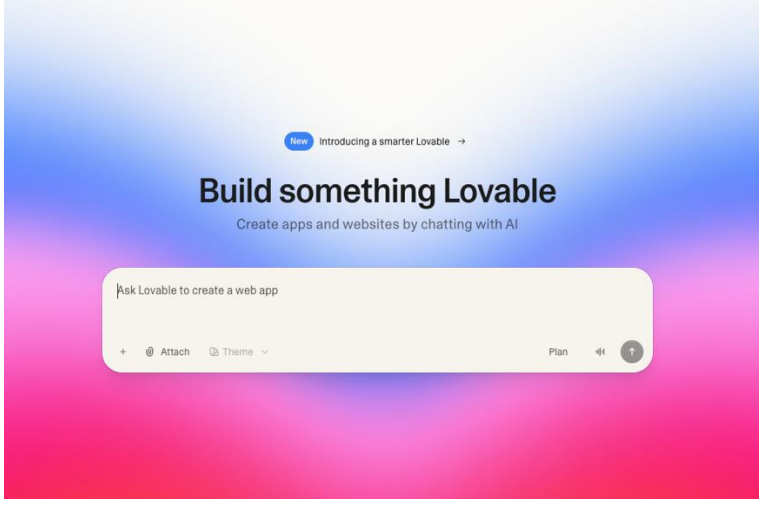
겨울방학 스터디
5회차

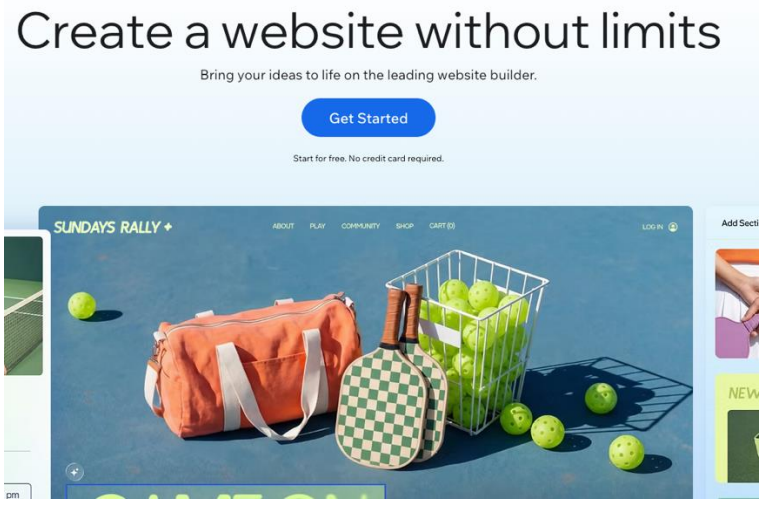
College of Art & Technology
Chung-Ang University

Complex Intelligent Systems Laboratory

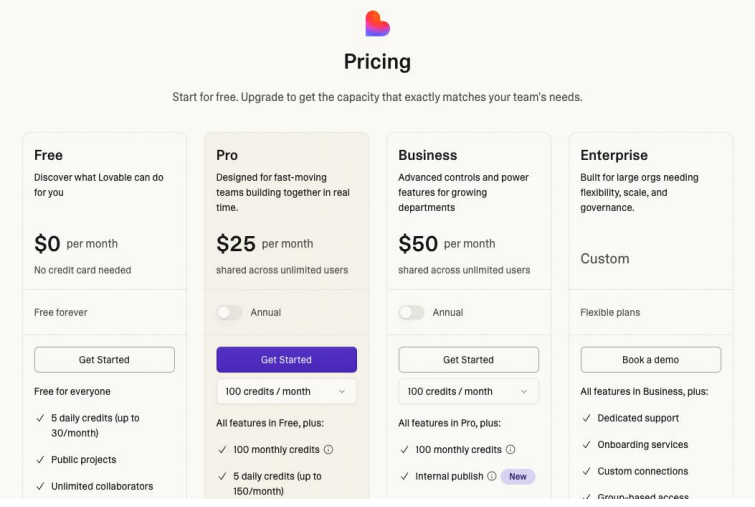
https://cislab.cau.ac.kr

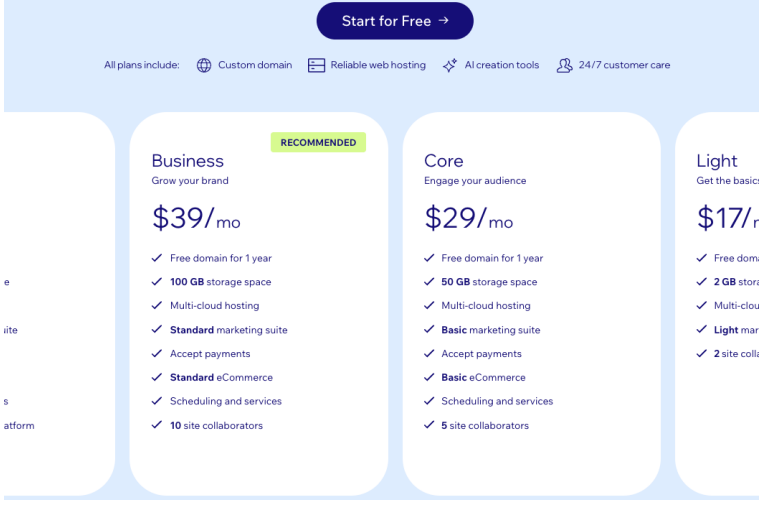
왜 배포를 직접 배워야 하는가?





왜 배포를 직접 배워야 하는가?



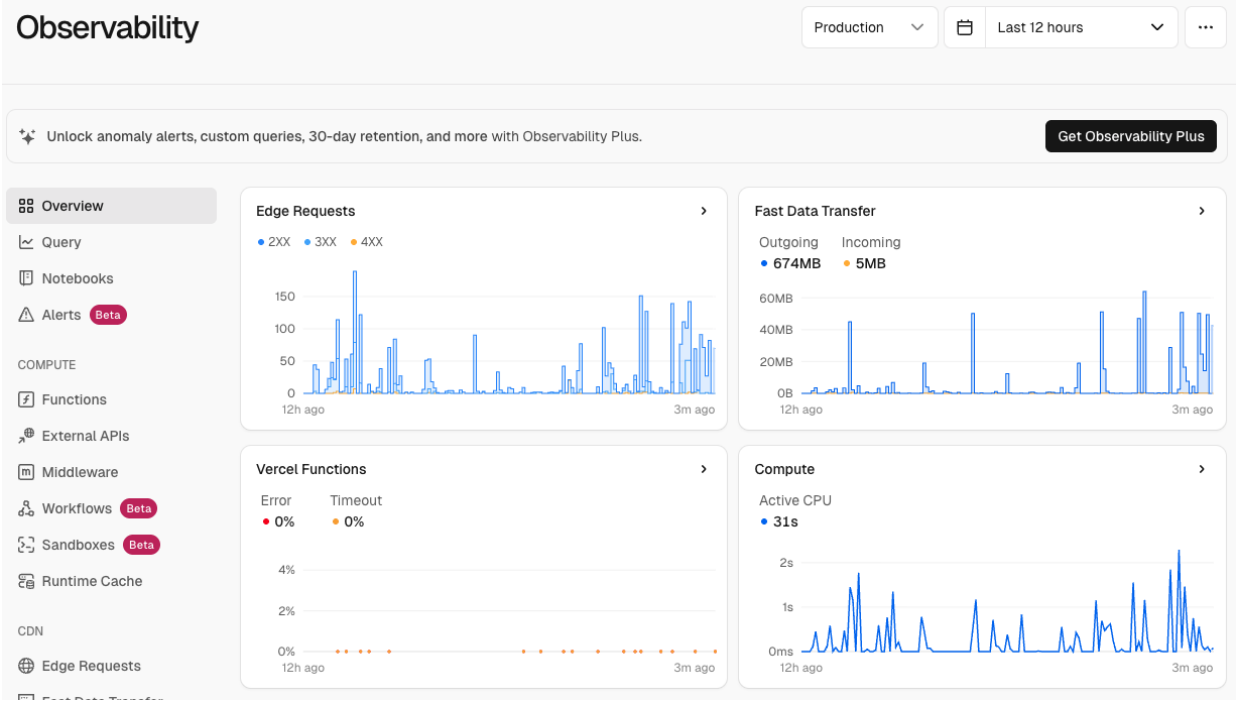


왜 배포를 직접 배워야 하는가?

or

OpenNEXT

왜 배포를 직접 배워야 하는가?

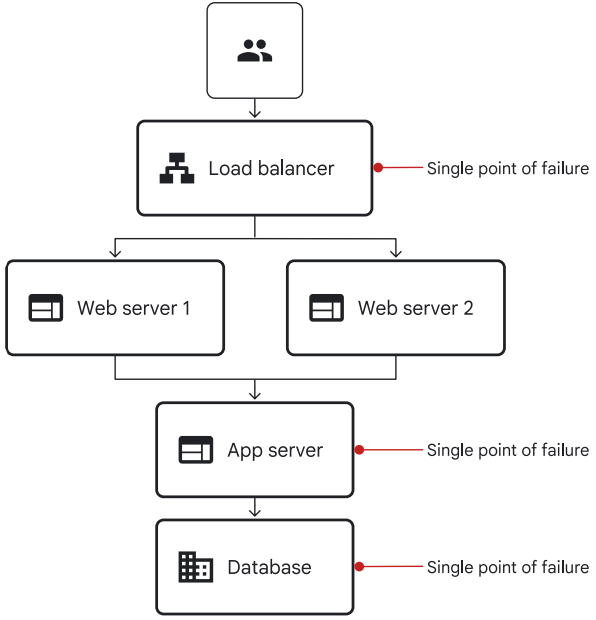


왜 배포를 직접 배워야 하는가?

