노코드 멀티 에이전트 워크플로우 · 보충 강의

Docker 와 컨테이너 , 쉽게 이해하기

n8n 컨테이너는 왜 localhost 가 아니라 postgres 라는 이름으로 DB 를 찾아갈까 ?



왜 생겼나

역사와 필요성



어떻게 통하나

Compose 와 네트워크



운영에서는

실제 서비스의 관점

오늘의 질문 : 이 설정은 왜 안 될까요 ?

n8n 에서 PostgreSQL Credential 을 만들 때 가장 많이 하는 실수

X 많이 하는 설정

```
Host      localhost
Port      5432
Database  n8n_data
User      postgres
```

결과 : connect ECONNREFUSED 127.0.0.1:5432

O 맞는 설정

```
Host      postgres
Port      5432
Database  n8n_data
User      postgres
```

조건 : 두 컨테이너가 같은 Docker 네트워크에 있을 것

오늘 이 두 가지를 이해하면 됩니다

1. 컨테이너 안의 localhost 는 '그 컨테이너 자기 자신' 이다
2. 같은 네트워크의 컨테이너는 '서비스 이름' 으로 서로를 찾는다

“제 PC 에서는 되는데요 ?” — 모든 것의 시작

프로그램은 코드만으로 돌지 않습니다 . 코드를 둘러싼 ‘환경’ 이 함께 맞아야 합니다

내 노트북 O

- Python 3.12
- PostgreSQL 16
- 라이브러리 A 2.1
- Ubuntu 24.04

팀원 노트북 X

- Python 3.10
- PostgreSQL 14
- 라이브러리 A 1.8
- macOS

운영 서버 X

- Python 3.9
- PostgreSQL 없음
- 라이브러리 A 없음
- Ubuntu 20.04

그래서 나온 생각 코드와 함께 ‘환경 전체 (OS 파일 · 언어 · 라이브러리 · 설정)’ 를 한 상자에 담아 보내면 어디서나 똑같이 돌지 않을까 ?

환경을 맞추는 방법은 이렇게 발전했습니다

한 대에 하나 → 한 대에 여러 '가짜 컴퓨터' → 한 대에 여러 '가벼운 상자'



물리 서버

1990 년대 ~

서비스마다 서버 한 대

- 설치를 손으로
- 서버가 놀아도 그대로
- 새 서버 준비에 몇 주



가상머신 (VM)

2000 년대 ~

한 대에 여러 '가짜 컴퓨터'

- VM 마다 OS 를 통째로
- 수 GB · 부팅 수십 초 ~ 분
- 그래도 서버 활용률 ↑



컨테이너

2013 년 Docker ~

한 대에 여러 '가벼운 상자'

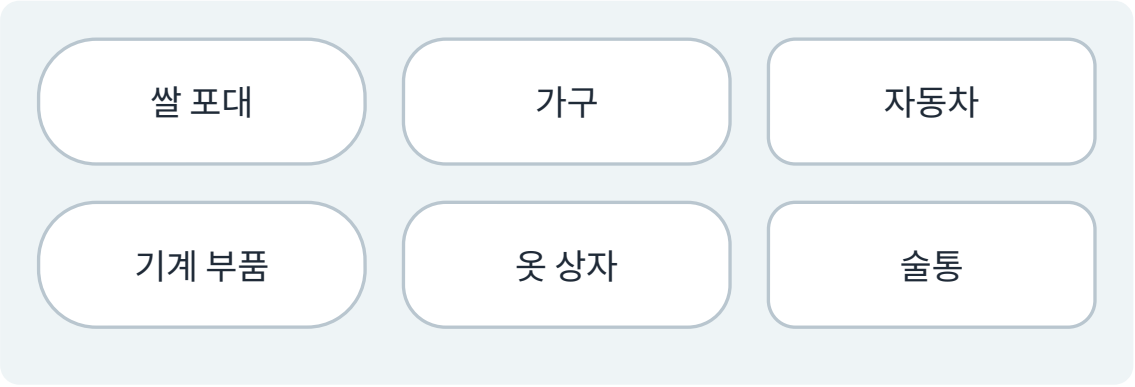
- OS 의 핵심 (커널) 은 공유
- 수십 ~ 수백 MB · 시작 1~2 초
- 이미지 하나로 어디서나

