Day 2: Python 문서 파이프라인 + Docker - 완전 가이드

> **GitHub Repository**: RAG-on-Local-CPU-minimal

> 파일 읽기 + 설정 분리 + Docker 실행

🎯 학습 목표

Day 2의 핵심은 다음과 같습니다:

> **"문서를 파일에서 로딩하고, 설정으로 분리하고, Docker로 검증한다"**

이 Day를 마치면:

- ☑ 파일 I/O (`with open`) 이해
- ☑ `config.yaml`로 설정 분리
- ☑ `BASE_DIR` 경로 처리 방식 이해
- ☑ Docker 빌드 및 실행
- ☑ 왜 이런 구조로 만들었는지 이해

주의: Day 2에는 LLM, Embedding, API가 없습니다!

📂 프로젝트 구조

```
Rag_minimal_day2/
├─ app/
│  ├─ main.py         # 실행 진입점
│  ├─ pipeline.py     # 처리 흐름
│  ├─ loader.py       # 문서 로딩
│  └─ config.py       # 설정 로더
│
├─ data/
│  └─ docs.txt        # 실습용 문서
│
├─ config.yaml        # 실행 설정
├─ requirements.txt
├─ Dockerfile
└─ README.md
```

중요: 모든 실행은 프로젝트 루트 기준!

🤔 왜 이런 구조인가?

1. 왜 app/ 폴더로 모았나?

Day 1 구조:

```
main.py
pipeline.py
loader.py
```

Day 2 구조:

```
app/
  ├─ main.py
  ├─ pipeline.py
  └─ loader.py
```

이유:

1. **패키지화**: app을 하나의 모듈로 관리
2. **확장성**: 파일이 늘어나도 app/ 안에 정리
3. **배포 준비**: Docker, FastAPI 등에서 필수 구조
4. **실무 표준**: 대부분의 Python 프로젝트가 이렇게 구성

2. 왜 config.yaml을 분리했나?

Before (코드에 하드코딩):

```python
def load_documents():
    with open("data/docs.txt") as f:  # 하드코딩
        return f.read().split("\n")
```

After (설정 파일 분리):

```yaml
# config.yaml
data:
  document_path: data/docs.txt
```

```python
# loader.py
path = config["data"]["document_path"]  # 설정에서 읽기
```

장점:

- ☑ 코드 수정 없이 동작 변경
- ☑ 환경별 설정 관리 (dev, prod)
- ☑ 테스트 용이 (다른 파일로 테스트)
- ☑ 실무 베스트 프랙티스

3. 왜 BASE_DIR 패턴인가?

문제:

```
# 이렇게 하면?
with open("data/docs.txt") as f:
```

실행 위치에 따라 경로가 달라짐:

```
python app/main.py        # ☑ 작동
cd app && python main.py  # ❌ 에러!
```