AI 시대에도 인프라 운영은 경험이 핵심

장애 대응, 보안, 스케일링은 코드 개발과 다른 영역

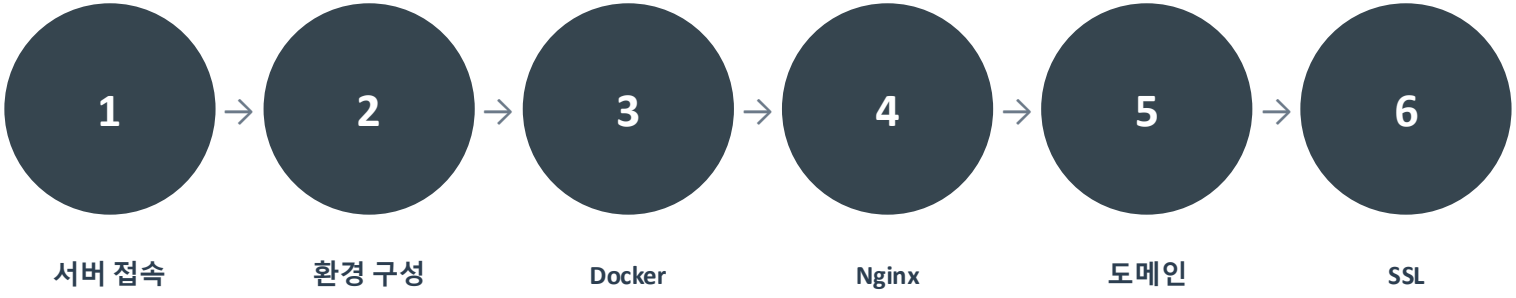
서비스를 배포하고 운영해본 경험의 가치



KMAP 프로젝트 예시

- **kmap.example.com**
  React Frontend
- **kmap.example.com/api**
  FastAPI Backend
- **PostgreSQL**
  내부 네트워크
- **모든 요청**
  HTTPS 적용

배포 프로세스



SSH 연결 방법

SSH = Secure Shell: 원격 서버에 암호화된 연결로 안전하게 접속하는 프로토콜

방법 1: 터미널 직접 접속

```
ssh [사용자명]@[서버IP]
ssh kmap@192.168.xxx.xxx
```

방법 2: VSCode Remote SSH ⭐

- Extension: "Remote - SSH" 설치
- 익숙한 IDE 환경에서 원격 작업
- 파일 탐색, 편집, 터미널 모두 가능

리눅스 명령어 (폴더 탐색)

`pwd`	Print Working Directory. 현재 위치 경로 확인
`ls -la`	-l: 상세정보(권한,크기,날짜), -a: 숨김파일(.env 등) 포함
`cd ~/project`	~는 홈 디렉토리(/home/사용자명)의 단축 표현
`mkdir -p a/b/c`	-p: 중간 디렉토리 없으면 자동 생성
`cat / less`	cat: 전체출력(짧은파일), less: 페이지단위(q로 종료)
`sudo`	SuperUser DO. 관리자 권한 실행 (패키지 설치 등)

리눅스 명령어 (시스템 및 리소스 정보)

시스템 정보

`uname -a`	커널 버전, 아키텍처(x86_64/arm64) 확인
`cat /etc/os-release`	Ubuntu 버전 확인. 패키지 설치 시 참고

리소스 확인

`htop`	CPU/메모리 실시간 모니터링. 빨간색=위험
`df -h`	Disk Free. -h: GB단위 표시. Docker 이미지 용량 확인
`free -h`	메모리 사용량. available이 실제 사용 가능한 메모리

리눅스 명령어 (패키지 관리)

`sudo apt update`	패키지 목록만 갱신 (설치X). 최신 버전 정보 동기화
`sudo apt upgrade -y`	설치된 패키지 실제 업그레이드. -y: 모든 질문에 Yes
`sudo apt install -y git curl vim htop`	여러 패키지 한 번에 설치

💡 update와 upgrade 혼동 주의! update 없이 install하면 구버전이 설치될 수 있음

개발 vs 배포 환경

항목	Development	Production
소스 반영	볼륨 마운트	이미지에 포함
디버깅	상세 로그	에러만
빌드	개발 서버	빌드된 정적 파일
DB 데이터	초기화 가능	영속성 필수

Docker 최적화 전략

1. **Multi-stage Build**
   이미지 크기 최소화

2. **레이어 캐싱 활용**
   빌드 시간 단축

3. **.dockerignore 활용**
   불필요한 파일 제외

4. **non-root 사용자**
   보안 강화

.dockerignore 파일 작성

왜 필요한가?

COPY . . 실행 시 모든 파일이 Docker 데몬으로 전송됨

- 빌드 시간 증가 (불필요한 파일 전송)
- 이미지 크기 증가
- 민감 정보 노출 위험 (.env, .git 등)
- 캐시 무효화 (관련 없는 파일 변경에도 재빌드)

✅ **효과**

빌드 컨텍스트: 500MB → 5MB (100배 감소)
빌드 시간: 2분 → 20초

.dockerignore 예시

```
# 의존성 (컨테이너에서 새로 설치)
node_modules/
__pycache__/

# 버전 관리 (불필요)
.git/
.gitignore

# 환경 설정 (민감 정보!)
.env
.env.*

# 불필요한 파일
*.log
.DS_Store
Dockerfile*
docker-compose*
```

Frontend Dockerfile (React)

```
# Build stage
FROM node:20-alpine AS builder
WORKDIR /app
COPY package*.json ./
RUN npm ci
COPY . .
RUN npm run build

# Production stage
FROM nginx:alpine
COPY --from=builder /app/dist /usr/share/nginx/html
```

코드 설명

- node:20-alpine: 경량 이미지 (~50MB vs debian ~350MB)
- AS builder: 이 스테이지에 이름 부여 (나중에 참조)
- package*.json 먼저 복사 → 캐시 최적화
- npm ci: lock 파일 기준 정확히 설치 (npm install보다 빠름)
- FROM nginx:alpine: 새 이미지로 시작 (node 없음)
- --from=builder: 이전 스테이지에서 빌드 결과만 복사

✅ 최종 이미지: nginx + 정적파일만 (~30MB)
node_modules, 소스코드 없음 (보안↑, 용량↓)

멀티 스테이지 빌드

❌ 일반 빌드 시 이미지에 포함되는 것들

- node_modules (수백 MB)
- 빌드 도구 (webpack, typescript 등)
- 소스코드 전체
- 개발용 의존성 (devDependencies)

→ 보안 위험 + 배포 시간 증가 + 저장 공간 낭비

용량 비교

일반 빌드: ~1.2GB

Multi-stage: ~30MB

약 97% 감소!

보안 측면

- 소스코드 노출 방지
- node_modules 내 취약점 제거

동작 원리

Stage 1 (builder)

- Node.js 설치
- npm install (의존성)
- npm run build (빌드)

→ 결과물만 복사

Stage 2 (production)

- Nginx만 설치
- 빌드 결과물만 복사
- 나머지는 모두 포함되지 않음

Backend Dockerfile (FastAPI)

```
FROM python:3.11-slim

WORKDIR /app

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

COPY . .

CMD ["uvicorn", "main:app", "--host", "0.0.0.0"]
```

코드 설명

- python:3.11-slim: 경량 이미지 (~150MB vs full ~900MB)
- WORKDIR /app: 작업 디렉토리 설정 (없으면 자동 생성)
- requirements.txt 먼저 복사 → 레이어 캐싱 최적화
- --no-cache-dir: pip 캐시 저장 안 함 → 이미지 용량 절약
- COPY . .: 소스코드 복사 (.dockerignore로 불필요 파일 제외)
- main:app: main.py 파일의 app 객체
- --host 0.0.0.0: 모든 인터페이스에서 접근 허용
  (기본값 127.0.0.1은 컨테이너 외부 접근 불가)

docker-compose.dev.yml

```yaml
services:
  frontend:
    build: ./frontend
    volumes:
      - ./frontend:/app  # 소스 마운트
    ports:
      - "5173:5173"
    command: npm run dev
```

코드 설명

- build: ./frontend: Dockerfile 위치 지정

- volumes: 호스트 폴더를 컨테이너에 연결
  → 호스트에서 파일 수정 시 컨테이너에 즉시 반영
  → Hot Reload 가능 (실시간 코드 변경 반영)
  ⚠ 프로덕션에서는 절대 사용 금지 (보안 위험)

- ports: "호스트포트:컨테이너포트"
  → 호스트 5173으로 접속하면 컨테이너 5173으로 연결

- command: Dockerfile의 CMD 덮어쓰기
  → 개발 서버 실행 (빌드 대신 실시간 컴파일)

docker-compose.prod.yml - 전체 구조

```
services:
  frontend: ...
  backend: ...
  db: ...
  nginx: ...

networks:
  app-network:

volumes:
  postgres-data:
```

코드 설명

- services: 4개의 독립 컨테이너 정의
  각각 별도의 프로세스로 실행됨

- networks: 사용자 정의 네트워크
  같은 네트워크 내 컨테이너끼리 서비스명으로 통신
  예: nginx에서 http://backend:8000 으로 접근
  (IP 몰라도 됨 - Docker DNS가 자동 해석)

- volumes: Named Volume (Docker가 관리하는 영구 저장소)
  컨테이너 삭제해도 데이터 유지
  DB 데이터 보존에 필수

docker-compose.prod.yml - Frontend & Backend

```
frontend:
  build:
    context: ./frontend
    dockerfile: Dockerfile.prod
  restart: unless-stopped

backend:
  build: ./backend
  env_file: .env
  depends_on:
    - db
  restart: unless-stopped
```

