

# 우리팀의 CI / CD

7기 BE 칼리

# 나



암것도 모르는 사람

# 목차

- 1.CI / CD 란
- 2.우리 팀의 개발 및 운영 환경
3. 운영 환경 안정화를 위한 인프라 개선 사항

# 1. CI/CD 란?



# 1. CI/CD 란?

Google

CI / CD 란

×

전체

이미지

동영상


쇼핑

뉴스

짧은 동영상

웹

더보기

 Red Hat  
https://www.redhat.com › topics › devops › what-is-ci-cd

## CI/CD: 지속적 통합과 배포의 핵심 개념과 차이점 이해하기

2025. 2. 17. — CI/CD(지속적 통합/지속적 배포)는 애플리케이션 개발부터 배포까지 자동화된 프로세스로 효율성을 향상하는 방법입니다. CI/CD의 개념, 주요 툴, ...

요약

CI/CD가 중요한 이유


CI/CD에서 CD란?

CI/CD, DevOps, 플랫폼 엔지니...

 오픈소스컨설팅 테크블로그  
https://tech.osci.kr › Posts › Cloud


## 컨테이너 환경의 CI/CD 전략

2024. 2. 19. — 지속적 통합(Continuous Integration, CI): 자동화 프로세스를 통한 지속적인 통합을 의미합니다.  
· 지속적 제공(Continuous Delivery, CD): 변경 사항을 ...

 카카오클라우드  
https://blog.kakaocloud.com › ...

## <지식 사전> CI/CD란? DevOps의 핵심 - 지속적 통합/배포의 ...

2024. 11. 1. — **CI/CD**는 '지속적 통합(Continuous Integration)'과 '지속적 배포(Continuous Deployment)'를 아우르는 개발 방법론입니다. 이는 개발자가 코드를 작성하는 ...

 티스토리  
https://pm-developer-justdoit.tistory.com › ...

## [CICD] CICD의 개념 및 동작원리 2)

2023. 10. 28. — 1. CICD란 무엇일까? ... CICD는 Continuous Integration / Continuous Deployment 의 약자입니다. 한국어로 하면 지속적인 통합 과 지속적인 배포 입니다.

5

# 1. CI/CD 란?

Google

CI / CD 란

×

전체

이미지

동영상

쇼핑

뉴스

짧은 동영상

웹

더보기

 Red Hat

<https://www.redhat.com › topics › devops › what-is-ci-cd>

CI/CD: 지속적 통합과 배포의 핵심 개념과 차이점 이해하기


2025. 2. 17. — CI/CD(지속적 통합/지속적 배포)는 애플리케이션 개발부터 배포까지 자동화된 프로세스로 효율성을 향상하는 방법입니다. CI/CD의 개념, 주요 툴, ...

요약

CI/CD가 중요한 이유

CI/CD에서 CD란?

CI/CD, DevOps, 플랫폼 엔지니...

 오픈소스컨설팅 테크블로그

<https://tech.osci.kr › Posts › Cloud>

컨테이너 환경의 CI/CD 전략


2024. 2. 19. — 지속적 통합(Continuous Integration, CI): 자동화 프로세스를 통한 지속적인 통합을 의미합니다. · 지속적 제공(Continuous Delivery, CD): 변경 사항을 ...

 카카오클라우드

<https://blog.kakaocloud.com › ...>

<지식 사전> CI/CD란? DevOps의 핵심 - 지속적 통합/배포의 ...

2024. 11. 1. — CI/CD는 '지속적 통합(Continuous Integration)'과 '지속적 배포(Continuous Deployment)'를 아우르는 개발 방법론입니다. 이는 개발자가 코드를 작성하는 ...

 티스토리

<https://pm-developer-justdoit.tistory.com › ...>

[CICD] CICD의 개념 및 동작원리 2)

2023. 10. 28. — 1. CICD란 무엇일까? ... CICD는 Continuous Integration / Continuous Deployment 의 약자입니다. 한국어로 하면 지속적인 통합 과 지속적인 배포 입니다.

# 1. CI/CD 란?

Google CI / CD 란

전체 이미지 동영상 쇼핑 뉴스 짧은 동영상 웹 더보기

**Red Hat**  
https://www.redhat.com › topics › devops › what-is-ci-cd  
**CI/CD: 지속적 통합과 배포의 핵심 개념과 차이점 이해하기**  
2025. 2. 17. — CI/CD(지속적 통합/지속적 배포)는 애플리케이션 개발부터 배포까지 자동화된 프로세스로 효율성을 향상하는 방법입니다. CI/CD의 개념, 주요 툴, ...  
요약 CI/CD가 중요한 이유 CI/CD에서 CD란? CI/CD, DevOps, 플랫폼 엔지니...

**오픈소스컨설팅 테크블로그**  
https://tech.osci.kr › Posts › Cloud  
**컨테이너 환경의 CI/CD 전략**  
2024. 2. 19. — 지속적 통합(Continuous Integration, CI): 자동화 프로세스를 통한 지속적인 통합을 의미합니다.  
· 지속적 제공(Continuous Delivery, CD): 변경 사항을 ...