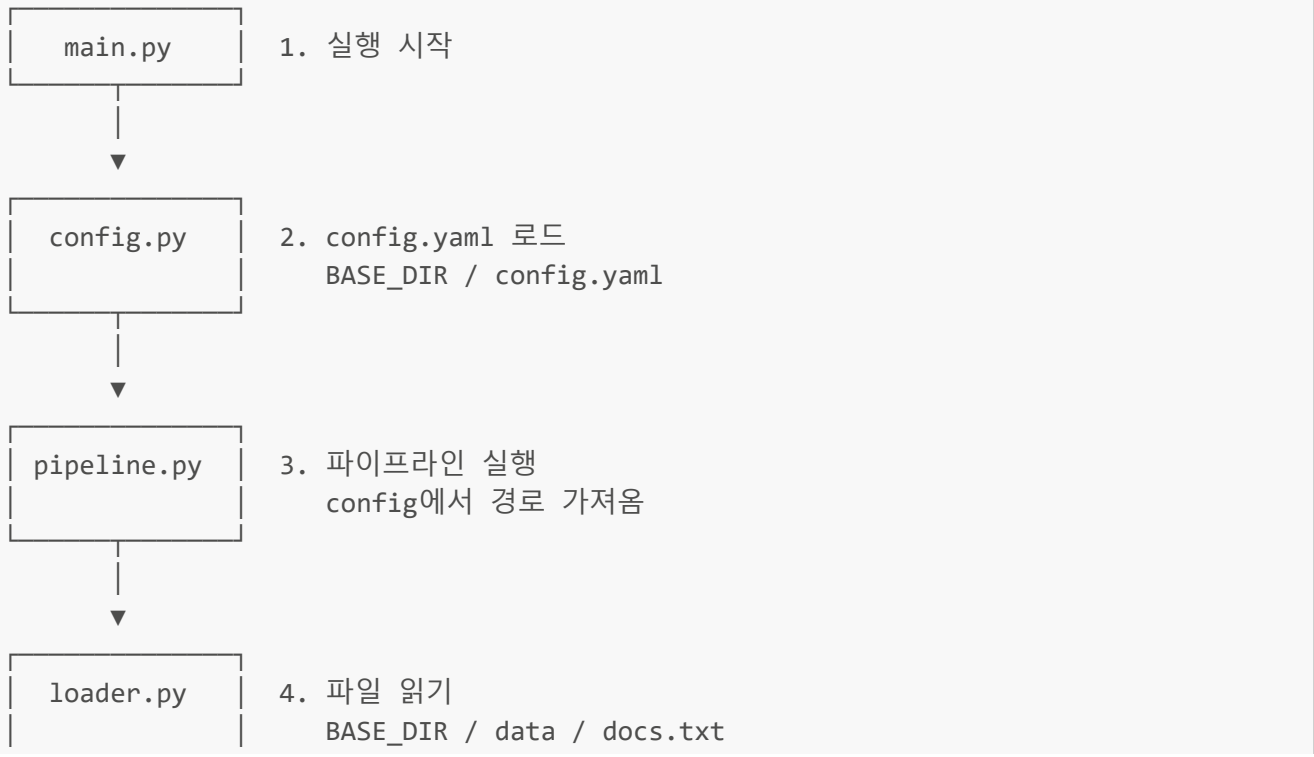
해결:

```
BASE_DIR = Path(__file__).resolve().parent.parent
doc_path = BASE_DIR / "data" / "docs.txt"
```

효과:

- ☑ 어디서 실행해도 동일
- ☑ Docker 내부에서도 동일
- ☑ 상대 경로 문제 완전 해결

🔄 실행 흐름도



```
└──────────┬──────────┘
           │
           ▼
      문서 반환
           │
           ▼
      결과 출력
```

⚙ 로컬 실행

Windows (PowerShell):

```
cd Rag_minimal_day2
python app/main.py
```

macOS/Linux (Bash):

```
cd Rag_minimal_day2
python app/main.py
# 또는
python3 app/main.py
```

예상 출력:

```
=== QUERY ===
RAG가 뭐야?

=== DOCUMENTS ===
- (1) RAG는 Retrieval Augmented Generation이다
- (2) 문서를 검색해서 답변한다
```

Docker 실행 (Day 2 핵심)

1. Docker 이미지 빌드

Windows (PowerShell):

```
docker build -t rag-minimal-day2 .
```

macOS/Linux (Bash):

```
docker build -t rag-minimal-day2 .
```

빌드 과정:

```
1. Dockerfile 읽기
2. Python 3.10 이미지 다운로드
3. 작업 디렉토리 /app 생성
4. 파일 복사 (app/, data/, config.yaml)
5. 패키지 설치 (requirements.txt)
6. 이미지 완성
```

2. 컨테이너 실행

```
docker run --rm rag-minimal-day2
```

- `--rm`: 실행 후 컨테이너 자동 삭제

Docker 내부 구조

```
/app  (WORKDIR)
├─ app/
│  ├─ main.py
│  ├─ pipeline.py
│  ├─ loader.py
│  └─ config.py
├─ data/
│  └─ docs.txt
└─ config.yaml
```

중요: BASE_DIR 덕분에 로컬과 동일하게 작동!