코드 설명

- context: 빌드 컨텍스트 (파일 참조 기준 경로)
- dockerfile: 기본값(Dockerfile) 대신 다른 파일 지정

- restart 정책:
  - no: 재시작 안 함 (기본값)
  - always: 항상 (수동 중지해도)
  - on-failure: 에러 시만
  - unless-stopped: 수동 중지 전까지 항상

- env_file: .env 파일의 환경변수를 컨테이너에 주입
  민감 정보(DB 비밀번호 등) 관리용

- depends_on: db 컨테이너가 시작된 후 backend 시작
  (단, db가 "준비"됐는지는 보장 안 함)

docker-compose.prod.yml - DB & Nginx

```
db:
  image: postgres:15-alpine
  volumes:
    - postgres-data:/var/lib/postgresql/data
  environment:
    POSTGRES_PASSWORD: ${DB_PASSWORD}
  restart: unless-stopped

nginx:
  image: nginx:alpine
  ports:
    - "80:80"
    - "443:443"
  volumes:
    - ./nginx/nginx.conf:/etc/nginx/nginx.conf
```

코드 설명

- image: 공식 이미지 사용 (빌드 없이 바로 실행)
- volumes (db): DB 데이터를 Named Volume에 저장
  /var/lib/postgresql/data: PostgreSQL 기본 경로
- ${변수명}: 호스트 환경변수 또는 .env에서 값 가져옴
  ⚠ 하드코딩 금지
- ports (nginx): 80=HTTP, 443=HTTPS
  외부에서 접근하는 유일한 진입점
- volumes (nginx): 바인드 마운트
  호스트 설정 파일을 컨테이너에 연결

Docker 볼륨

⚠️ 컨테이너의 특성

컨테이너는 일시적(ephemeral) → 삭제 시 내부 데이터 모두 소멸. DB 컨테이너 재시작/업데이트 시 모든 데이터 손실! 😱

해결책: 볼륨으로 데이터를 컨테이너 외부에 저장

Bind Mount	Named Volume
`./data:/app/data`	`postgres-data:/data`
호스트 경로 직접 지정	Docker가 경로 관리
개발 환경에 적합 (소스 마운트)	프로덕션에 적합 (DB 데이터)
호스트 파일 시스템 의존적	Docker 볼륨 영역 (/var/lib/docker/)

💡 볼륨 확인: docker volume ls | 볼륨 삭제: docker volume rm [볼륨명] (주의: 데이터 삭제!)

환경변수 전략

⚠ .env 파일은 절대 Git에 커밋하지 않음!

.env.example (커밋 O)

```
DB_PASSWORD=your_password_here
SECRET_KEY=your_secret_key
```

.env (커밋 X, 서버에서 직접 생성)

```
DB_PASSWORD=actual_secure_pwd
SECRET_KEY=generated_secret
```

헬스체크 전략

```
backend:
  healthcheck:
    test: ["CMD", "curl", "-f", "http://localhost:8000/health"]
    interval: 30s
    timeout: 10s
    retries: 3
  restart: unless-stopped
```

FastAPI /health 엔드포인트 구현 예시

```
@app.get("/health")
def health_check():
    return {"status": "healthy"}
```

코드 설명

- test: 컨테이너 내부에서 실행할 체크 명령어
  curl -f: HTTP 에러 시 실패 반환
  /health: 앱에서 구현 필요 (단순히 200 OK 반환)

- interval: 체크 주기 (30초마다 호출)
- timeout: 응답 대기 시간 (10초 내 응답 없으면 실패)
- retries: 연속 3번 실패 시 unhealthy 상태로 표시

📌 동작 흐름:
30초마다 체크 → 10초 내 응답 없으면 실패 →
3번 연속 실패 → unhealthy → 자동 재시작

재시작 전략

왜 재시작 정책이 필요한가?

• 앱 크래시, 메모리 부족 등으로 컨테이너가 예기치 않게 종료 • 서버 재부팅 후 서비스 자동 시작 필요 • 24/7 운영되는 서비스의 가용성 보장

정책	동작
`no` (기본값)	재시작 안 함. 개발/테스트용
`always`	항상 재시작. 수동 중지해도 Docker 재시작 시 다시 실행됨
`on-failure`	에러(exit code ≠ 0)로 종료 시만 재시작. 정상 종료는 재시작 안 함
`unless-stopped` ⭐	수동 중지 전까지 항상 재시작. 수동 중지 시 Docker 재시작해도 중지 유지

📌 권장 설정

• 프로덕션 서비스: unless-stopped (대부분의 경우 최적) • 배치/크론 작업: on-failure (정상 종료 시 재실행 방지) • 개발 환경: no (수동 제어)

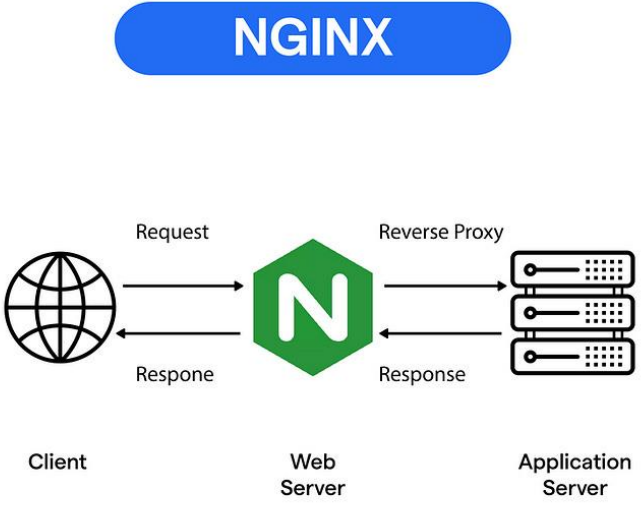
Nginx란?

웹 서버 역할

Client → Nginx → 정적 파일 (HTML, CSS, JS)

리버스 프록시 역할

Client → Nginx → Backend API / Frontend App



Nginx는 왜 사용하는가

1. 정적 파일 서빙 최적화
2. SSL 종료 (HTTPS 처리)
3. 로드 밸런싱 (다중 서버 시)
4. 캐싱 & 압축
5. 보안 (rate limiting, 헤더 설정)

Nginx 효과 예시

❌ 문제점

- **포트 직접 노출**
  8000, 5173 등 → 보안 위험, 포트 스캔 공격
- **백엔드가 SSL 처리**
  Python/Node가 암호화 → 성능 저하
- **정적 파일 서빙 비효율**
  Python/Node가 HTML/CSS/JS 전송 → 느림
- **단일 장애점**
  서버 1대 다운 → 서비스 전체 중단

✅ Nginx 도입 효과

- **80/443만 노출**
  내부 포트 숨김, 공격 표면 최소화
- **SSL 종료 지점 분리**
  Nginx가 암호화 처리 → 백엔드 부담 감소
- **정적 파일 최적화**
  Nginx가 직접 서빙 → 매우 빠름
- **로드밸런싱 준비**
  서버 증설 시 즉시 분산 가능

💡 Nginx = 보안 + 성능 + 확장성의 기반이므로 프로덕션 배포 시 필수

Nginx (리버스 프록시)

Reverse Proxy (Nginx)

`Internet → Proxy → Server`

- 서버를 숨김 (보안)
- 외부에는 Nginx만 노출
- 내부 서버 구조 은닉

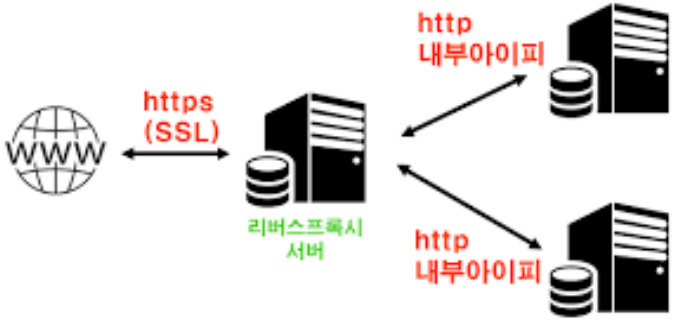
리버스 프록시의 핵심 역할

요청 분배

URL 경로별로 다른 서버로 라우팅

보안 계층

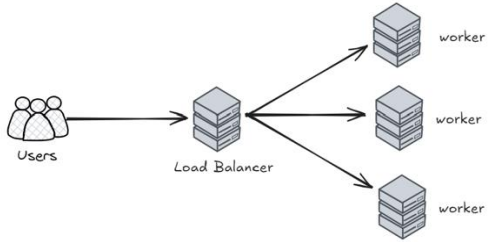
내부 서버 IP/포트 숨김



Nginx (로드 밸런싱)

문제 상황: 사용자 증가 → 단일 서버 한계 → 응답 지연/다운

해결책: 동일 서버 여러 대 + 요청 분배



분배 알고리즘

- Round Robin: 순서대로 돌아가며
- Least Conn: 연결 적은 서버로
- IP Hash: 같은 사용자 → 같은 서버

Nginx 로드밸런싱 설정 예시

```
upstream backend {
    server backend1:8000;
    server backend2:8000;
}

location /api {
    proxy_pass http://backend;
}
```

upstream: 서버 그룹 정의
proxy_pass에서 그룹명 사용
→ 자동으로 요청 분배

Nginx를 이용한 아키텍처 확장

Nginx가 있으면 확장이 쉬워진다

현재: 단일 서버

Nginx
↓
Frontend
Backend
DB

→

확장 1: 수평 스케일링

Nginx (LB)
↓
Backend x 3
↓
DB (별도)

→

확장 2: 마이크로서비스

Nginx (라우터)
/api/users → User서비스
/api/orders → Order서비스
/api/payments → Pay서비스

확장 3: CDN 연동

User → CDN (정적파일) → Nginx → Backend (API만)