**카카오클라우드**  
https://blog.kakaocloud.com › ...  
**<지식 사전> CI/CD란? DevOps의 핵심 - 지속적 통합/배포의 ...**  
2024. 11. 1. — CI/CD는 '지속적 통합(Continuous Integration)'과 '지속적 배포(Continuous Deployment)'를 아우르는 개발 방법론입니다. 이는 개발자가 코드를 작성하는 ...

**티스토리**  
https://pm-developer-justdoit.tistory.com › ...  
**[CICD] CICD의 개념 및 동작원리 2)**  
2023. 10. 28. — 1. CICD란 무엇일까? ... CICD는 Continuous Integration / Continuous Deployment 의 약자입니다. 한국어로 하면 지속적인 통합 과 지속적인 배포 입니다.

**CI** (Continuous Integration)  
- 지속적인 통합

**CD** (Continuous Deployment)  
- 지속적인 배포

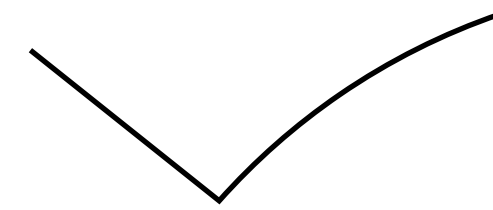
# 1.1. C란

지속적인 통합



# 1.1. C란

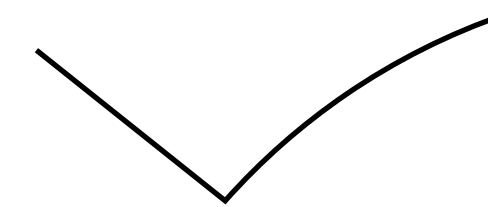
코드의



지속적인 통합

# 1.1. CI란

코드의



(여러 개발자들이 함께 개발하는 과정 속에서) **지속적인 통합**

# 1.1. CI란

코드의  
↓  
(여러 개발자들이 함께 개발하는 과정 속에서) **지속적인 통합**

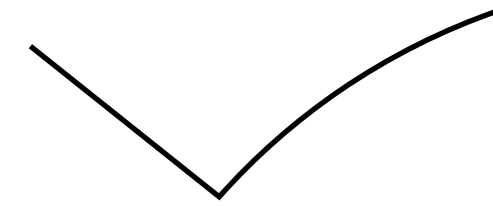
dev



# 1.1. CI란

(여러 개발자들이 함께 개발하는 과정 속에서) **지속적인 통합**

코드의



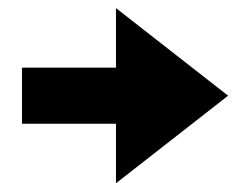
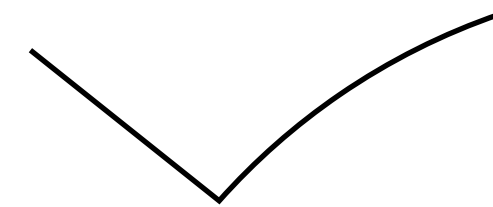
dev



# 1.1. CI란

(여러 개발자들이 함께 개발하는 과정 속에서) **지속적인 통합**

코드의



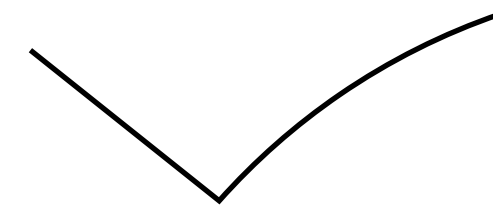
dev



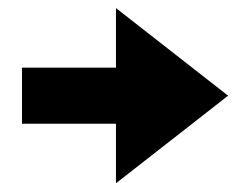
# 1.1. CI란

(여러 개발자들이 함께 개발하는 과정 속에서) **지속적인 통합**

코드의



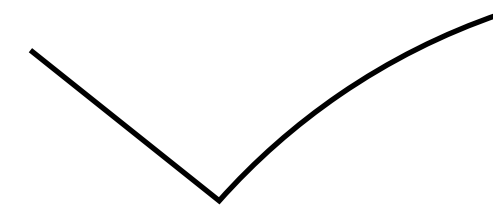
dev



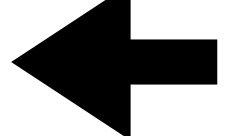
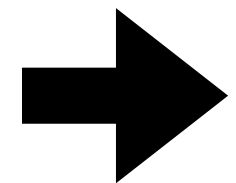
# 1.1. CI란

