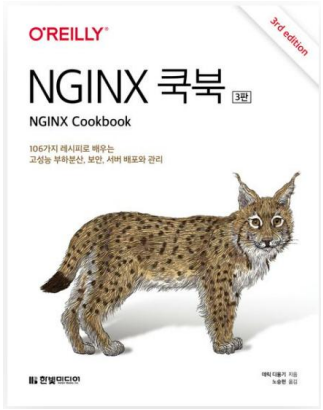
1. 참고 도서

참고 도서

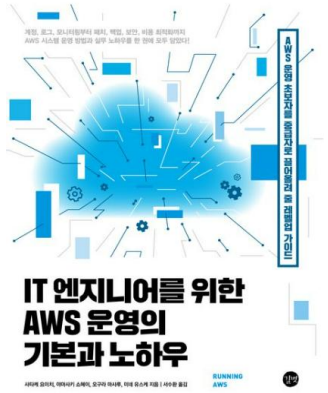
- 완독: 도커의 기본적인 전체 내용 학습.

< 그림으로 배우는 도커 >



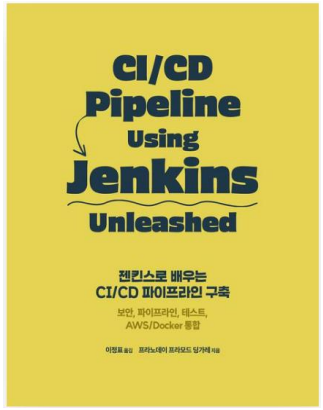
- 리버스 프록시 설정 참고.
- 부하분산 일부 학습.

< Nginx 쿡북 >



- EC2 서버 내용 학습.

< IT 엔지니어를 위한 AWS 운영의 기본과 노하우 >

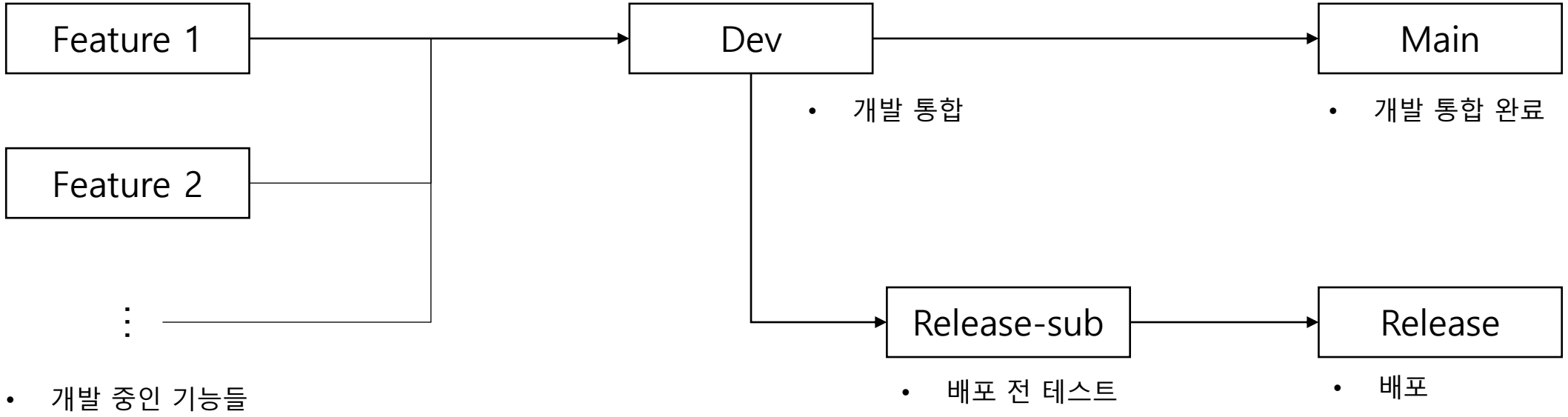


- Jenkins의 Docker 세팅.
- Maven 프로젝트를 통한 실습.