VM 도 여전히 쓰입니다 — 우리 팀 배포 VM 안에서 Docker 컨테이너를 돌리는 것처럼 , 둘은 함께 씁니다

이름이 ‘컨테이너’ 인 이유 : 운송 컨테이너

1956 년 규격화된 운송 컨테이너가 물류를 바꿨듯 , Docker 는 소프트웨어 배포를 바꿨습니다

컨테이너 이전의 항구



모양이 다 달라서 배 · 트럭 · 크레인마다 다르게 다뤄야 함 → 느리고 잘 깨짐

규격 컨테이너 이후



안에 무엇이 있든 겉모양이 같아서 배 · 트럭 · 크레인이 똑같이 다룸

화물 → 내 프로그램 + 라이브러리 + 설정

규격 컨테이너 → Docker 이미지 (어디서나 같은 모양)

배 · 트럭 · 크레인 → 노트북 · 팀 VM · 클라우드 (Docker 만 있으면 됨)

Docker 는 갑자기 나온 게 아닙니다

40 년 넘게 쌓인 리눅스의 ‘격리’ 기술을 누구나 쓰기 쉽게 묶은 것이 Docker

1979 — chroot

프로그램이 볼 수 있는 폴더를 제한

2000 — FreeBSD jail

프로세스를 ‘감옥’처럼 분리

2008 — cgroups · LXC

리눅스에서 CPU · 메모리 제한 + 격리

2013 — Docker 공개

이미지 · 명령어 · 공유 저장소로 ‘쉽게’

2015 — OCI · Kubernetes 1.0

컨테이너 표준 · 여러 서버 관리

지금 — 사실상 표준

n8n · PostgreSQL · Ollama 도 공식 이미지

Docker 가 더한 세 가지

이미지

환경을 파일 하나로 묶고 , 층 (layer) 으로 쌓아 재사용

쉬운 명령어

docker run 한 줄이면 실행

공유 저장소

Docker Hub 에서 공식 이미지를 받아 씀

가상머신 vs 컨테이너 : 단독주택과 아파트

가장 큰 차이는 ‘OS 를 통째로 갖느냐 , 건물 기초 (커널) 를 함께 쓰느냐’

가상머신 = 단독주택

집마다 기초 · 배관 · 전기를 따로



크기	수 GB
시작	수십 초 ~ 분
격리	더 강함

컨테이너 = 아파트 호실

기초 · 배관은 함께 , 현관문은 각자



크기	수십 ~ 수백 MB
시작	1 ~ 2 초
격리	충분히 강함 (커널은 공유)

이미지와 컨테이너 : 붕어빵 틀과 붕어빵

이미지는 '설계도 + 재료가 담긴 틀', 컨테이너는 그 틀로 찍어 낸 '실행 중인 하나'

이미지 (image)

- 읽기 전용 틀
- 예 : postgres:16
- n8nio/n8n:2.42.6
- Docker Hub 에서 받음

docker run

postgres-1
실행 중 · 내 데이터 A

postgres-2
실행 중 · 내 데이터 B

postgres-3
멈춤 · 지우면 사라짐

같은 틀로 여러 개를 찍어 낼 수 있음

```
# 이미지 받기
docker pull postgres:16

# 이미지로 컨테이너 실행
docker run -d --name db \
  -e POSTGRES_PASSWORD=pw \
  postgres:16

# 실행 중인 컨테이너 보기
docker ps
```

이미지 이름 읽는 법 `n8nio/n8n` = 누가 만든 무엇 , `:2.42.6` = 태그 (버전). 수업이 버전을 고정하는 이유 : 모두 같은 틀을 쓰려고