(여러 개발자들이 함께 개발하는 과정 속에서) **지속적인 통합**

코드의



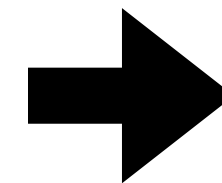
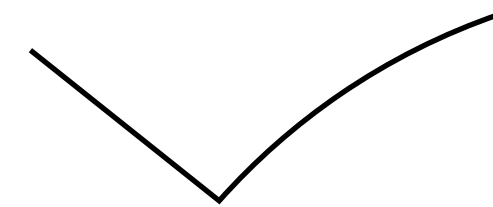
dev



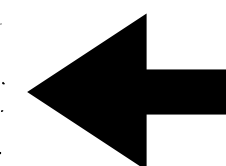
# 1.1. CI란

(여러 개발자들이 함께 개발하는 과정 속에서) **지속적인 통합**

코드의



dev



하는 일: 정상 빌드 + 정상 테스트



# 1.1. CI란

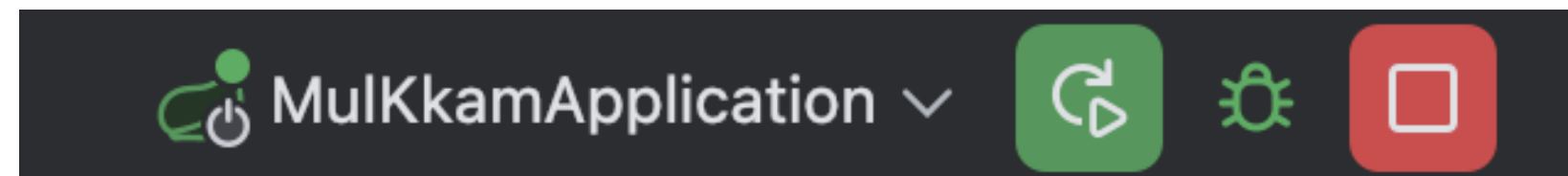
해야 하는 일

- 정상 빌드
- 정상 테스트

# 1.1. CI란

해야 하는 일

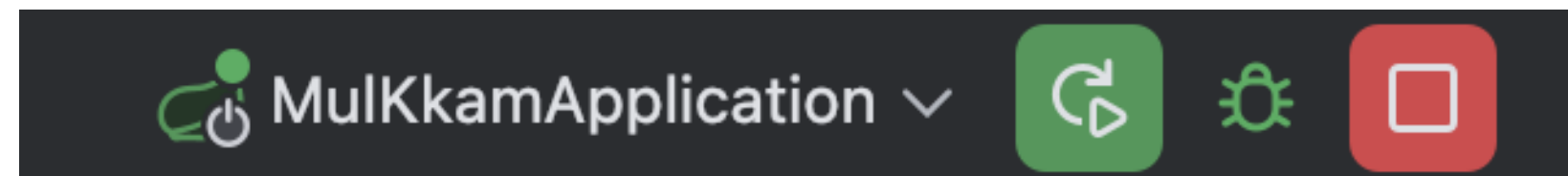
- 정상 빌드
- 정상 테스트



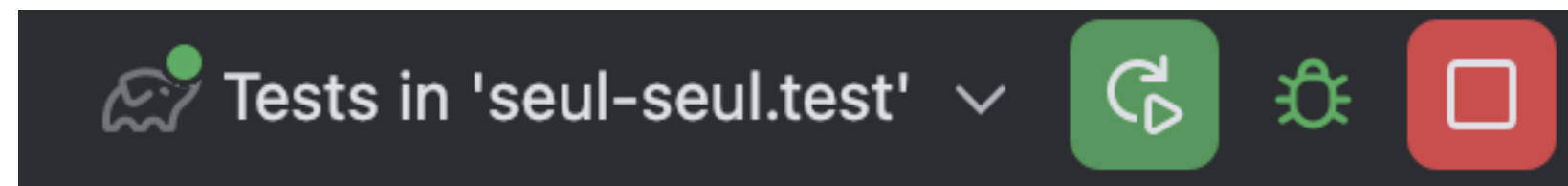