디버깅 방법

방법 1: print() 디버깅

경로 디버깅 (loader.py):

```python
def load_documents(path):
    BASE_DIR = Path(__file__).resolve().parent.parent
    doc_path = BASE_DIR / path

    print("=" * 50)
    print("DEBUG: load_documents()")
```

```
    print("=" * 50)
    print(f"1. __file__: {__file__}")
    print(f"2. BASE_DIR: {BASE_DIR}")
    print(f"3. Input path: {path}")
    print(f"4. Full doc_path: {doc_path}")
    print(f"5. File exists: {doc_path.exists()}")

    with open(doc_path, "r", encoding="utf-8") as f:
        content = f.read()

    print(f"6. Content length: {len(content)} chars")
    print(f"7. First 100 chars: {content[:100]}")
    print("=" * 50)

    return content.split("\n")
```

설정 디버깅 (config.py):

```
def load_config():
    BASE_DIR = Path(__file__).resolve().parent.parent
    config_path = BASE_DIR / "config.yaml"

    print(f"📋 Config path: {config_path}")
    print(f"📋 Config exists: {config_path.exists()}")

    with open(config_path) as f:
        config = yaml.safe_load(f)

    print(f"📋 Config loaded: {config}")
    return config
```

방법 2: VS Code 디버거

사전 준비:

- VS Code Python 확장 설치
- Python Interpreter 선택

디버깅 시작:

1. `app/main.py` 파일 열기
2. 브레이크포인트 설정 (줄 번호 왼쪽 클릭)
3. **F5** → "Python Debugger" 선택
4. 변수 확인:
    - `config`: 로드된 설정
    - `doc_path`: 계산된 경로
    - `documents`: 읽은 문서들

📖 코드 Deep Dive

main.py 한 줄씩

```python
# Line 1-2: 필요한 모듈 임포트
from config import load_config
from pipeline import pipeline

# Line 4: 메인 함수 정의
def main():
    # Line 5: 사용자 쿼리
    query = "RAG가 뭐야?"

    # Line 6: 설정 파일 로드
    # config.yaml을 읽어서 딕셔너리로 반환
    config = load_config()

    # Line 7: 파이프라인 실행
    # query와 config를 전달
    result = pipeline(query, config)

    # Line 9-11: 결과 출력
    print(f"=== QUERY ===")
    print(result["query"])

    print(f"\n=== DOCUMENTS ===")
    for doc in result["documents"]:
        print(f"- {doc}")

# Line 18-19: 직접 실행 시만 main() 호출
if __name__ == "__main__":
    main()
```

핵심 포인트:

- `load_config()`: 설정을 코드로부터 분리
- `pipeline(query, config)`: 설정을 전달받아 처리
- 출력 로직도 main에 있음 (파이프라인은 처리만)

config.py 한 줄씩

```python
# Line 1-2: 필요한 모듈
from pathlib import Path
import yaml

# Line 4: 설정 로드 함수
def load_config():
    # Line 5-6: BASE_DIR 계산
    # __file__: 현재 파일 경로 (config.py)
    # .resolve(): 절대 경로로 변환
```

```
    # .parent.parent: app/ → Rag_minimal_day2/
    BASE_DIR = Path(__file__).resolve().parent.parent

    # Line 7: config.yaml 경로 생성
    # BASE_DIR / "config.yaml"
    config_path = BASE_DIR / "config.yaml"

    # Line 8-9: YAML 파일 읽기
    with open(config_path) as f:
        return yaml.safe_load(f)
```

핵심 포인트:

- `Path(__file__)`: 파일 기준 경로 계산의 시작
- `.parent.parent`: 상위 두 단계로 이동
- `yaml.safe_load()`: YAML을 딕셔너리로 변환

pipeline.py 한 줄씩

```
# Line 1: 문서 로더 임포트
from loader import load_documents

# Line 3: 파이프라인 함수
def pipeline(query, config):
    # Line 4-5: config에서 경로 가져오기
    # config["data"]["document_path"]
    # → "data/docs.txt"
    doc_path = config["data"]["document_path"]

    # Line 6: 문서 로딩
    documents = load_documents(doc_path)

    # Line 8-9: config에서 최대 문서 수 가져오기
    max_docs = config["pipeline"]["max_docs"]

    # Line 10: 문서 개수 제한
    documents = documents[:max_docs]

    # Line 12-15: 결과 딕셔너리 반환
    return {
        "query": query,
        "documents": documents
    }
```

핵심 포인트:

- `config["data"]["document_path"]`: 설정에서 값 읽기
- `documents[:max_docs]`: 슬라이싱으로 개수 제한
- 파이프라인은 "무엇을 할지"만 정의

loader.py 한 줄씩

```
# Line 1: Path 모듈 임포트
from pathlib import Path

# Line 3: 문서 로드 함수
def load_documents(path):
    # Line 4-5: BASE_DIR 계산
    # __file__: loader.py의 경로
    # .parent.parent: app/ → Rag_minimal_day2/
    BASE_DIR = Path(__file__).resolve().parent.parent

    # Line 6: 전체 경로 계산
    # BASE_DIR / path
    # → Rag_minimal_day2 / data / docs.txt
    doc_path = BASE_DIR / path

    # Line 8-9: 파일 읽기
    # encoding="utf-8": 한글 깨짐 방지
    with open(doc_path, "r", encoding="utf-8") as f:
        content = f.read()

    # Line 11: 줄바꿈으로 분리
    # "line1\nline2\nline3" → ["line1", "line2", "line3"]
    return content.split("\n")
```

핵심 포인트:

- 매번 BASE_DIR 재계산 (파일마다 다를 수 있음)
- `BASE_DIR / path`: Path 객체의 / 연산자
- `with open`: 자동으로 파일 닫힘 (안전)

학습 효율을 높이는 팁

1. 경로 디버깅 체계

문제 발생 시 체크 순서:

```
# 1. __file__ 확인
print("__file__:", __file__)
# → .../app/loader.py

# 2. BASE_DIR 확인
BASE_DIR = Path(__file__).resolve().parent.parent
print("BASE_DIR:", BASE_DIR)
# → .../Rag_minimal_day2

# 3. 입력 경로 확인
print("Input path:", path)
```

```
# → data/docs.txt

# 4. 최종 경로 확인
doc_path = BASE_DIR / path
print("Full path:", doc_path)
# → .../Rag_minimal_day2/data/docs.txt

# 5. 파일 존재 여부
print("Exists:", doc_path.exists())
# → True or False
```

2. 파일 I/O 패턴

안전한 파일 읽기:

```
# ☑ 권장 (자동으로 닫힘)
with open(file_path, "r", encoding="utf-8") as f:
    content = f.read()

# ❌ 비권장 (수동으로 닫아야 함)
f = open(file_path, "r")
content = f.read()
f.close()
```

파일이 없을 때 처리:

```
from pathlib import Path

doc_path = BASE_DIR / path

if not doc_path.exists():
    print(f"ERROR: File not found: {doc_path}")
    return []

with open(doc_path, "r", encoding="utf-8") as f:
    return f.read().split("\n")
```

3. Docker 트러블슈팅

문제 1: 빌드 실패

```
# 에러: Dockerfile not found
# 해결: 프로젝트 루트에서 실행
cd Rag_minimal_day2
docker build -t rag-minimal-day2 .
```

문제 2: 파일 복사 실패

```
# 에러: COPY failed: file not found
# 해결: Dockerfile에서 경로 확인
COPY app/ /app/app/
COPY data/ /app/data/
COPY config.yaml /app/
```

문제 3: 실행 결과 다름

```
# 원인: BASE_DIR 경로 차이
# 해결: BASE_DIR 패턴 사용하면 해결됨!
```

🔧 직접 해보기: 실습

실습 1: config.yaml 수정

config.yaml 수정:

```
data:
  document_path: data/docs.txt

pipeline:
  max_docs: 5  # 2 → 5로 변경
```

실행 결과: 더 많은 문서가 출력됩니다!

