

프로젝트 빌드 및 배포 문서

서비스 및 컨테이너 포트 매핑 정리

서비스명	컨테이너명	내부 포트	역할 설명
Frontend	frontend	80	클라이언트(React/Vue 등)의 정적 페이지 제공
Backend API	backend	8080	Spring, Django 등 백엔드 API 처리
Algorithm API	algorithm	8000	추천 알고리즘 처리 (예: Python FastAPI 등)
Nginx Proxy	nginx	80:80 , 443:443	클라이언트 요청을 받아 각 서비스로 라우팅
Jenkins	jenkins	8080 , 50000	CI/CD 자동화 서버 (웹 UI는 8080, 에이전트는 50000 포트 사용)
Prometheus	prometheus	9090	시계열 기반 모니터링 시스템, 메트릭 수집 및 시각화
Node Exporter	node-exporter	9100	호스트 시스템의 CPU, 메모리, 디스크 등 메트릭 수집
cAdvisor	cadvisor	8080	컨테이너 리소스 사용량 및 성능 모니터링 도구
Grafana	grafana	3000	다양한 시각화 대시보드를 제공하는 도구 (Prometheus 등과 연동)
Elasticsearch	elasticsearch	9200 , 9300	로그 저장 및 검색 엔진, 단일 노드 클러스터
Logstash	logstash	5000 , 5044 , 9600	로그 수집 및 파싱 파이프라인 도구 (Elasticsearch에 전달)
Kibana	kibana	5601	Elasticsearch 시각화 대시보드 툴
Filebeat	filebeat	-	로그 수집 에이전트, 컨테이너/시스템 로그를 Logstash에 전달

사용된 기술 스택 및 버전

구성 요소	기술 스택	버전	
-------	-------	----	--

Frontend	React	Node.js 22 + VITE	
Backend	Spring Boot	Spring Boot + OpenJDK 17	
Algorithm	FastAPI	FastAPI 0.115 / python 3.11	
Proxy	Nginx	1.24	
CI/CD	Jenkins	2.426 LTS	
Monitoring	Prometheus	2.51.0	
	Node Exporter	1.7.0	
	cAdvisor	0.49.1	
	Grafana	10.2.3	
Logging	Elasticsearch	8.12.1	
	Logstash	8.12.1	
	Kibana	8.12.1	
	Filebeat	8.12.1	
Database	MySQL (AWS RDS). REDIS		

환경 변수 설정 정보

1. 백엔드 서버 환경변수

`/home/ubuntu/knockknock/backend/.env`

```
# Database 설정
DB_URL=
DB_USERNAME=
DB_PASSWORD=
DB_DRIVER=

# JWT 시크릿 키
JWT_SECRET=

# Redis 설정
REDIS_HOST=
REDIS_PORT=
```

```
# AWS 설정
AWS_ACCESS_KEY=
AWS_SECRET_KEY=
AWS_REGION=
AWS_S3_BUCKET=

# Elasticsearch 설정
ELASTICSEARCH_URI=
ELASTICSEARCH_USERNAME=
ELASTICSEARCH_PASSWORD=
```

2. 알고리즘 서버 환경변수

`/home/ubuntu/knockknock/algorithm/.env`

```
# Database 설정
DB_HOST=
DB_PORT=
DB_NAME=
DB_USERNAME=
DB_PASSWORD=

# Redis 설정
REDIS_HOST=
REDIS_PORT=
REDIS_DATABASE=

# 외부 API 키
PINECONE_API_KEY=
UPSTAGE_API_KEY=
OPENAI_API_KEY=
GOOGLE_API_KEY=
```

GitLab CI/CD 파이프라인 순서

1. Push / Merge

- **설명:** 개발자가 GitLab에 코드를 푸시하거나, Merge Request(MR)를 생성합니다. 이 단계는 CI/CD 파이프라인을 트리거하는 시작점이 됩니다.

2. Webhook 전송

- **설명:** GitLab은 Push 또는 Merge 이벤트를 감지하고 설정된 Webhook을 통해 파이프라인 트리거를 외부 서비스나 다른 시스템에 전달합니다.

3. GitLab Checkout

- **설명:** CI/CD 파이프라인이 시작되면 GitLab Runner가 저장소에서 최신 코드를 체크아웃하여 작업을 시작합니다.

4. Test & Build

- **설명:** 프로젝트의 테스트를 실행하고, 필요하다면 빌드를 수행합니다. 예를 들어, 유닛 테스트, 통합 테스트 및 빌드 스크립트를 실행합니다.

5. Docker Image Build

- **설명:** 테스트 및 빌드가 완료된 후, Dockerfile을 기반으로 애플리케이션의 Docker 이미지를 빌드합니다. 이 이미지에는 애플리케이션 및 필요한 종속성이 포함됩니다.