# 1.1. CI란

해야 하는 일

- 정상 빌드



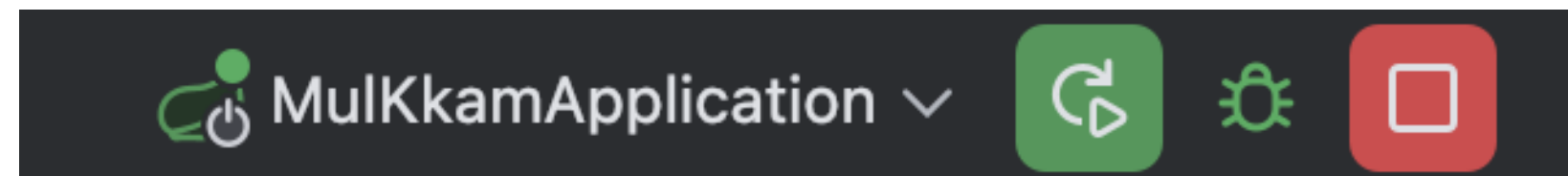
- 정상 테스트



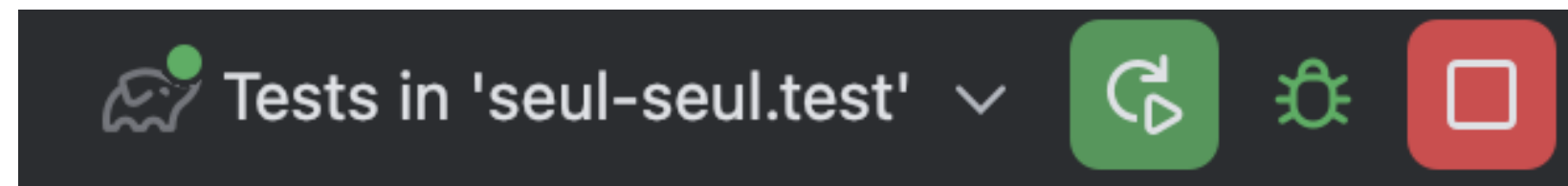
# 1.1. C란


## 해야 하는 일

- 정상 빌드




- 정상 테스트








2Jin1031 commented on Sep 21, 2024 · edited


Collaborator

☒  테스트는 잘 통과했나요?

☒  빌드는 성공했나요?

☒  불필요한 코드는 제거했나요?

☒  이슈는 등록했나요?

☒  라벨은 등록했나요?

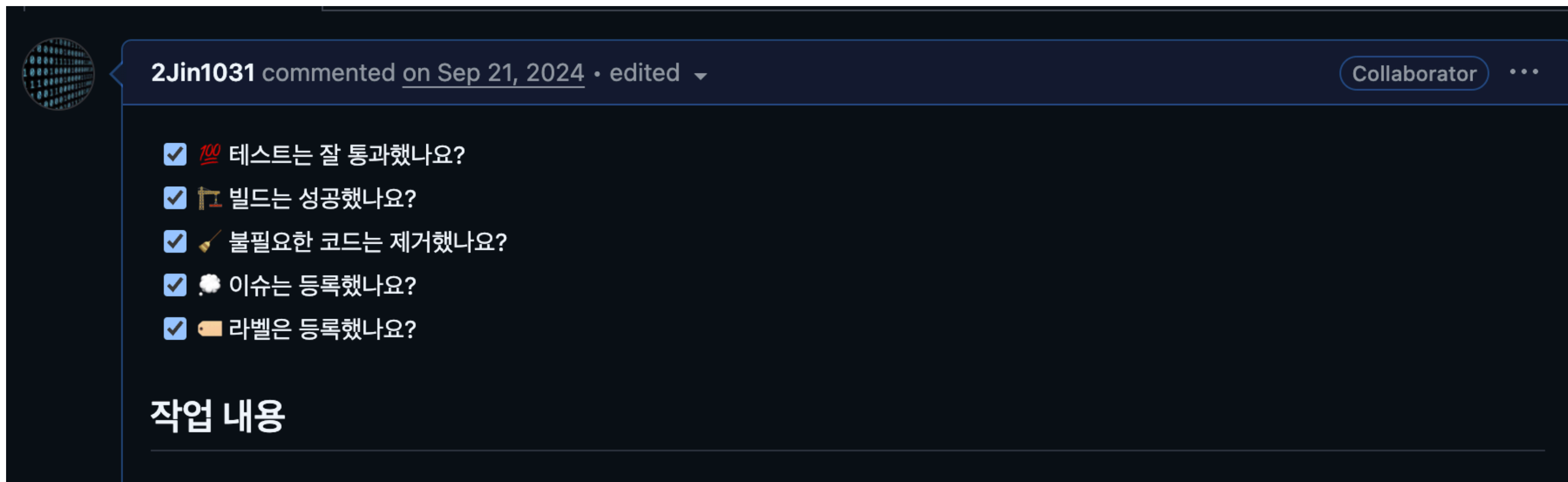
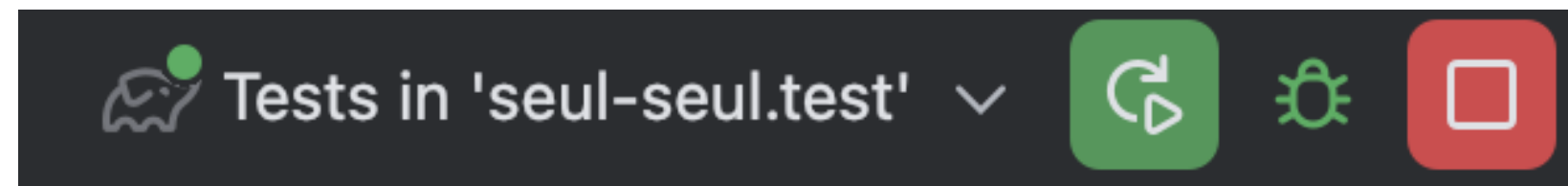
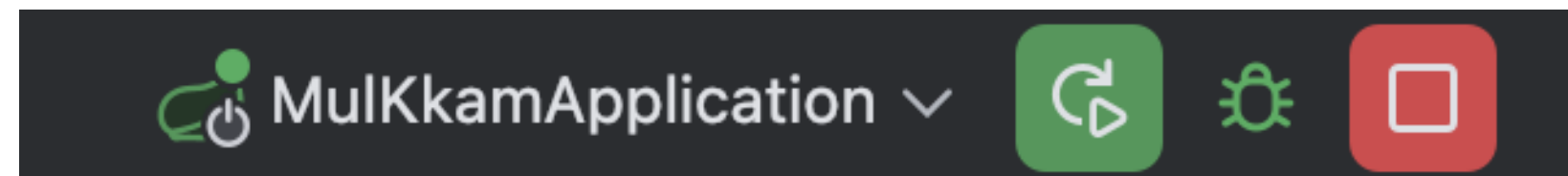
작업 내용