컨테이너 하나 = 작은 컴퓨터 한 대

각자 자기 파일 · 자기 프로그램 · 자기 네트워크를 가집니다 — 그래서 '자기 localhost' 도 따로

n8n

- 파일 : /home/node/.n8n
- 프로그램 : n8n 만 실행 중
- 네트워크 주소 : 172.18.0.3
- **localhost = n8n 자기 자신**

postgres

- 파일 : /var/lib/postgresql/data
- 프로그램 : PostgreSQL 만
- 네트워크 주소 : 172.18.0.4
- **localhost = postgres 자기 자신**

그래서

n8n 안에서 'localhost:5432' 에 접속하면 n8n 컨테이너 자기 안의 5432 번을 찾습니다 .

그 안에는 PostgreSQL 이 없으므로 → 연결 거부

아파트로 다시 비유하면



101 호에서 "우리 집 냉장고 (localhost)" 를 열면 101 호 냉장고가 열립니다 . 102 호 냉장고를 쓰려면 "102 호 (postgres)" 라고 호수를 불러야 합니다 .

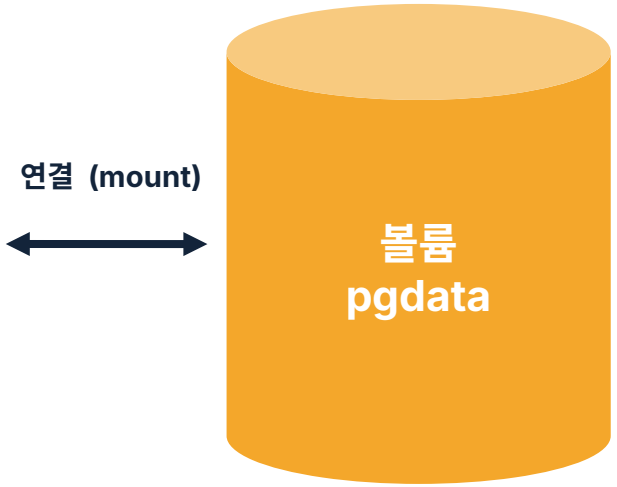
데이터는 어디에 ? 볼륨 (volume)

컨테이너는 지우면 사라집니다 — 남겨야 할 데이터는 컨테이너 바깥 '창고'에 둡니다

postgres (컨테이너)

언제든 지우고 새로 만들 수 있음
(이미지 버전을 올릴 때 등)

연결 (mount)