6. Docker Image Push

- **설명:** 빌드된 Docker 이미지를 Docker Registry(예: Docker Hub, GitLab Container Registry, AWS ECR 등)에 푸시합니다. 이를 통해 다른 시스템에서 이미지를 가져와 사용할 수 있습니다.

7. Docker Image Pull

- **설명:** 프로덕션 환경 또는 다른 서버에서 최신 Docker 이미지를 풀(pull)하여, 업데이트된 애플리케이션을 가져옵니다.

8. 컨테이너 교체

- **설명:** 최신 Docker 이미지가 풀된 후, 기존의 컨테이너를 교체하여 새로운 버전의 애플리케이션을 실행합니다. 일반적으로 이 작업은 롤링 업데이트 방식으로 진행되며, 서비스의 중단 없이 새 이미지를 배포합니다.

EC2 내부 설정

```
# HTTP 요청을 HTTPS로 리다이렉트
server {
    listen 80;
    server_name ddokddok.duckdns.org;
```

```

# 모든 HTTP 요청을 HTTPS로 리다이렉트
return 301 https://$host$request_uri;
}

# HTTPS 설정
server {
    listen 443 ssl;
    server_name ddokddok.duckdns.org;

    # SSL 인증서 경로 설정
    ssl_certificate /etc/letsencrypt/live/ddokddok.duckdns.org/fullchain.pem;
    ssl_certificate_key /etc/letsencrypt/live/ddokddok.duckdns.org/privkey.pem;

    # SSL 최적화 설정
    ssl_protocols TLSv1.2 TLSv1.3;
    ssl_prefer_server_ciphers on;
    ssl_session_timeout 10m;
    ssl_session_cache shared:SSL:10m;

    # 오류 페이지 설정
    error_page 414 /error_pages/uri_too_long.html;
    location = /error_pages/uri_too_long.html {
        root /usr/share/nginx/html;
        internal;
    }

    # 프론트엔드 요청을 처리하는 설정
    location / {
        proxy_pass http://frontend:80;
    }

    # 백엔드 API 요청을 처리하는 설정
    location /api/v1 {
        proxy_pass http://backend:8080;
    }

    # 알고리즘 추천 API 요청을 처리하는 설정
    location /algorithm/v1 {

```

```

    proxy_pass http://algorithm:8000;
    proxy_http_version 1.1;
    proxy_set_header Upgrade $http_upgrade;
    proxy_set_header Connection "upgrade";
    proxy_read_timeout 300s;
  }
}

```

서비스 설정 (Backend, Frontend, Algorithm, Redis, Nginx)

#/home/ubuntu/knockknock/docker-compose.yml

services:

backend:

container_name: knockknock-backend

image: imoong/knockknock-backend:latest


ports:

- "8080:8080"

env_file:

- backend/backend.env

volumes:

- backend-logs:/var/log/springboot #  로그 볼륨 추가

restart: always

networks:

- knockknock-network

- elk_elk #  ELK 네트워크 연결

depends_on:

- redis

frontend:


container_name: knockknock-frontend

image: imoong/knockknock-frontend:latest


env_file:

- frontend/frontend.env

environment:

- NODE_ENV=production #  프론트엔드 환경 설정

volumes:

- ./frontend/default.conf:/etc/nginx/conf.d/default.conf #  Nginx 설정

파일 마운트

```
restart: always
networks:
  - knockknock-network
```

```
algorithm:
  container_name: knockknock-algorithm
  image: imoong/knockknock-algorithm:latest
  ports:
    - "8000:8000" # 🚀 알고리즘 서비스 포트
  env_file:
    - algorithm/algorithm.env
  restart: always
  networks:
    - knockknock-network
  depends_on:
    - backend
```

```
nginx:
  container_name: nginx-proxy
  image: nginx:alpine
  ports:
    - "80:80"
    - "443:443" # 🔒 SSL 포트 설정
  volumes:
    - ./nginx/default.conf:/etc/nginx/conf.d/default.conf
    - ./nginx/html:/usr/share/nginx/html # 📁 HTML 파일
    - /etc/letsencrypt:/etc/letsencrypt:ro # 🔑 SSL 인증서
  depends_on:
    - frontend
    - backend
    - algorithm
  restart: always
  networks:
    - knockknock-network
```

```
redis:
  container_name: knockknock-redis
```

```
image: redis:alpine
volumes:
  - redis-data:/data # 🗄 Redis 데이터 저장
restart: always
networks:
  - knockknock-network
```