# 1.1. C란

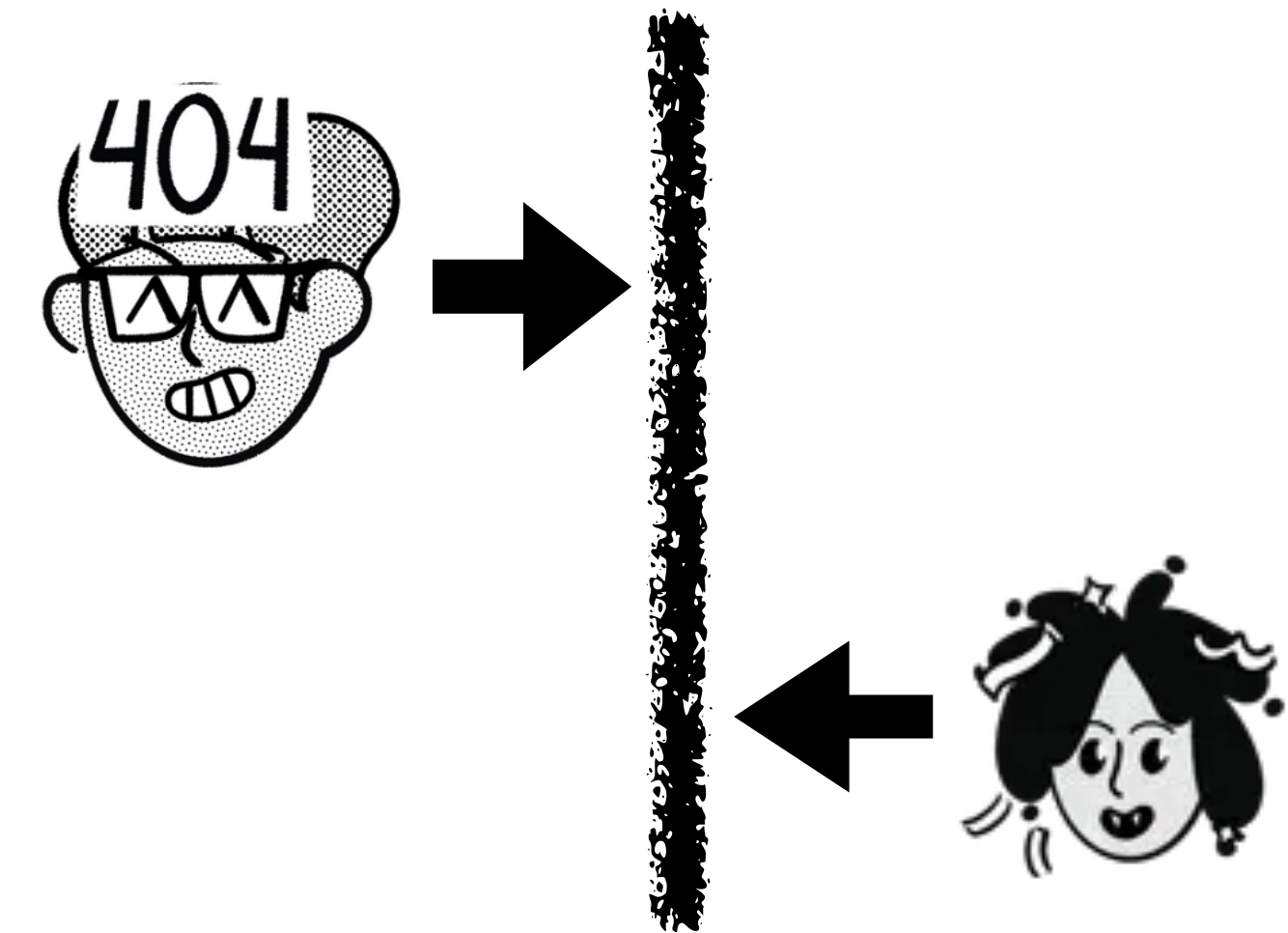
해야 하는 일

- 정상 빌드

- 정상 테스트