컨테이너를 지워도 남음

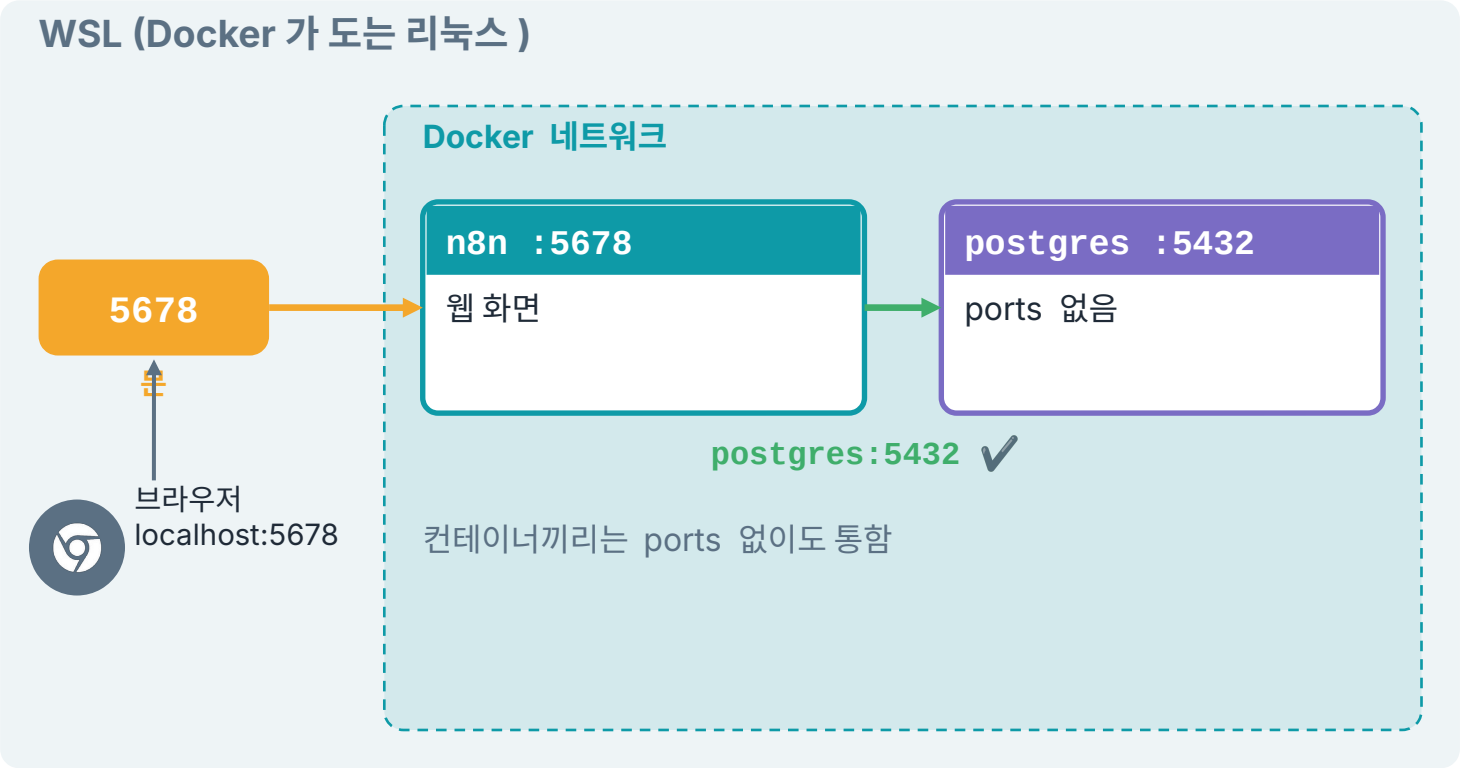
```yaml
services:
  postgres:
    image: postgres:16
    volumes:
      - pgdata:/var/lib/
          postgresql/data

volumes:
  pgdata:
```

우리 수업의 볼륨	day65-n8n_n8n-data → n8n 의 워크플로우 · 계정 · Credential · 실행 기록
가장 위험한 명령	docker compose down -v → 볼륨까지 지움 = 데이터 삭제
안전한 명령	docker compose down → 컨테이너만 지움, 볼륨은 그대로

포트 (ports): 바깥에서 컨테이너로 들어오는 문

ports 는 '바깥 → 컨테이너' 문이고, 컨테이너끼리 대화하는 데는 필요 없습니다




```yaml
services:
  n8n:
    ports:
      - "5678:5678"
#    바깥 : 컨테이너

  postgres:
    # ports 없음 = 바깥에서 못 들어옴
    # (운영 DB는 이게 안전)
```

"15432:5432" 라면 ? WSL 에서는 localhost:15432, n8n 같은 컨테이너에서는 여전히 postgres:5432 (컨테이너 안쪽 번호)

