_DIR = Config.DATA_DIR / "restored_dataset/labels_yolo"
MAP_PATH = Config.DATA_DIR / "raw_processed/train_annotations/category_mapping.normalized.json"

OUT_DIR.mkdir(parents=True, exist_ok=True)

mapping = json.loads(MAP_PATH.read_text(encoding="utf-8"))
coco_to_yolo = {int(v["coco_id"]): int(v["yolo_id"]) for v in mapping.values()}

coco = json.loads(IN_PATH.read_text(encoding="utf-8"))

# image_id -> (file_name, w, h)
img_index = {int(img["id"]): img for img in coco.get("images", [])}

# 파일별 라인 누적
lines_by_file = {}

for ann in coco.get("annotations", []):
    img = img_index[int(ann["image_id"])]
    w, h = float(img["width"]), float(img["height"])
    x, y, bw, bh = ann["bbox"]  # px

    # YOLO normalized
    xc = (x + bw / 2.0) / w
    yc = (y + bh / 2.0) / h
    nw = bw / w
    nh = bh / h

    cls = coco_to_yolo[int(ann["category_id"])]
    fname = Path(img["file_name"]).with_suffix(".txt").name
    lines_by_file.setdefault(fname, []).append(f"{cls} {xc:.6f} {yc:.6f} {nw:.6f} {nh:.6f}")

# 저장
for fname, lines in lines_by_file.items():
    (OUT_DIR / fname).write_text("\n".join(lines) + "\n", encoding="utf-8")

print(f"[DONE] YOLO labels -> {OUT_DIR} ({len(lines_by_file)} files)")
```

5.4 Kaggle/외부 제출 사양 복구 체크리스트

- 제출 포맷이 COCO면: categories.id와 annotation.category_id가 요구 사양(연속/원래ID)인지 확인
- 제출 포맷이 YOLO면: 이미지 확장자와 동일 stem의 txt가 존재하는지 확인
- 라벨 문자열(name)은 제출에 직접 사용되지 않아도, 사후 추적을 위해 mapping 파일을 보존

부록 A. 운영 팁(선택)

Windows에서 내려받은 파일에 Zone.Identifier가 다량 생성되어 관리가 어려운 경우, Windows 측에서는 PowerShell의 Unblock-File로 일괄 제거할 수 있습니다. WSL/Ubuntu에서 보이는 'Zone.Identifier'는 대개 Windows의 ADS/메타데이터가 파일로 나타난 경우로, 데이터셋 업로드 전 정리가 권장됩니다.

부록 B. 전체 실행 플로우(원클릭 요약)

```
# 0) (한 번만) Docker/CVAT 기동
cd ~/mission/proj-1/healtheat_vision__/cvat
docker compose up -d

# 1) 라벨/매핑 정규화 + 라벨 JSON 생성
uv run -m src.utils.normalize_labels
uv run -m src.utils.cvat_labels_gen

# 2) per-image COCO -> CVAT import용 변환/병합
uv run -m src.utils.cvat_coco_gen
uv run -m src.utils.cvat_merge

# 3) CVAT UI에서 Task 생성(이미지 업로드 + labels.json 입력) 후 coco_instances.json import

# 4) 검수 후 Export(COCO Instances) -> cvat_export/coco_instances.json 저장

# 5) 원래 사양 복구
uv run -m src.utils.restore_coco_ids
uv run -m src.utils.coco_to_yolo
```