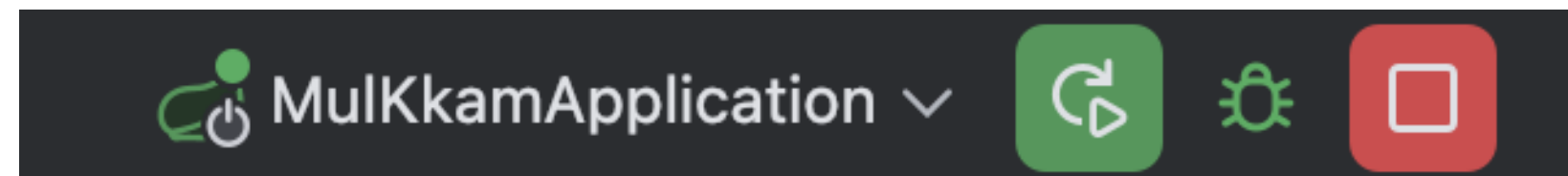
dev



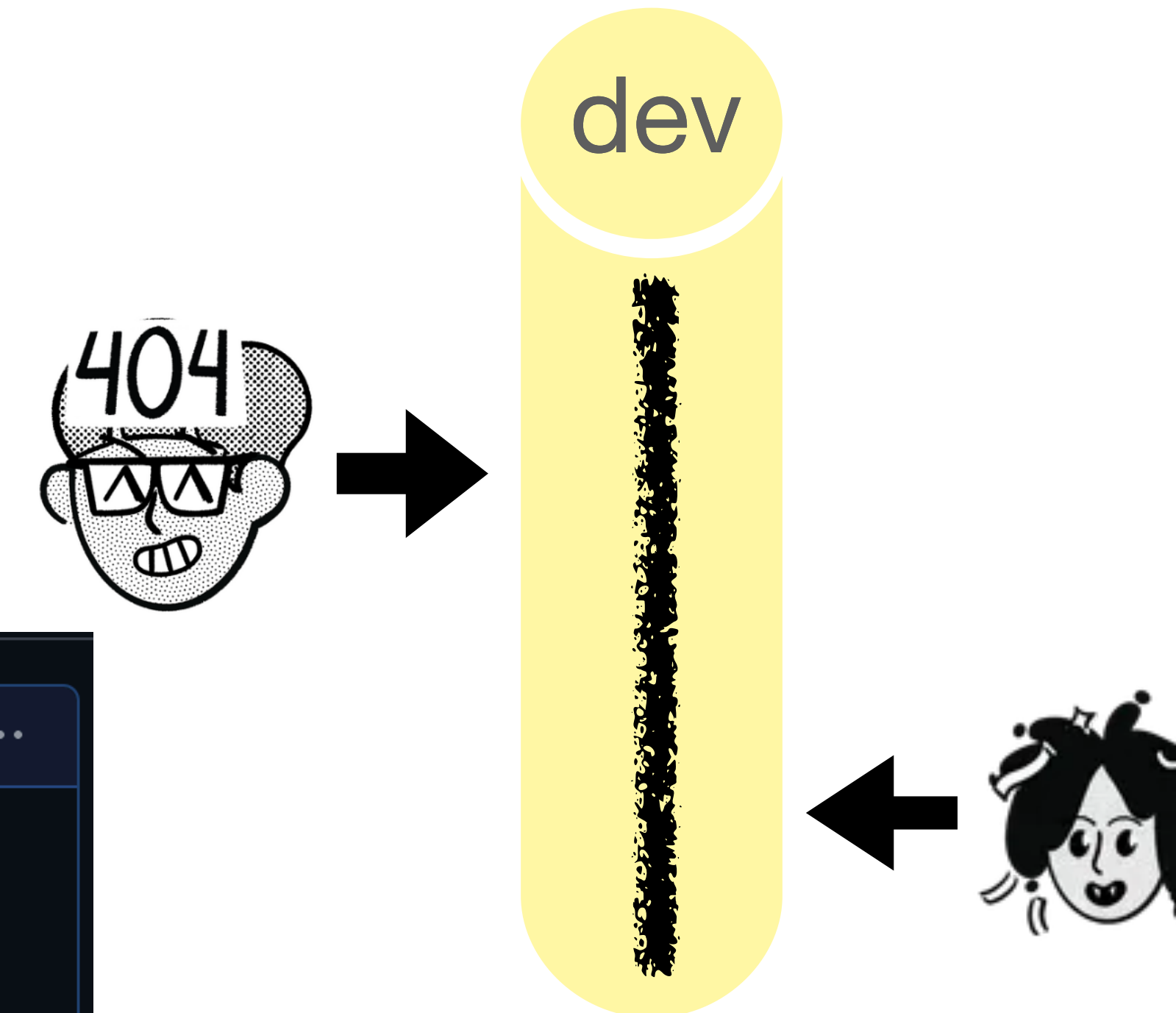