Docker Compose: 여러 컨테이너를 한 파일로

n8n 하나만으로는 서비스가 안 됩니다 — 함께 도는 컨테이너들을 '한 팀'으로 묶는 편성표

```yaml
name: day65-n8n
# ← 프로젝트 이름

services:
  n8n:
    image: n8nio/n8n:2.42.6
    ports: ["5678:5678"]
    volumes: [n8n-data:/home/node/.n8n]

  postgres:
    image: postgres:16
    environment:
      POSTGRES_PASSWORD: ${PG_PASSWORD}
    volumes: [pgdata:/var/lib/postgresql/data]

volumes:
  n8n-data:
  pgdata:
```

name — **프로젝트 이름 (없으면 폴더 이름)**

네트워크 이름이 여기서 정해짐 → day65-n8n_default

services — **함께 돌 컨테이너들**

여기 적힌 이름 (n8n, postgres) 이 서로 부르는 이름

volumes — **남겨야 할 데이터 창고**

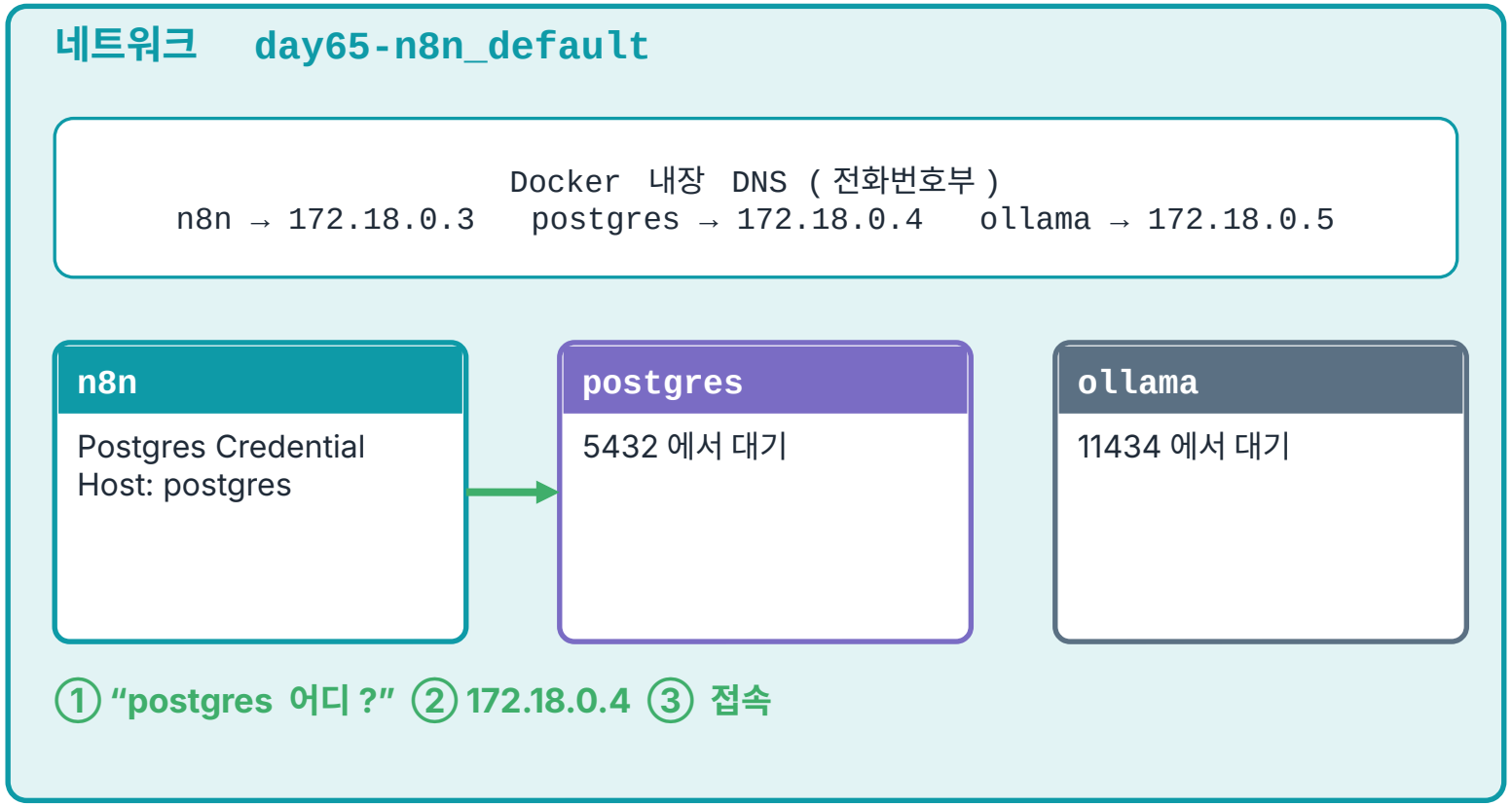
컨테이너를 지워도 남음

명령 하나로 — **up -d · ps · logs · down**

`docker compose ○○` → 이 폴더의 `compose.yml` 전체를 한 번에

핵심 : 같은 Compose = 같은 건물의 내선 전화