실습 2: 문서 추가

data/docs.txt에 추가:

```
(1) RAG는 Retrieval Augmented Generation이다
(2) 문서를 검색해서 답변한다
(3) Vector DB를 사용한다        ← 추가
(4) Embedding으로 변환한다      ← 추가
(5) 유사도 검색을 수행한다      ← 추가
```

config.yaml:

```
pipeline:
  max_docs: 5  # 더 많이 출력
```

실행 결과: 새로 추가한 문서도 함께 출력!

실습 3: 디버깅 메시지 추가

각 파일에 print 추가:

```
# main.py
def main():
    print("🚀 프로그램 시작")
    config = load_config()
    print(f"📋 설정 로드 완료: {config}")
    # ...

# config.py
def load_config():
    print(f"📋 설정 파일 로딩: {config_path}")
    # ...

# pipeline.py
def pipeline(query, config):
    print(f"📊 파이프라인 시작: {query}")
    # ...

# loader.py
def load_documents(path):
    print(f"📂 문서 로딩: {path}")
    # ...
```

실행 결과:

```
🚀 프로그램 시작
📋 설정 파일 로딩: .../config.yaml
📋 설정 로드 완료: {'data': {...}, ...}
📊 파이프라인 시작: RAG가 뭐야?
📂 문서 로딩: data/docs.txt
...
```

☑ Day 2 완료 기준

다음을 이해하고 실습했다면 완료입니다:

이해

- ✔ 왜 app/ 폴더 구조로 바뀌었는지
- ✔ 왜 config.yaml로 분리했는지
- ✔ BASE_DIR 패턴이 왜 필요한지
- ✔ Docker 빌드와 실행의 차이

코드

- ✔ main.py의 역할 (진입점)
- ✔ config.py의 역할 (설정 로드)
- ✔ pipeline.py의 역할 (흐름 정의)
- ✔ loader.py의 역할 (파일 읽기)

실습

- ✔ 로컬에서 실행 성공
- ✔ config.yaml 수정 후 결과 변화 확인
- ✔ 문서 추가/수정 후 반영 확인
- ✔ Docker 빌드 및 실행 성공
- ✔ 경로 디버깅 해봄

➡ SOON 다음 단계 (Day 3)

Day 3에서는 다음을 추가합니다:

FastAPI 도입:

```
@app.post("/query")
def query_endpoint(request: QueryRequest):
    result = pipeline(request.query, config)
    return result
```

변화:

- 스크립트 → API 서버
- 직접 실행 → HTTP 요청
- print → JSON 응답
- Swagger UI 제공

"스크립트 → 서비스" 전환!

💬 학습 후 자가 점검

다음 질문에 답할 수 있나요?

1. **BASE_DIR은 어떻게 계산되나?**
    - `Path(__file__).resolve().parent.parent`
2. **config.yaml의 장점은?**
    - 코드 수정 없이 동작 변경 가능
3. **with open의 장점은?**

- 자동으로 파일 닫힘 (안전)

4. **Docker 빌드와 실행의 차이는?**
   - 빌드: 이미지 생성 / 실행: 컨테이너 실행
5. **경로 에러 발생 시 첫 체크는?**
   - BASE_DIR과 doc_path를 print로 확인

모두 답할 수 있다면 Day 2 완료! 🎉

📜 라이선스

허용

- ☑ 개인 학습 목적 사용
- ☑ 출처 표시 후 비영리 공유

금지

- ❌ 상업적 이용
- ❌ 내용 수정 및 2차 저작
- ❌ 저작자 허락 없는 재배포

상업적 이용 문의: j4angguiop@gmail.com

⭐ **이 프로젝트가 도움이 되었다면 Star를 눌러주세요!**