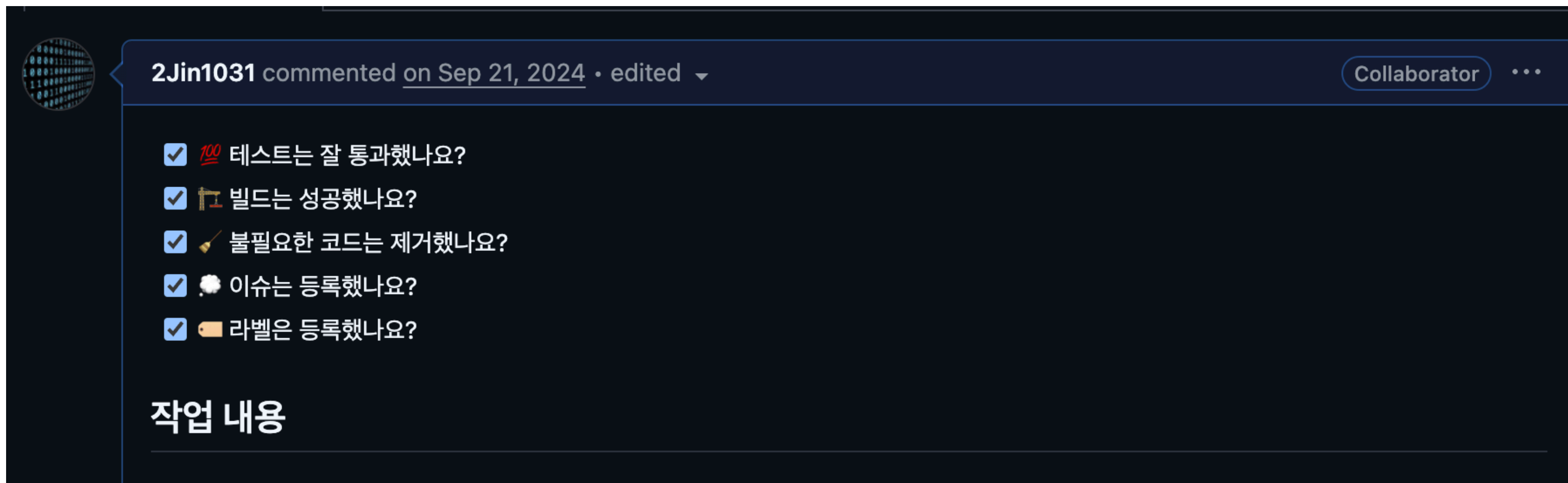
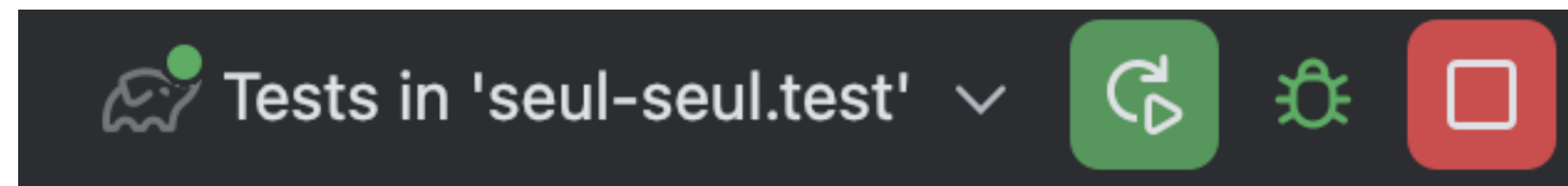
# 1.1. C란