Compose 는 프로젝트마다 네트워크를 하나 만들고 , 서비스 이름을 '내선 번호' 처럼 등록합니다



전화번호부

Compose 가 서비스 이름을 자동 등록 (주소는 바뀌어도 이름은 그대로)

내선 번호

서비스 이름 + 컨테이너 안쪽 포트 → postgres:5432

외선 (바깥)

다른 건물이나 WSL 에서 오려면 ports 로 연 문을 통해서만

컨테이너끼리는 '서비스이름 : 안쪽포트' 로 부른다 — n8n → http://ollama:11434, postgres:5432, http://data-api:8800

localhost 가 세 개 있습니다

같은 ‘localhost’ 라는 말이 어디서 쓰느냐에 따라 전혀 다른 곳을 가리킵니다

Windows	**브라우저에서 `localhost:5678`** → WSL 로 전달되어 n8n 의 문 (ports) 으로	O
WSL 터미널	**`curl localhost:8800 · flow.py`** → ports 로 연 문 (8800, 5678, 11434) 으로	O
n8n 컨테이너 안	**`localhost:5432`** → n8n 자기 자신 . PostgreSQL 은 없음	X

WSL 터미널에서 DB 보기

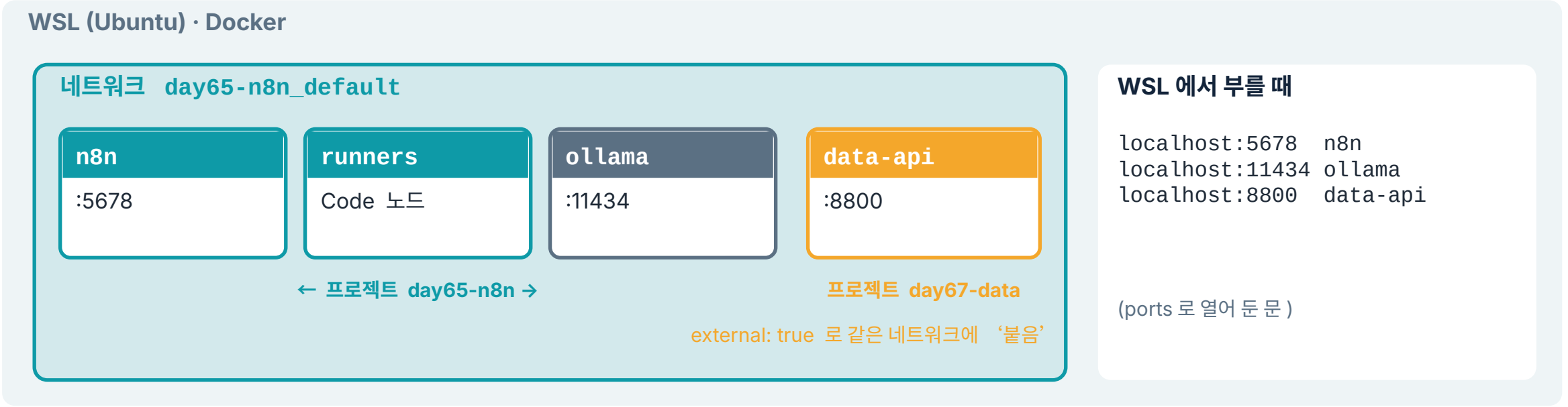
`psql -h localhost -p 15432` (ports 를 열었을 때만)

n8n 에서 DB 쓰기

`Host postgres · Port 5432` (같은 네트워크면 언제나)