사용자에게 빠른 응답. 정적 파일은 CDN이, API만 서버가 처리

Nginx 설정 파일 구조

```
events {
    worker_connections 1024;
}

http {
    include       mime.types;
    default_type  application/octet-stream;

    client_max_body_size 100M;
    server {
        listen 80;
        server_name example.com;
        ...
    }
}
```

코드 설명

- worker_connections 1024
  하나의 worker가 처리할 수 있는 최대 동시 연결 수

- include mime.types
  파일 확장자별 Content-Type 매핑
  .js → application/javascript
  .css → text/css 등

- default_type application/octet-stream
  알 수 없는 확장자의 기본 타입 (바이너리 다운로드)

- client_max_body_size
  클라이언트가 요청 본문(Body)에 담아 보낼 수 있는 최대 크기를 지정

- listen 80: HTTP 포트에서 수신
- server_name: 이 server 블록이 처리할 도메인

리버스 프록시 설정

```
location / {
    proxy_pass http://frontend:80;
    proxy_set_header Host $host;
    proxy_set_header X-Real-IP $remote_addr;
}

location /api {
    proxy_pass http://backend:8000;
    proxy_set_header Host $host;
    proxy_set_header X-Real-IP $remote_addr;
}
```

코드 설명

- location /: /로 시작하는 모든 요청
  → frontend 컨테이너의 80번 포트로 전달
  frontend: Docker 네트워크 내 서비스명 (DNS 자동 해석)

- proxy_set_header Host $host
  원본 요청의 Host 헤더 전달
  없으면 백엔드가 "frontend"를 Host로 인식

- proxy_set_header X-Real-IP $remote_addr
  실제 클라이언트 IP 전달
  없으면 모든 요청이 nginx IP로 보임 (로그/보안 문제)

- location /api: /api로 시작하는 요청은 backend로
  ⚠ 순서 중요: 더 구체적인 경로가 먼저 매칭

KMAP용 Nginx 설정 (예시)

```
server {
    listen 80;
    server_name kmap.example.com;

    location / {
        proxy_pass http://frontend:80;
    }

    location /api {
        rewrite ^/api(.*)$ $1 break;
        proxy_pass http://backend:8000;
    }
}
```

rewrite 설명

- rewrite ^/api(.*)$ $1 break;
  URL 변환 규칙

- ^/api(.*)$
  /api로 시작하는 URL에서 /api 이후 부분을 캡처

- $1: 캡처된 부분 (괄호 안)
- break: 변환 후 다른 rewrite 규칙 적용 안 함

📌 변환 예시:
/api/users → /users (backend는 /users로 받음)
/api/health → /health

✅ 왜 필요한가:
FastAPI가 /api/users가 아닌 /users로
라우팅 설계된 경우 URL을 맞춰줌

정적 파일 캐싱

```
location ~* \.(js|css|png|jpg|jpeg|gif|ico|svg)$ {
    expires 1y;
    add_header Cache-Control "public, immutable";
}
```

코드 설명

- ~*: 대소문자 구분 없는 정규식 매칭
  해당 확장자로 끝나는 모든 요청에 적용

- expires 1y: 브라우저 캐시 만료 시간 1년
  응답 헤더에 Expires 추가

- Cache-Control "public, immutable"
  - public: CDN 등 중간 캐시도 저장 가능
  - immutable: 파일이 절대 변경되지 않음
    (새로고침해도 재검증 안 함)

⚠ 주의: 파일 변경 시 파일명도 변경 필요
예: app.js → app.abc123.js
Vite/Webpack이 자동으로 처리해줌

Gzip 압축 설정

```
http {
    gzip on;
    gzip_types text/plain text/css application/json
               application/javascript text/xml;
    gzip_min_length 1000;
    gzip_comp_level 6;
}
```

코드 설명

- gzip on: Gzip 압축 활성화
- gzip_types: 압축할 MIME 타입 지정
  이미지(png, jpg)는 이미 압축되어 있어 제외
- gzip_min_length 1000
  1000바이트 미만은 압축 안 함
  작은 파일은 압축 오버헤드가 더 클 수 있음
- gzip_comp_level 6
  압축 레벨 (1-9). 6이 속도/압축률 균형점
  높을수록 CPU↑, 압축률↑

✅ 효과: JS/CSS 파일 70-80% 용량 감소
→ 페이지 로딩 속도 향상

로그 설정

```
http {
    access_log /var/log/nginx/access.log;
    error_log  /var/log/nginx/error.log;
}

# 실시간 로그 확인
docker logs -f nginx_container
tail -f /var/log/nginx/access.log
```

코드 설명

- access_log: 모든 요청 기록
  IP, 시간, URL, 상태코드, 응답크기
  트래픽 분석, 문제 추적용

- error_log: 에러만 기록
  설정 오류, 백엔드 연결 실패 등
  디버깅 필수

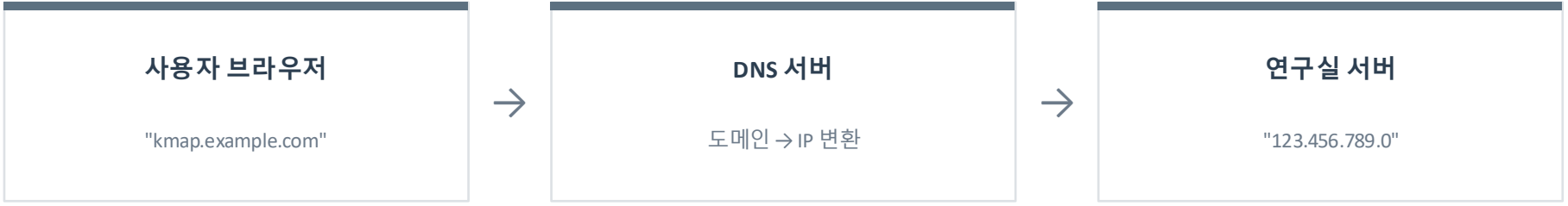
- docker logs -f
  Docker 컨테이너 로그 실시간 확인
  -f: follow (실시간 스트리밍)

- tail -f
  파일 로그 실시간 확인
  (볼륨 마운트한 경우)