모니터링 서비스 (Prometheus, Node Exporter, cAdvisor, Grafana)

```
#!/home/ubuntu/monitoring/docker-compose.yml

services:
  prometheus:
    image: prom/prometheus:latest
    container_name: prometheus
    ports:
      - "9090:9090" # 📊 Prometheus UI 포트
    volumes:
      - ./prometheus/prometheus.yml:/etc/prometheus/prometheus.yml
      - prometheus_data:/prometheus # 📁 데이터 저장 볼륨
    command:
      - '--config.file=/etc/prometheus/prometheus.yml'
      - '--storage.tsdb.path=/prometheus'
    restart: unless-stopped
    networks:
      - monitoring

  node-exporter:
    image: prom/node-exporter:latest
    container_name: node-exporter
    ports:
      - "9100:9100" # 💻 시스템 메트릭 수집 포트
    volumes:
      - /proc:/host/proc:ro
      - /sys:/host/sys:ro
      - /:/rootfs:ro
```



```
command:
  - '--path.procfs=/host/proc'
  - '--path.rootfs=/rootfs'
restart: unless-stopped
networks:
  - monitoring

cadvisor:
  image: gcr.io/cadvisor/cadvisor:latest
  container_name: cadvisor
  ports:
    - "9280:8080" # 📊 리소스 모니터링 포트
  volumes:
    - /:/rootfs:ro
    - /var/run:/var/run:ro
    - /sys:/sys:ro
  restart: unless-stopped
  networks:
    - monitoring

grafana:
  image: grafana/grafana:latest
  container_name: grafana
  ports:
    - "3000:3000" # 📊 Grafana UI 포트
  volumes:
    - grafana_data:/var/lib/grafana # 📁 Grafana 데이터 저장
  environment:
    - GF_SECURITY_ADMIN_USER= # 👤 관리자 계정
    - GF_SECURITY_ADMIN_PASSWORD= # 🗝️ 관리자 비밀번호
  restart: unless-stopped
  networks:
    - monitoring
```

🔍 ELK 스택 서비스 (Elasticsearch, Logstash, Kibana, Filebeat)

```
#!/home/ubuntu/elk/docker-compose.yml
```

```
services:
```

```
  elasticsearch:
```

```
    image: docker.elastic.co/elasticsearch/elasticsearch:7.17.9
```

```
    container_name: elasticsearch
```

```
    command: >
```

```
      bash -c "elasticsearch-plugin install analysis-nori --batch || true  
      && /usr/local/bin/docker-entrypoint.sh"
```

```
    environment:
```

- node.name=elasticsearch
- cluster.name=elk-cluster
- discovery.type=single-node
- "ES_JAVA_OPTS=-Xms512m -Xmx512m"
- xpack.security.enabled=true
- ELASTIC_PASSWORD=fB0N28Nu75ySXcScWWzd

```
    volumes:
```

- esdata:/usr/share/elasticsearch/data # 📁 Elasticsearch 데이터 저장

```
    ports:
```

- "9200:9200"
- "9300:9300"

```
    networks:
```

- elk

```
    restart: unless-stopped
```

```
  logstash:
```

```
    image: docker.elastic.co/logstash/logstash:7.17.9
```

```
    container_name: logstash
```

```
    volumes:
```

- ./logstash/config/logstash.yml:/usr/share/logstash/config/logstash.yml

```
    |
```

- ./logstash/pipeline:/usr/share/logstash/pipeline

```
    ports:
```

- "5000:5000/tcp"
- "5000:5000/udp"
- "5044:5044"

```
    environment:
```

```
      LS_JAVA_OPTS:
```

```

ELASTICSEARCH_USERNAME:
ELASTICSEARCH_PASSWORD:
networks:
  - elk
depends_on:
  - elasticsearch
restart: unless-stopped

kibana:
  image: docker.elastic.co/kibana/kibana:7.17.9
  container_name: kibana
  environment:
    - ELASTICSEARCH_HOSTS=http://elasticsearch:9200
    - ELASTICSEARCH_USERNAME=elastic
    - ELASTICSEARCH_PASSWORD=fB0N28Nu75ySXcScWWzd
  volumes:
    - ./kibana/config/kibana.yml:/usr/share/kibana/config/kibana.yml
  ports:
    - "5601:5601" # 📊 Kibana UI 포트
  networks:
    - elk
  depends_on:
    - elasticsearch
  restart: unless-stopped

filebeat:
  image: docker.elastic.co/beats/filebeat:7.17.9
  container_name: filebeat
  volumes:
    - ./filebeat/filebeat.yml:/usr/share/filebeat/filebeat.yml:ro
    - /var/lib/docker/containers:/var/lib/docker/containers:ro
    - /var/log:/var/log:ro
    - knockknock_backend-logs:/var/log/springboot:ro
  user: root
  environment:
    - ELASTICSEARCH_USERNAME=
    - ELASTICSEARCH_PASSWORD=
  networks:

```

```
- elk
depends_on:
  - elasticsearch
  - logstash
restart: unless-stopped
```

각각의 `docker-compose.yml` 파일이 있는 디렉터리에서 아래 명령어를 실행하면 해당 서비스들을 백그라운드에서 실행할 수 있습니다:

```
docker compose up -d
```

이 명령어는 `docker-compose.yml` 파일을 참조하여 정의된 서비스들을 실행합니다. 각 파일을 처리하는 디렉터리에서 이 명령어를 실행하세요.