< 젠킨스로 배우는 CI/CD 파이프라인 구축 >

2. 개인 회고

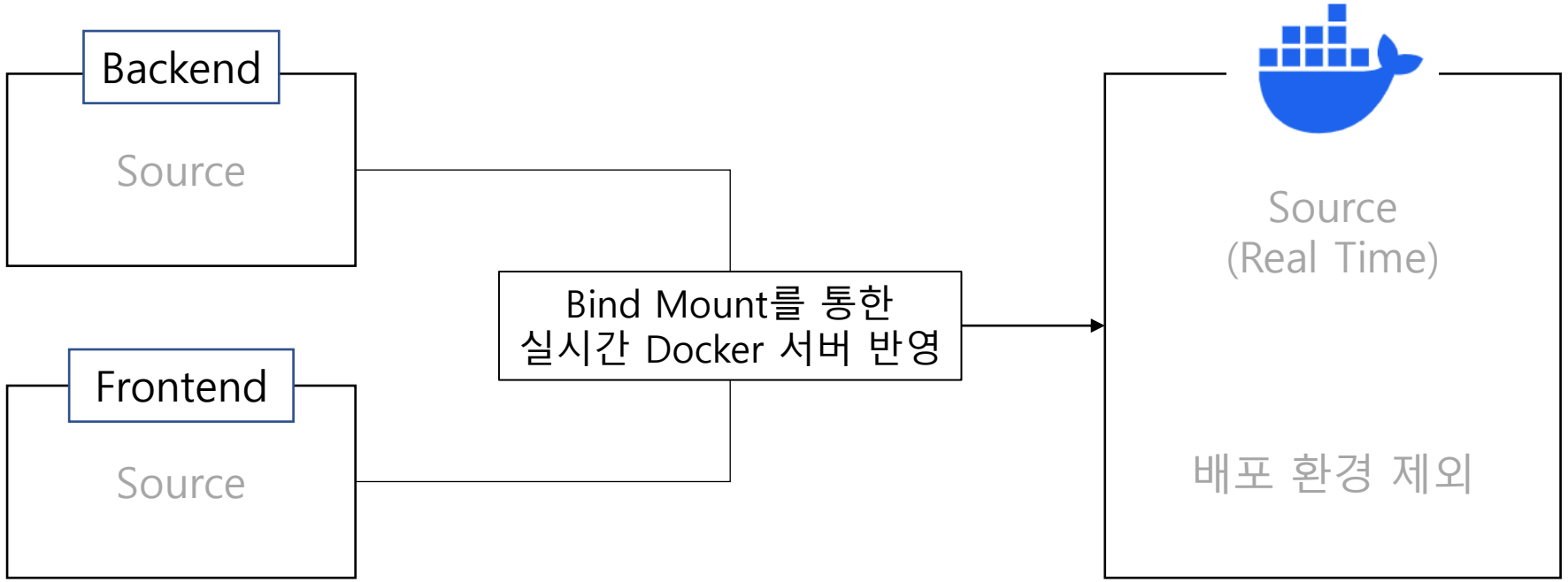
Branch 전략을 통한 개발 / 배포 환경 관리



회고

- 개발 환경과 배포 환경을 독립적으로 관리 → 배포 안정성 확보
- 개발은 Dev → Main으로, 배포는 Dev → Release-sub → Release로 병렬적으로 진행되도록 구현하여 불필요한 시간 단축.
- Jenkins의 필요성. 배포와 동시에 통합 테스트를 하기에 자동화 파이프를 통한 시간 확보의 필요성을 체감.

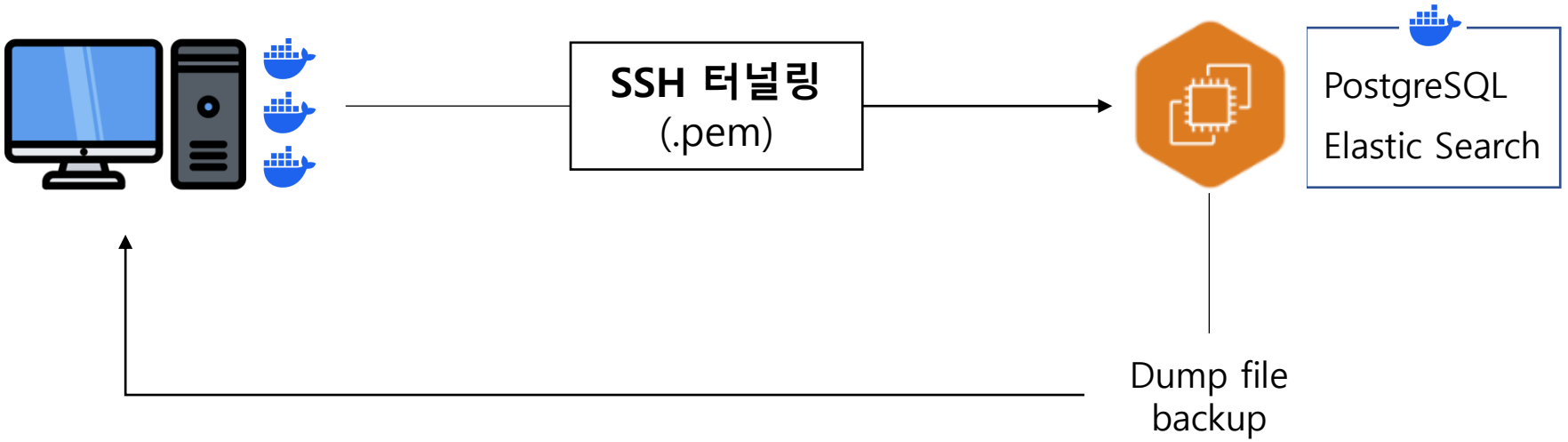
바인드 마운트를 통한 실시간 개발 환경



회고

- 개발과 동시에 결과를 도커 서버에서 실시간 확인.
- 간단한 디버깅에 편리.
- 배포 환경으로 수정할 것이 비교적 적음.
- 간헐적으로 바인드 마운트가 꼬여 docker compose build --no-cache가 필요.