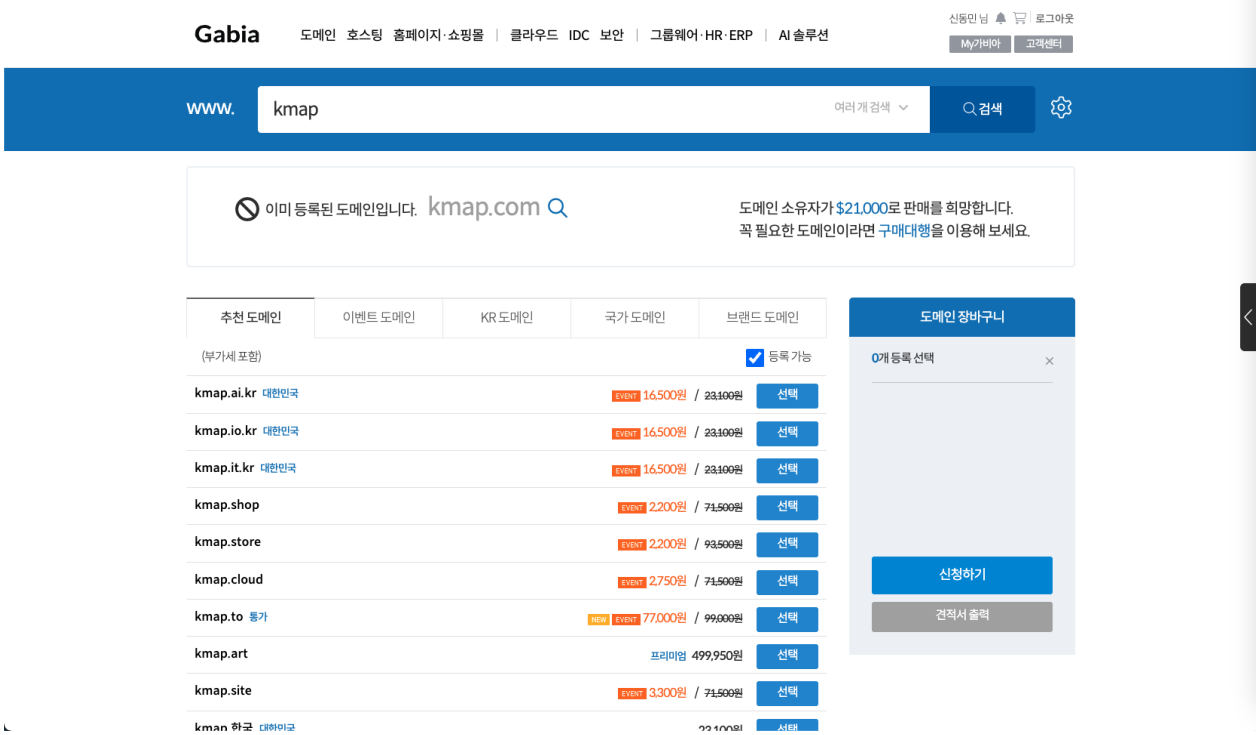
Nginx 체크리스트

- ☑ 리버스 프록시 설정 완료
- ☑ Frontend/Backend 라우팅 동작
- ☑ Gzip 압축 활성화
- ☑ 정적 파일 캐싱 설정

도메인 원리



도메인 구매

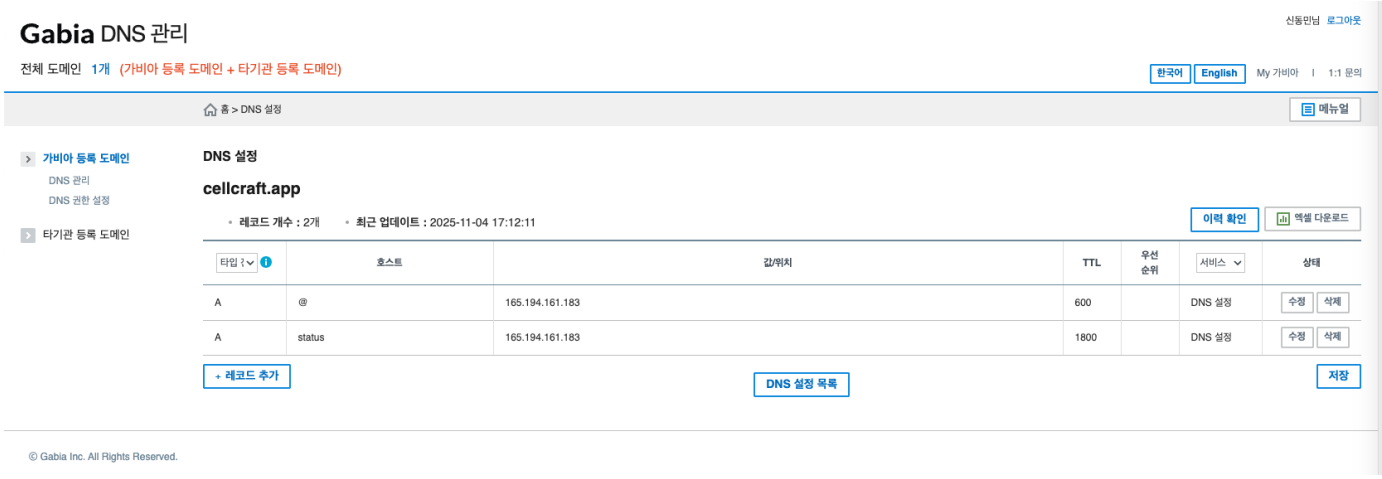

Gabia
도메인 호스팅 홈페이지·쇼핑몰 | 클라우드 IDC 보안 | 그룹웨어·HR·ERP | AI 솔루션
신동민님 | 로그아웃
My가비아 고객센터
www.
kmap
여러개검색
검색
이미 등록된 도메인입니다. kmap.com
도메인 소유자가 $21,000로 판매를 희망합니다.
꼭 필요한 도메인이라면 구매대행을 이용해 보세요.
추천 도메인
이벤트 도메인
KR 도메인
국가 도메인
브랜드 도메인
(부가세 포함)
등록 가능
kmap.ai.kr 대한민국 EVENT 16,500원 / 23,100원 선택
kmap.io.kr 대한민국 EVENT 16,500원 / 23,100원 선택
kmap.it.kr 대한민국 EVENT 16,500원 / 23,100원 선택
kmap.shop EVENT 2,200원 / 71,500원 선택
kmap.store EVENT 2,200원 / 93,500원 선택
kmap.cloud EVENT 2,750원 / 71,500원 선택
kmap.to 통가 NEW EVENT 77,000원 / 99,000원 선택
kmap.art 프리미엄 499,950원 선택
kmap.site EVENT 3,300원 / 71,500원 선택
도메인 장바구니
0개 등록 선택
신청하기
견적서 출력

DNS 레코드 종류

A	도메인 → IPv4 주소 `example.com → 123.456.789.0`
AAAA	도메인 → IPv6 주소
CNAME	도메인 → 다른 도메인 (별칭) `www.example.com → example.com`
TXT	텍스트 정보 (인증용)

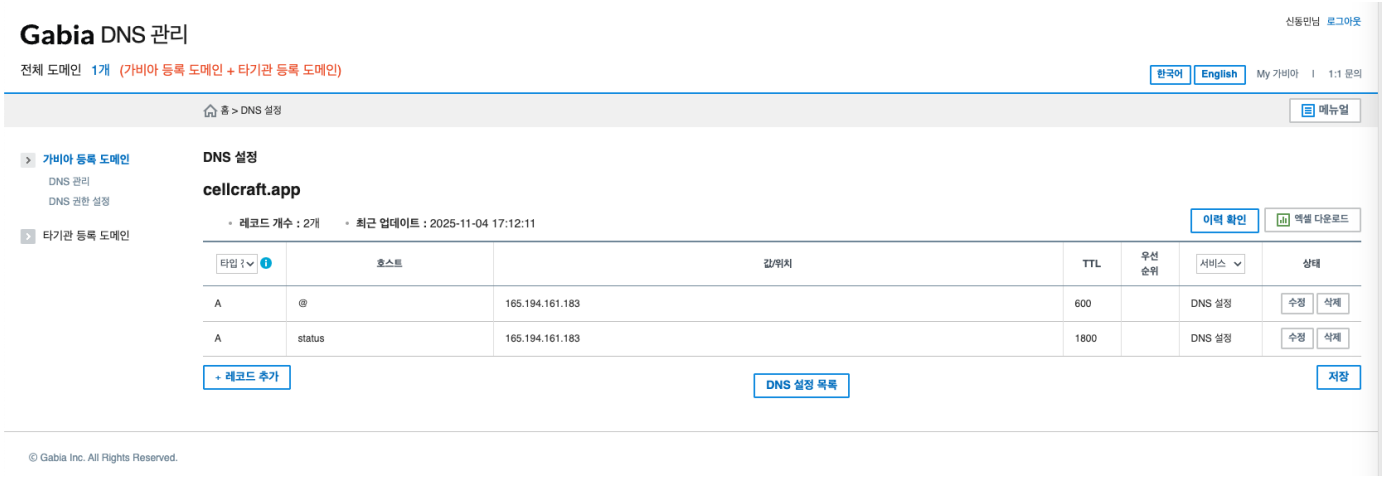
DNS 레코드 설정 예시

호스트	@ (루트 도메인)
타입	A
값	[서버 고정 IP]
TTL	600 (10분)



DNS 레코드 설정 예시

호스트	status
타입	A
값	[서버 고정 IP]
TTL	1800 (30분)



DNS 전파 확인 예시

명령어로 확인

```
nslookup example.com
dig example.com
```

```
→  ~ nslookup status.cellcraft.app

Server:         165.194.128.1
Address:        165.194.128.1#53

Non-authoritative answer:
Name:   status.cellcraft.app
Address: 165.194.161.183
```

```
→  ~ dig status.cellcraft.app


; <<>> DiG 9.10.6 <<>> status.cellcraft.app
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 11802
;; flags: qr rd ra; QUERY: 1, ANSWER: 1, AUTHORITY: 3, ADDITIONAL: 5

;; OPT PSEUDOSECTION:
; EDNS: version: 0, flags:; udp: 1232
;; QUESTION SECTION:
;status.cellcraft.app.          IN      A

;; ANSWER SECTION:
status.cellcraft.app.   1636    IN      A       165.194.161.183

;; AUTHORITY SECTION:
cellcraft.app.          8412    IN      NS      ns.gabia.co.kr.
cellcraft.app.          8412    IN      NS      ns1.gabia.co.kr.
cellcraft.app.          8412    IN      NS      ns.gabia.net.

;; ADDITIONAL SECTION:
ns.gabia.co.kr.         18458   IN      A       43.201.170.100
ns1.gabia.co.kr.        71231   IN      A       20.200.205.248
ns.gabia.net.           32292   IN      A       121.78.117.39
ns.gabia.net.           32292   IN      A       211.234.124.90

;; Query time: 2 msec
;; SERVER: 165.194.128.1#53(165.194.128.1)
;; WHEN: Thu Jan 29 11:50:41 KST 2026
;; MSG SIZE  rcvd: 201
```

도메인 체크리스트

- ☑ 도메인 구매 완료
- ☑ A 레코드 설정 (도메인 → 서버 IP)
- ☑ DNS 전파 확인 (nslookup)
- ☑ 브라우저에서 도메인으로 접속 가능

HTTPS를 사용하는 이유

1. 데이터 암호화 (비밀번호, 개인정보 보호)
2. 신원 인증 (피싱 사이트 방지)
3. SEO 가산점 (Google 검색 순위)
4. 브라우저 경고 방지 ("안전하지 않음")
5. 최신 웹 기능 사용 (Service Worker 등)

HTTP의 위험성

HTTP = 평문 전송 (암호화 없음)

🕵️ 시나리오 1: 패킷 스니핑

1. 카페 와이파이에서 로그인
2. 옆 테이블 해커가 Wireshark 실행
3. HTTP 패킷 캡처
4. 비밀번호 평문으로 노출!

→ ID: user@email.com
→ PW: mypassword123 😱

🦹 시나리오 2: 중간자 공격 (MITM)

1. 해커가 중간에서 요청 가로채기
2. 데이터 변조 (금액, 계좌번호 등)
3. 사용자/서버 모두 모르는 상태

→ 송금 10만원 → 100만원으로 변조
→ 내 계좌 → 해커 계좌로 변조

"내 서비스는 작은데..."