우리 수업의 실제 구성 그대로 보기

Last3Mon/65/n8n (프로젝트 day65-n8n) + Last3Mon/76/data-api (프로젝트 day67-data)




```yaml
# Last3Mon/76/data-api/compose.yml
name: day67-data
services:
  data-api:
    networks: [n8n]
networks:
  n8n: { external: true,
         name: day65-n8n_default }
```

n8n 노드에서 부를 때

```
http://ollama:11434
http://data-api:8800
```

→ 둘 다 서비스 이름 + 안쪽 포트

실습 : n8n 이 쓸 PostgreSQL 을 붙이는 두 가지 방법

data-api 와 같은 방식 (B) 을 권합니다 — n8n 설치 파일을 건드리지 않아도 됩니다

A. 같은 compose.yml 에 넣기

```
# ~/day65-n8n 쪽 compose 에 추가
services:
  n8n: ...
  postgres:
    image: pgvector/pgvector:pg16
    environment:
      POSTGRES_USER: app
      POSTGRES_PASSWORD: ${PG_PASSWORD}
      POSTGRES_DB: app
    volumes: [pgdata:/var/lib/postgresql/data]
volumes:
  pgdata:
```

B. 따로 만들고 같은 네트워크에 붙이기 (권장)

```
# 예 : ~/pg/compose.yml
name: course-pg
services:
  postgres:
    image: pgvector/pgvector:pg16
    environment: { ... A 와 같음 ... }
    volumes: [pgdata:/var/lib/postgresql/data]
    networks: [n8n]
networks:
  n8n: { external: true, name: day65-n8n_default }
volumes:
  pgdata:
```

n8n Postgres Credential `Host postgres` `Port 5432` `Database app` `User app` `Password (.env)`

다른 Compose 프로젝트끼리는 기본적으로 못 봅니다

건물이 다르면 내선 전화가 안 됩니다 — 이어 주는 방법 세 가지



1	**같은 compose.yml 에 넣기**	가장 단순 . 한 팀이 같이 켜고 끄는 구성일 때	좋음
2	**external 네트워크로 붙이기**	따로 관리하되 이름으로 통신 (수업의 data-api 방식)	좋음
3	**호스트의 열린 문으로 돌아가기**	host.docker.internal:15432 (리눅스는 extra_hosts 설정 필요)	임시용

연결이 안 될 때 점검 순서 5 단계

위에서부터 하나씩 — 대부분 1~3 단계에서 원인이 나옵니다

1	**떠 있나 ?**	`docker ps`	postgres 컨테이너가 Up 인지 (Restarting 이면 logs 부터)
2	**같은 네트워크인가 ?**	`docker network inspect day65-n8n_default`	Containers 목록에 n8n 과 postgres 가 둘 다 있는지
3	**이름 · 포트가 맞나 ?**	`Host = 서비스 이름 , Port = 안쪽 포트`	localhost ✘ · 바깥 포트 (15432) ✘ → postgres:5432
4	**네트워크 안에서 닿나 ?**	`docker run --rm --network day65-n8n_default \ postgres:16 pg_isready -h postgres -p 5432`	accepting connections 가 나오면 네트워크는 정상
5	**계정이 맞나 ?**	`docker logs <postgres 컨테이너 >`	password authentication failed → 사용자 · 비밀번호 · DB 이름

4 단계 명령은 임시 컨테이너를 같은 네트워크에 붙여 ‘n8n 자리에서’ 접속해 보는 방법입니다 (끝나면 자동 삭제)