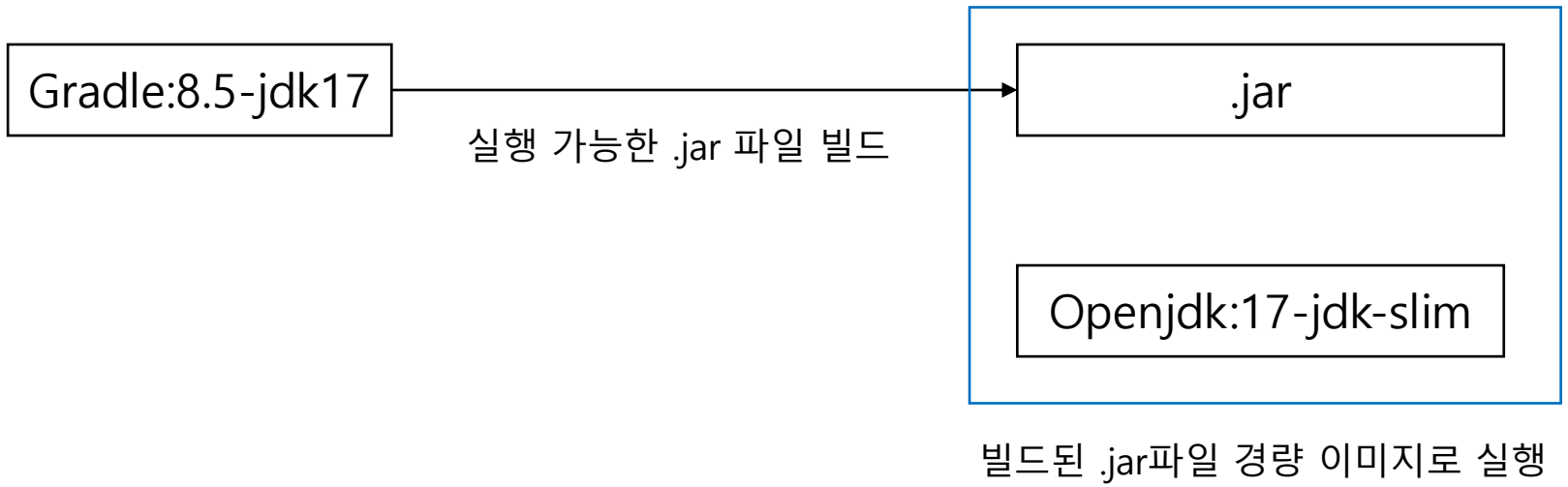
SSH 터널링을 통한 LIVE DB 활용



회고

- SSH 터널링을 통해 보안과 실시간성을 확보.
- EC2 내에 DB를 가동하여 실시간으로 데이터를 적재하며 개발을 진행.
- EC2 서버가 다운될 경우를 대비 dump 파일 → 로컬 저장 백업.
- 간헐적 터널링 실패.
  → Dockerfile 업데이트 및 바인드 마운트를 설정하여 application.properties를 동기화
  → 트러블슈팅 완료.

백엔드 이미지 경량화 (멀티스테이지)



회고

- Java를 사용하는 경우 .jar를 활용하여 경량화가 가능하다.
  → 이미지 축소, 보안 강화, 배포 효율 증가로 이점이 많은 방법
- .jar 빌드할 때 Dockerfile의 코드 순서 중요. (캐시나 COPY 등에서 문제 발생하는 경우가 있다).
- Flink와 같은 다른 java app에서도 확장하여 적용할 수 있겠다는 생각이 들었다.

3. 아쉬운 점 & 흥미를 가지게 된 부분

Jenkins

- 배포를 하면서 개발 환경의 테스트까지 진행하는데 시간이 많이 소모되었다.
- Jenkins를 사용하면 시간을 감축할 수 있다 기대되어 도입해보고 싶다.

Nginx

- 프로젝트에서는 단순 프록시만 진행하였는데 부하 분산과 본안 관련 환경 변수가 정말 많다는 걸 배웠다.
- 서버를 다수 활용할 수 있는 환경에서 부하분산과 보안설정을 구현해보는 것도 분산처리에 있어 도전적인 과제가 되겠다 생각한다.

Kubernetes

- 개별 컨테이너를 수동으로 관리하고, 물리적 자원을 통합 관리하지 않았더니 예상한 대로 EC2 서버 RAM의 여유가 적었다.
- 오케스트레이션을 통해 통합 환경을 최적화 해보고 싶다.

Prometheus & Grafana

- EC2의 가용 RAM이 부족하기에 모니터링 툴을 제외하였다.
- 처음부터 모니터링 툴을 도입하여 배포 테스트를 하면서 추적 관리하면 시스템 아키텍처를 관리하기 좋겠다는 생각이 들었다.