- 사용자 신뢰 문제: 브라우저에 "안전하지 않음" 경고 → 이탈률 증가
- 검색 엔진 불이익: Google은 HTTPS 사이트를 우선 순위로
- 습관 형성: 잘 모르더라도 보안 의식을 갖추는 것이 중요

HTTPS 동작 원리

TLS 핸드셰이크 (간략)

1 **연결 요청**
Client → Server
"HTTPS로 연결할게요"

→

2 **인증서 전달**
Server → Client
인증서(공개키) 전달

→

3 **인증서 검증**
Client가 CA 통해
"진짜 서버 맞네!" 확인

→

4 **암호화 통신**
대칭키 생성
이후 모든 통신 암호화

인증서의 역할

"이 서버가 진짜 example.com이다"

- 신뢰할 수 있는 CA(인증기관)가 발급
- 브라우저가 CA 목록을 내장
- 위조 불가능

Let's Encrypt

✅ 무료 인증서 발급 CA

- 비영리 단체 운영
- 90일 유효 (자동 갱신)
- 전 세계 수억 개 사이트에서 사용

Let's Encrypt & Certbot

Let's Encrypt

- 무료 SSL 인증서 발급 기관
- 90일 유효 (자동 갱신 가능)

Certbot

- Let's Encrypt 인증서 발급 도구
- Nginx 설정 자동 수정 가능

Certbot 설치

```
# Ubuntu (apt)
sudo apt install certbot python3-certbot-nginx

# Snap으로 설치 (권장)
sudo snap install --classic certbot
sudo ln -s /snap/bin/certbot /usr/bin/certbot
```

코드 설명

- apt 저장소 버전: 안정적이지만 구버전일 수 있음
- python3-certbot-nginx: Nginx 자동 설정 플러그인

- Snap 패키지 (권장)
  최신 버전 유지됨, Certbot 공식 권장
  --classic: 시스템 전체 접근 권한 필요

- ln -s: 심볼릭 링크 생성
  certbot 명령어를 어디서든 실행 가능하게

인증서 발급

```
# Nginx 플러그인으로 자동 설정
sudo certbot --nginx -d example.com -d www.example.com

# 이메일 입력, 약관 동의 후 발급 완료
```

⚠ 실패 원인: DNS 미연결, 80포트 차단, Nginx 미실행

발급 과정

1. 이메일 입력 (만료 알림용)
2. 약관 동의
3. Let's Encrypt 서버가 도메인 소유권 확인
   (80포트로 검증 파일 요청)
4. 인증서 발급 및 Nginx 설정 자동 수정

Nginx SSL 설정

```
server {
    listen 443 ssl;
    server_name example.com;

    ssl_certificate /etc/letsencrypt/live/example.com/fullchain.pem;
    ssl_certificate_key
/etc/letsencrypt/live/example.com/privkey.pem;

    # ... 기존 설정
}
```

코드 설명

- listen 443 ssl: 443 포트에서 SSL/TLS 사용
- ssl_certificate (fullchain.pem)
  인증서 체인 파일 (내 인증서 + 중간 인증서)
  브라우저가 신뢰 체인 검증에 사용
- ssl_certificate_key (privkey.pem)
  개인 키 파일 (절대 외부 노출 금지!)
  암호화/복호화에 사용

📁 파일 위치:
/etc/letsencrypt/live/도메인명/

HTTP → HTTPS 리다이렉트

```
server {
    listen 80;
    server_name example.com;

    return 301 https://$server_name$request_uri;
}
```

코드 설명

- listen 80: HTTP 요청을 받는 server 블록

- return 301: 영구 이동 (Permanent Redirect)
  브라우저/검색엔진이 기억
  (302는 임시 이동 - 테스트용)

- $server_name: 요청받은 도메인명
- $request_uri: 요청 경로 + 쿼리스트링

📌 동작 예시:
http://example.com/page?id=1
→ https://example.com/page?id=1

인증서 자동 갱신

```
# 갱신 테스트
sudo certbot renew --dry-run

# cron에 자동 등록됨 (확인)
sudo systemctl status certbot.timer

# 수동 갱신 (필요시)
sudo certbot renew
```

코드 설명

- --dry-run: 실제 갱신 없이 테스트만
  성공 메시지 나오면 자동 갱신 정상 작동

- certbot.timer: 자동 갱신 타이머
  하루 2번 자동 실행
  만료 30일 전부터 갱신 시도

- certbot renew: 수동 갱신 (긴급 상황 시)
  만료 30일 이상 남았으면 갱신 안 됨
  (--force-renewal로 강제)

📌 Let's Encrypt: 유효기간 90일, 자동 갱신 권장

CellCraft 예시

```
server {
    listen 443 ssl;
    server_name cellcraft.app;

    ssl_certificate /etc/letsencrypt/live/cellcraft.app/fullchain.pem;
    ssl_certificate_key /etc/letsencrypt/live/cellcraft.app/privkey.pem;
    include /etc/letsencrypt/options-ssl-nginx.conf;
    ssl_dhparam /etc/letsencrypt/ssl-dhparams.pem;

    client_max_body_size 10G;

    location / {
        proxy_pass http://127.0.0.1:8080/;
        proxy_set_header Host $host;
        proxy_set_header X-Real-IP $remote_addr;
        proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
        proxy_set_header X-Forwarded-Proto $scheme;
        proxy_redirect off;
    }
}
```

CellCraft 예시

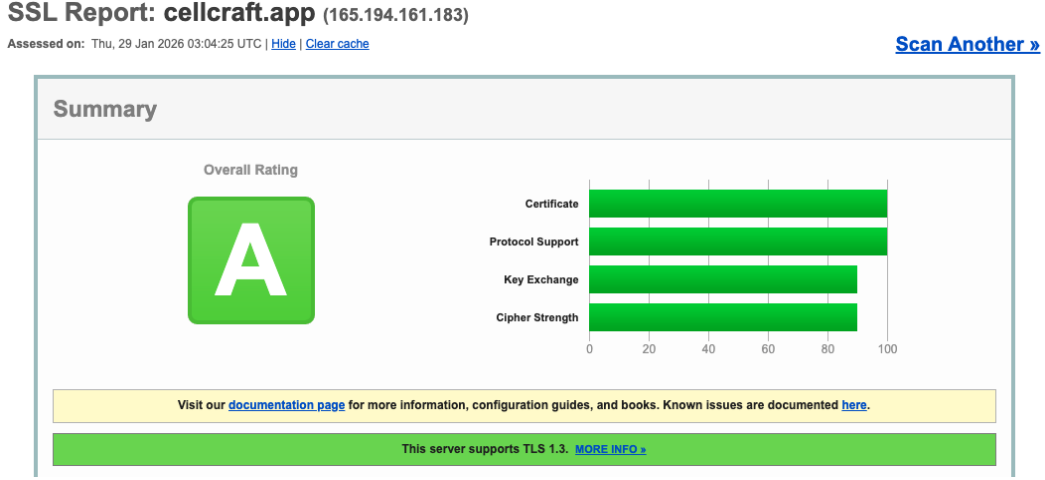
```
# status.cellcraft.app HTTP → HTTPS 리다이렉트
server {
    listen 80;
    server_name status.cellcraft.app;

    location /.well-known/acme-challenge/ {
        root /var/www/certbot;
    }

    location / {
        return 301 https://$host$request_uri;
    }
}

# status.cellcraft.app HTTPS (Uptime Kuma Status Page)
server {
    listen 443 ssl;
    server_name status.cellcraft.app;

    ssl_certificate /etc/letsencrypt/live/cellcraft.app/fullchain.pem;
    ssl_certificate_key /etc/letsencrypt/live/cellcraft.app/privkey.pem;
    include /etc/letsencrypt/options-ssl-nginx.conf;
    ssl_dhparam /etc/letsencrypt/ssl-dhparams.pem;

    client_max_body_size 10M;

    # 루트 경로 접속 시 상태 페이지로 리다이렉트
    location = / {
        return 301 https://status.cellcraft.app/status/cellcraft;
    }

    # 모든 경로는 Uptime Kuma로 프록시
    location / {
        proxy_pass http://127.0.0.1:3001;
        proxy_set_header Host $host;
        proxy_set_header X-Real-IP $remote_addr;
        proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
        proxy_set_header X-Forwarded-Proto $scheme;

        # WebSocket support (필수)
        proxy_http_version 1.1;
        proxy_set_header Upgrade $http_upgrade;
        proxy_set_header Connection "upgrade";

        proxy_redirect off;
    }
}
```

SSL 설정 검증

온라인 테스트

https://www.ssllabs.com/ssltest/

목표: A 등급 이상

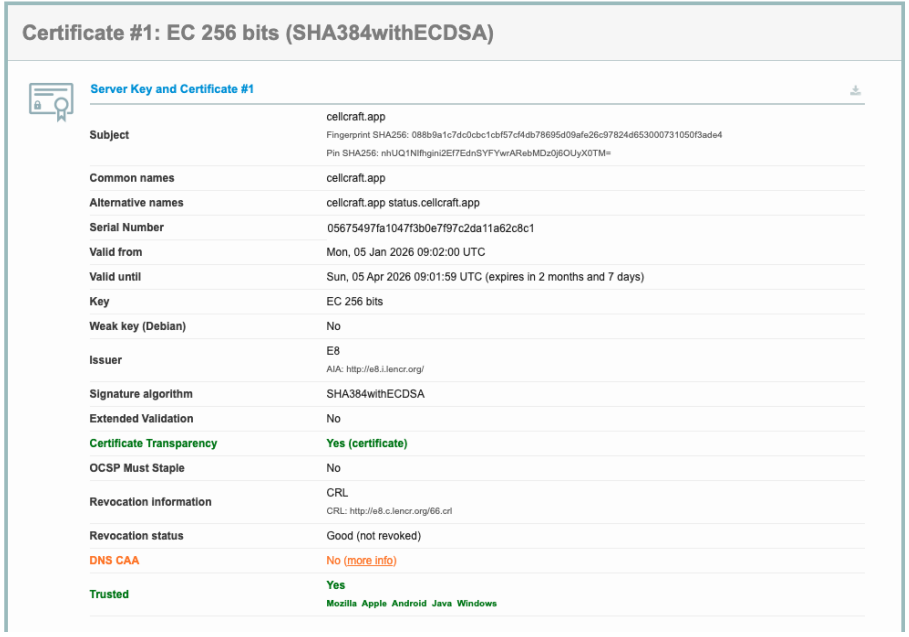


SSL 설정 검증

온라인 테스트

https://www.ssllabs.com/ssltest/

목표: A 등급 이상



Certificate #1: EC 256 bits (SHA384withECDSA)