자주 하는 실수 6 가지

오류 메시지를 보면 대부분 원인을 바로 알 수 있습니다

실수	보이는 증상	고치는 법
Host 에 localhost	`ECONNREFUSED 127.0.0.1:5432`	Host 를 서비스 이름 (postgres) 으로
바깥 포트를 씀	`ECONNREFUSED postgres:15432`	컨테이너끼리는 안쪽 포트 5432
다른 프로젝트에 띄움	`ENOTFOUND postgres`	같은 compose 에 넣거나 external 네트워크
DB 가 아직 준비 전	켜자마자 연결 실패, 잠시 뒤엔 됨	healthcheck + depends_on (service_healthy)
down -v 로 정리	워크플로우 · DB 데이터가 사라짐	평소엔 docker compose down (-v 금지)
비밀번호를 compose.yml 에	저장소에 비밀번호가 올라감	.env 에 두고 ${PG_PASSWORD} 로 , 커밋 전 tscan

운영에서 Docker 가 주는 다섯 가지

개발할 때의 편리함이 운영에서는 ‘사고를 줄이는 힘’ 이 됩니다



같은 이미지

노트북에서 시험한 그 이미지가 팀 VM · 운영 서버에서 그대로 돎



쉬운 되돌리기

새 버전이 문제면 이전 태그로 바꿔 다시 up (n8n 2.42.6 → 이전 버전)



자동 재시작

restart: unless-stopped → 서버가 재부팅돼도 다시 켜짐



자원 제한

컨테이너마다 CPU · 메모리 상한 → 하나가 서버 전체를 잡아먹지 않게



보이는 운영

docker logs · ctop · 상태 (health) 로 어느 컨테이너가 문제인지 바로

운영에서는 이것만은 꼭

실습에서는 넘어가도 되지만 , 실제 서비스에서는 사고로 이어지는 것들



이미지 버전 고정

latest 대신 정확한 태그 (postgres:16.x, n8nio/n8n:2.42.6)

모르는 사이에 버전이 바뀌는 사고 방지



비밀은 .env · 비밀 관리 도구에

compose.yml · 저장소에 비밀번호를 쓰지 않음

커밋 전 tscan staged



DB 포트는 바깥에 열지 않기

PostgreSQL 은 ports 없이 내부 네트워크로만

인터넷에서 DB 를 두드리는 공격 차단



볼륨 백업

pg_dump 로 정기 백업 , 복구도 실제로 해 보기

‘ 백업은 복구해 봐야 백업’



상태 확인 · 로그

healthcheck · docker logs · 모니터링

문제를 사용자보다 먼저 알기

한 대에서 여러 대로 : Compose → Kubernetes

오늘 배운 개념은 규모가 커져도 이름만 바뀌어 그대로 쓰입니다

Docker Compose		Kubernetes
서버 한 대 · 팀 VM · 작은 서비스 · 1인 SaaS 초기		서버 여러 대 · 자동 확장 · 무중단 배포가 필요할 때
`compose.yml`	→	매니페스트 (YAML)
`services`	→	Deployment · Pod
서비스 이름으로 찾기	→	Service 이름으로 찾기 (클러스터 DNS)
`ports`	→	Service · Ingress
`volumes`	→	PersistentVolume
`restart`	→	자동 재시작 · 다른 서버로 옮겨 실행

처음부터 Kubernetes 가 필요하지는 않습니다 — 대부분은 Compose 로 시작해 필요할 때 옮깁니다

정리

오늘의 다섯 문장

이 다섯 문장이면 n8n ↔ PostgreSQL 연결은 다시 헷갈리지 않습니다

1. 컨테이너는 환경째로 담은 상자 — 어디서나 똑같이 돈다
2. 이미지는 틀 , 컨테이너는 실행 중인 하나 , 데이터는 볼륨에
3. 컨테이너 안의 localhost 는 그 컨테이너 자기 자신이다
4. 같은 네트워크의 컨테이너는 ‘서비스이름 : 안쪽포트’ 로 부른다
5. ports 는 바깥에서 들어오는 문 — 컨테이너끼리는 필요 없다

```
n8n  →  postgres:5432
```