**Ubuntu 개발 환경 정리 문서 (KRASE 프로젝트)**

**1. 노트북 초기 세팅**

* **노트북 환경**: x86 기반 및 ARM 기반 노트북 모두 가능
* **운영체제 설치**: Ubuntu 24.04.2 LTS Desktop 버전 설치 완료
* **설치 방식**: 공식 홈페이지에서 ISO 파일 다운로드 → 부팅 USB 제작 (Rufus 등 사용) → 설치

**2. Docker 설치 및 개발 환경 준비**

**2-1. Docker 설치 명령어**

sudo apt update

sudo apt install -y docker.io

sudo systemctl enable docker

sudo systemctl start docker

**2-2. Docker Compose 설치 (선택)**

sudo apt install -y docker-compose

**2-3. 일반 사용자 권한으로 Docker 사용 (Optional)**

sudo usermod -aG docker $USER

# 이후 로그아웃 & 재로그인 필요

**3. 개발 방식 선택: Docker 컨테이너 vs Ubuntu 가상환경**

|  |  |  |
| --- | --- | --- |
| **항목** | **Docker 컨테이너 환경** | **Ubuntu 가상환경 (로컬 개발)** |
| 격리성 | 강함 (OS 수준 격리) | 약함 (동일 OS 내 구성 공유) |
| 구성 속도 | 빠름 (이미지 기반) | 느림 (직접 패키지 설치 필요) |
| 재현성 | 우수 (Dockerfile로 자동화) | 낮음 (개발자별 환경 차이 발생) |
| 운영과 개발의 일관성 | 높음 | 낮음 |
| 리소스 사용량 | 낮음 (가상머신보다 가볍다) | 중간 |
| 디버깅/직접 접근 | 어려울 수 있음 (컨테이너 진입 필요) | 직접 접근 쉬움 |
| 권장 용도 | 배포, 협업 개발, CI/CD | 초기 테스트, 교육, 직접 실습 |

**✅ 결론**

* **Docker 컨테이너 개발**은 재현성과 확장성이 좋아 **프로젝트 환경 통합 관리**에 유리합니다.
* **Ubuntu 로컬 개발**은 설정과 디버깅이 쉬워 **학습 및 테스트**에 적합합니다.

**4. Docker 기반 파이썬 개발 환경 구성**

**4-1. Dockerfile 작성 예시**

FROM python:3.10-slim

WORKDIR /app

COPY requirements.txt .

RUN pip install --no-cache-dir -r requirements.txt

COPY . .

EXPOSE 5000

CMD ["gunicorn", "-w", "2", "-b", "0.0.0.0:5000", "app:app"]

**4-2. 요구사항 파일 예시 (requirements.txt)**

Flask==2.3.2

gunicorn==21.2.0

**4-3. Docker 이미지 빌드 및 실행**

docker build -t krase-homepage .

docker run -d -p 5000:5000 --name krase-homepage krase-homepage

⚠️ 위 방식은 Python 애플리케이션을 Docker 컨테이너에서 실행하며, 로컬 시스템에 Python을 설치하지 않아도 됩니다.

추가로 원하시는 구성 (예: GitHub 연동, Nginx 리버스 프록시, Let's Encrypt 인증서 발급 등)에 대해 계속해서 이어서 정리 가능합니다.