Server Key and Certificate #1

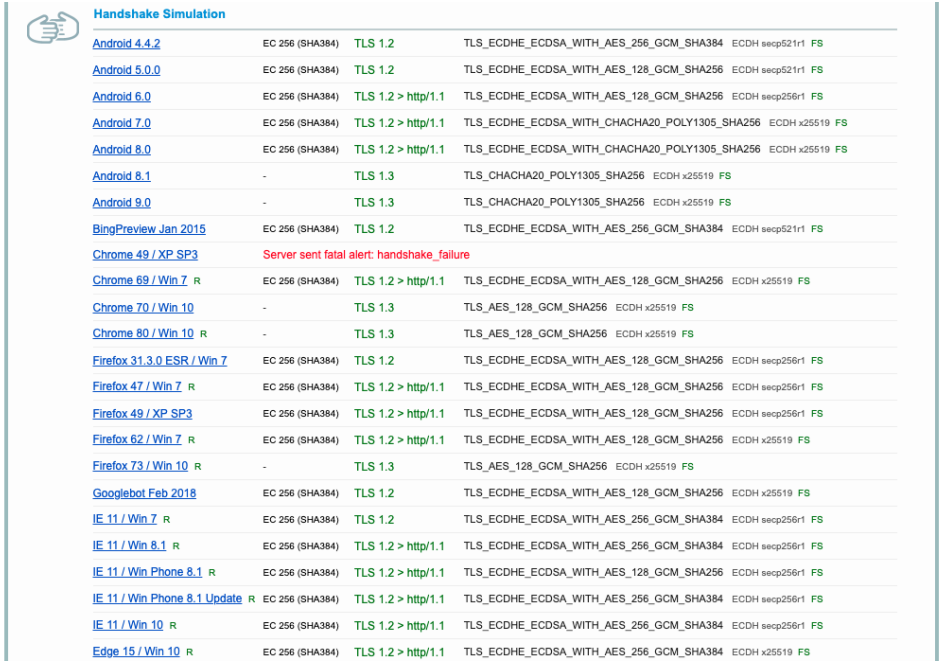
Subject	cellcraft.app Fingerprint SHA256: 088b9a1c7dc0cbc1cbf57cf4db78695d09afe26c97824d653000731050f3ade4 Pin SHA256: nhUQ1NIfhgini2Ef7EdnSYFYwrARebMDz0j6OUyX0TM=
Common names	cellcraft.app
Alternative names	cellcraft.app status.cellcraft.app
Serial Number	05675497fa1047f3b0e7f97c2da11a62c8c1
Valid from	Mon, 05 Jan 2026 09:02:00 UTC
Valid until	Sun, 05 Apr 2026 09:01:59 UTC (expires in 2 months and 7 days)
Key	EC 256 bits
Weak key (Debian)	No
Issuer	E8 AIA: http://e8.i.lencr.org/
Signature algorithm	SHA384withECDSA
Extended Validation	No
Certificate Transparency	Yes (certificate)
OCSP Must Staple	No
Revocation information	CRL CRL: http://e8.c.lencr.org/66.crl
Revocation status	Good (not revoked)
DNS CAA	No (more info)
Trusted	Yes Mozilla Apple Android Java Windows

SSL 설정 검증

온라인 테스트

https://www.ssllabs.com/ssltest/

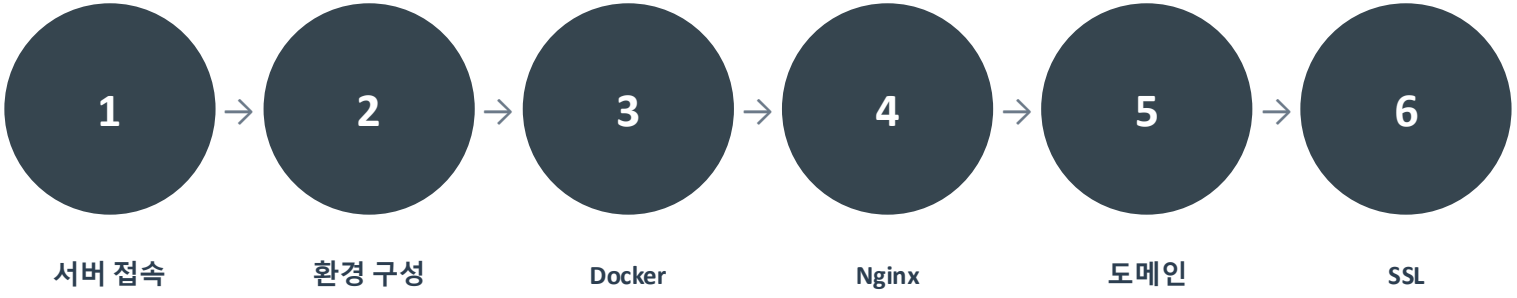
목표: A 등급 이상



Handshake Simulation

Android 4.4.2	EC 256 (SHA384)	TLS 1.2	TLS_ECDHE_ECDSA_WITH_AES_256_GCM_SHA384 ECDH secp521r1 FS
Android 5.0.0	EC 256 (SHA384)	TLS 1.2	TLS_ECDHE_ECDSA_WITH_AES_128_GCM_SHA256 ECDH secp521r1 FS
Android 6.0	EC 256 (SHA384)	TLS 1.2 > http/1.1	TLS_ECDHE_ECDSA_WITH_AES_128_GCM_SHA256 ECDH secp256r1 FS
Android 7.0	EC 256 (SHA384)	TLS 1.2 > http/1.1	TLS_ECDHE_ECDSA_WITH_CHACHA20_POLY1305_SHA256 ECDH x25519 FS
Android 8.0	EC 256 (SHA384)	TLS 1.2 > http/1.1	TLS_ECDHE_ECDSA_WITH_CHACHA20_POLY1305_SHA256 ECDH x25519 FS
Android 8.1	-	TLS 1.3	TLS_CHACHA20_POLY1305_SHA256 ECDH x25519 FS
Android 9.0	-	TLS 1.3	TLS_CHACHA20_POLY1305_SHA256 ECDH x25519 FS
BingPreview Jan 2015	EC 256 (SHA384)	TLS 1.2	TLS_ECDHE_ECDSA_WITH_AES_256_GCM_SHA384 ECDH secp521r1 FS
Chrome 49 / XP SP3	Server sent fatal alert: handshake_failure		
Chrome 69 / Win 7 R	EC 256 (SHA384)	TLS 1.2 > http/1.1	TLS_ECDHE_ECDSA_WITH_AES_128_GCM_SHA256 ECDH x25519 FS
Chrome 70 / Win 10	-	TLS 1.3	TLS_AES_128_GCM_SHA256 ECDH x25519 FS
Chrome 80 / Win 10 R	-	TLS 1.3	TLS_AES_128_GCM_SHA256 ECDH x25519 FS
Firefox 31.3.0 ESR / Win 7	EC 256 (SHA384)	TLS 1.2	TLS_ECDHE_ECDSA_WITH_AES_128_GCM_SHA256 ECDH secp256r1 FS
Firefox 47 / Win 7 R	EC 256 (SHA384)	TLS 1.2 > http/1.1	TLS_ECDHE_ECDSA_WITH_AES_128_GCM_SHA256 ECDH secp256r1 FS
Firefox 49 / XP SP3	EC 256 (SHA384)	TLS 1.2 > http/1.1	TLS_ECDHE_ECDSA_WITH_AES_128_GCM_SHA256 ECDH secp256r1 FS
Firefox 62 / Win 7 R	EC 256 (SHA384)	TLS 1.2 > http/1.1	TLS_ECDHE_ECDSA_WITH_AES_128_GCM_SHA256 ECDH x25519 FS
Firefox 73 / Win 10 R	-	TLS 1.3	TLS_AES_128_GCM_SHA256 ECDH x25519 FS
Googlebot Feb 2018	EC 256 (SHA384)	TLS 1.2	TLS_ECDHE_ECDSA_WITH_AES_128_GCM_SHA256 ECDH x25519 FS
IE 11 / Win 7 R	EC 256 (SHA384)	TLS 1.2	TLS_ECDHE_ECDSA_WITH_AES_256_GCM_SHA384 ECDH secp256r1 FS
IE 11 / Win 8.1 R	EC 256 (SHA384)	TLS 1.2 > http/1.1	TLS_ECDHE_ECDSA_WITH_AES_256_GCM_SHA384 ECDH secp256r1 FS
IE 11 / Win Phone 8.1 R	EC 256 (SHA384)	TLS 1.2 > http/1.1	TLS_ECDHE_ECDSA_WITH_AES_128_GCM_SHA256 ECDH secp256r1 FS
IE 11 / Win Phone 8.1 Update R	EC 256 (SHA384)	TLS 1.2 > http/1.1	TLS_ECDHE_ECDSA_WITH_AES_256_GCM_SHA384 ECDH secp256r1 FS
IE 11 / Win 10 R	EC 256 (SHA384)	TLS 1.2 > http/1.1	TLS_ECDHE_ECDSA_WITH_AES_256_GCM_SHA384 ECDH secp256r1 FS
Edge 15 / Win 10 R	EC 256 (SHA384)	TLS 1.2 > http/1.1	TLS_ECDHE_ECDSA_WITH_AES_256_GCM_SHA384 ECDH x25519 FS

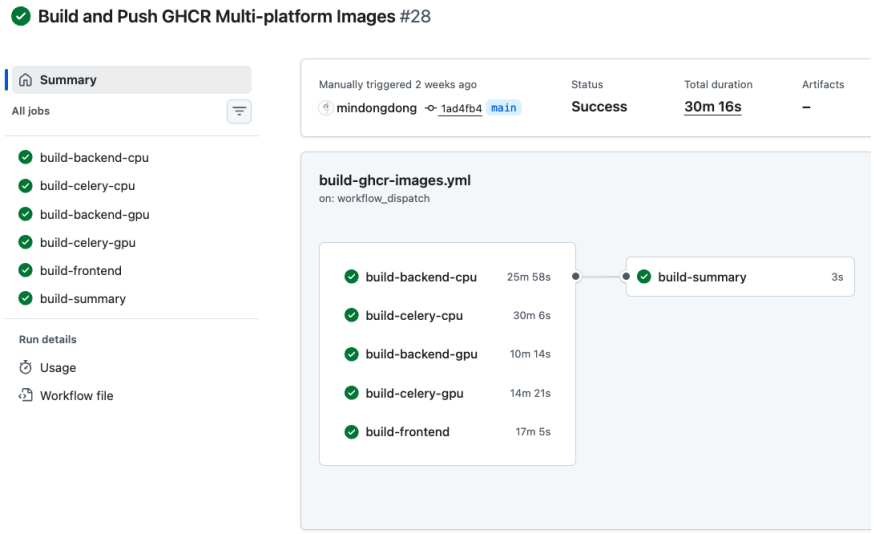
배포 프로세스

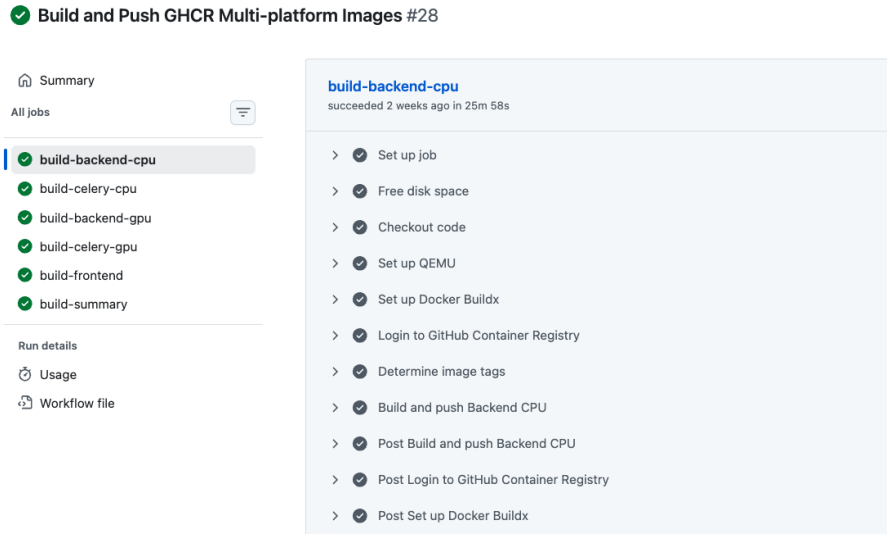


흔한 오류

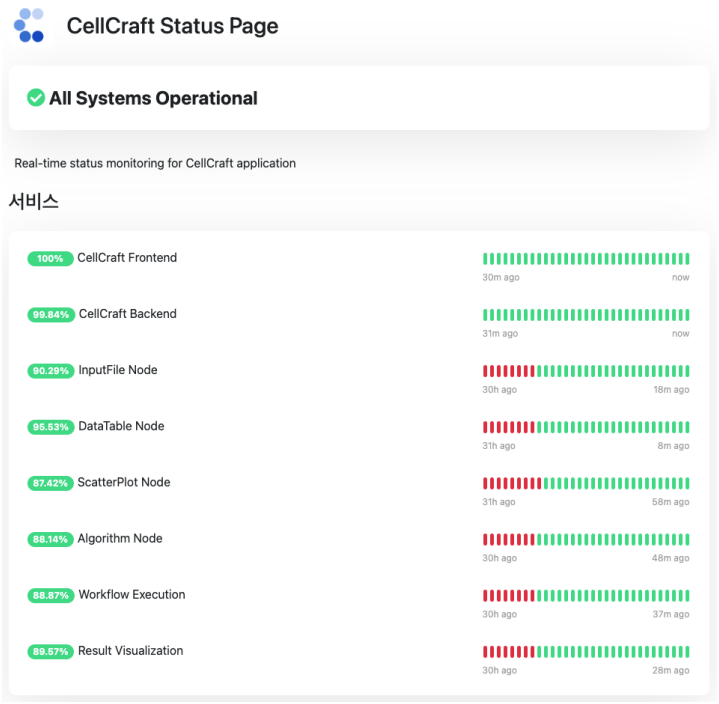
❌ **502 Bad Gateway**	→ 도커 컨테이너 실행 확인
❌ **Connection Refused**	→ 방화벽, 포트 확인
❌ **SSL 발급 실패**	→ DNS 전파 완료 확인, 80포트 오픈 여부 확인
❌ **컨테이너 계속 재시작**	→ docker logs로 에러 확인

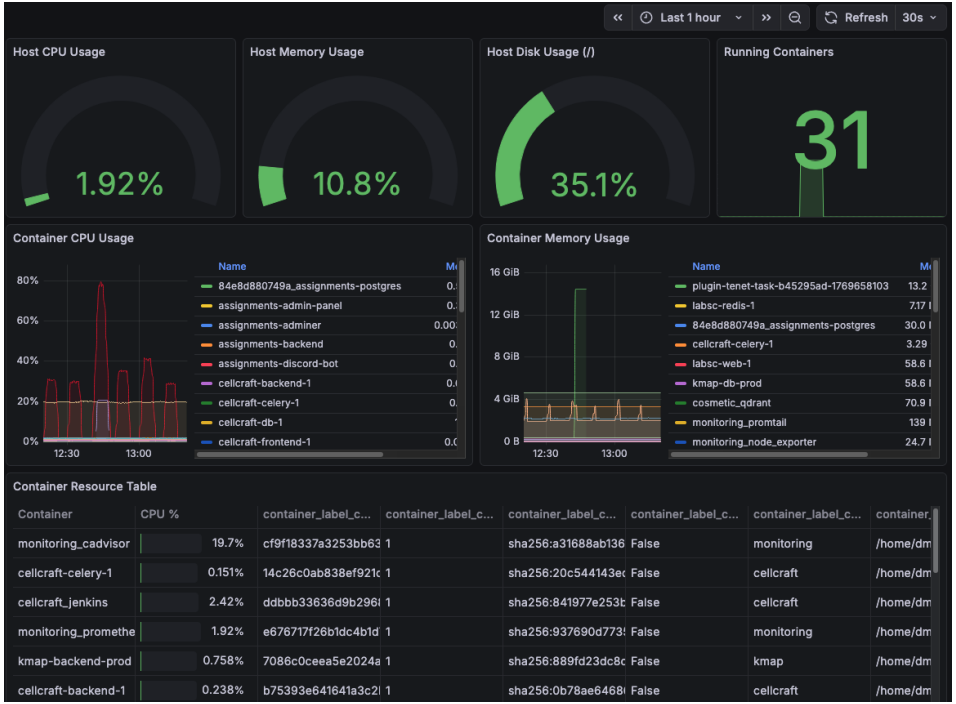
다음 회차 (CI/CD)


Build and Push GHCR Multi-platform Images #28
Summary
All jobs
build-backend-cpu
build-celery-cpu
build-backend-gpu
build-celery-gpu
build-frontend
build-summary
Run details
Usage
Workflow file
Manually triggered 2 weeks ago
mindongdong 1ad4fb4 main
Status
Success
Total duration
30m 16s
Artifacts
–
build-ghcr-images.yml
on: workflow_dispatch
build-backend-cpu 25m 58s
build-celery-cpu 30m 6s
build-backend-gpu 10m 14s
build-celery-gpu 14m 21s
build-frontend 17m 5s
build-summary 3s


Build and Push GHCR Multi-platform Images #28
Summary
All jobs
build-backend-cpu
build-celery-cpu
build-backend-gpu
build-celery-gpu
build-frontend
build-summary
Run details
Usage
Workflow file
build-backend-cpu
succeeded 2 weeks ago in 25m 58s
Set up job
Free disk space
Checkout code
Set up QEMU
Set up Docker Buildx
Login to GitHub Container Registry
Determine image tags
Build and push Backend CPU
Post Build and push Backend CPU
Post Login to GitHub Container Registry
Post Set up Docker Buildx

다음 회차 (CI/CD)


CellCraft Status Page
All Systems Operational
Real-time status monitoring for CellCraft application
서비스
100% CellCraft Frontend
30m ago
now
99.84% CellCraft Backend
31m ago
now
90.29% InputFile Node
30h ago
18m ago
95.53% DataTable Node
31h ago
8m ago
87.42% ScatterPlot Node
31h ago
58m ago
88.14% Algorithm Node
30h ago
48m ago
88.87% Workflow Execution
30h ago
37m ago
89.57% Result Visualization
30h ago
28m ago


Last 1 hour
Refresh
30s
Host CPU Usage
1.92%
Host Memory Usage
10.8%
Host Disk Usage (/)
35.1%
Running Containers
31
Container CPU Usage
80%
60%
40%
20%
0%
12:30
13:00
Name
84e8d880749a_assignments-postgres
assignments-admin-panel
assignments-adminer
assignments-backend
assignments-discord-bot
cellcraft-backend-1
cellcraft-celery-1
cellcraft-db-1
cellcraft-frontend-1
Container Memory Usage
16 GiB
12 GiB
8 GiB
4 GiB
0 B
12:30
13:00
Name
plugin-tenet-task-b45295ad-1769658103
labsc-redis-1
84e8d880749a_assignments-postgres
cellcraft-celery-1
labsc-web-1
kmap-db-prod
cosmetic_qdrant
monitoring_promtail
monitoring_node_exporter
Container Resource Table
Container
CPU %
container_label_c...
monitoring_cadvisor
19.7%
cellcraft-celery-1
0.151%
cellcraft_jenkins
2.42%
monitoring_promethe
1.92%
kmap-backend-prod
0.758%
cellcraft-backend-1
0.238%
False
monitoring
cellcraft
/home/dm

Q&A

다음 회차 안내

🗓️ 2/3 (화) – CI/CD