## 해야 하는 일

- 정상 빌드

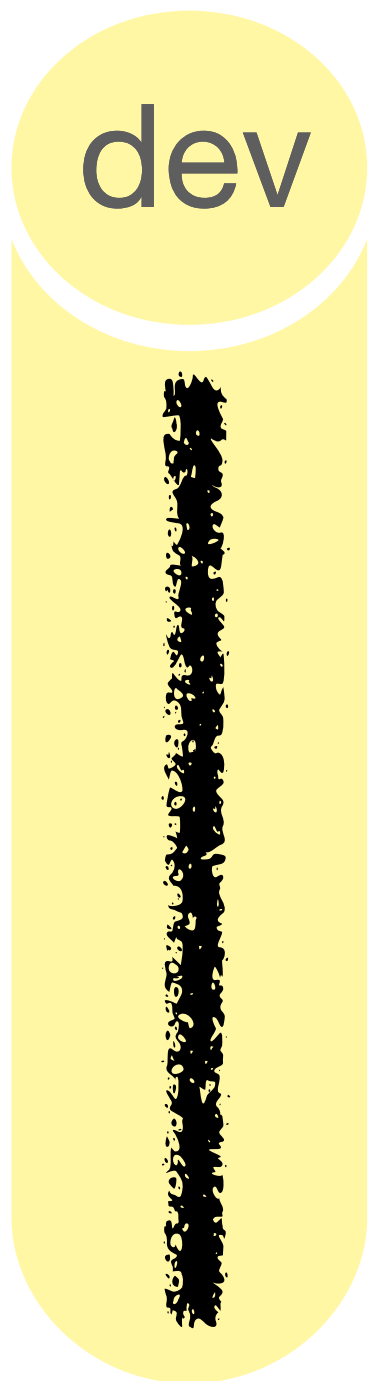


- 정상 테스트



# 1.1. CI란

자동화된 빌드와 테스트 수행 -> 신뢰성과 안정성에 대한 확신



# 1.1. CI란

자동화된 빌드와 테스트 수행 -> 신뢰성과 안정성에 대한 확신

<CI 스크립트>

dev

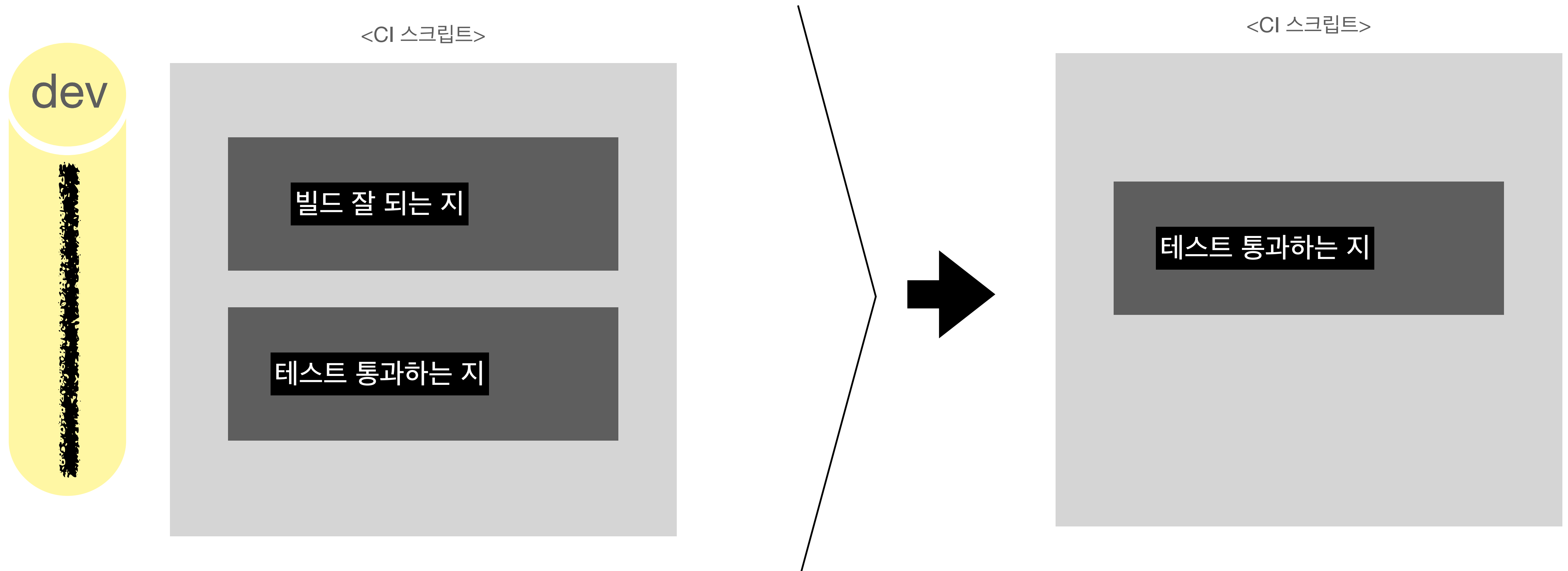
빌드 잘 되는 지

테스트 통과하는 지



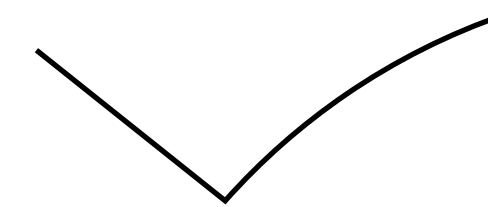
# 1.1. CI란

자동화된 빌드와 테스트 수행 -> 신뢰성과 안정성에 대한 확신



# 1.1. CI란

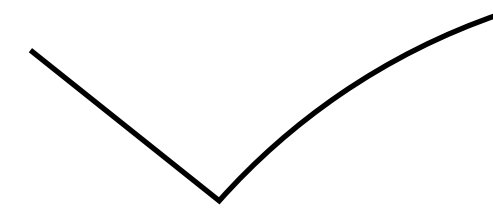
코드의



(여러 개발자들이 함께 개발하는 과정 속에서) **지속적인 통합**

# 1.1. C란

코드의



지속적인 통합

# 1.1. C란

지속적인 통합

## 1.2. CD란

지속적인 배포

## 1.2. CD란

지속적인 배포

(운영 환경에 코드 업데이트)

## 1.2. CD란

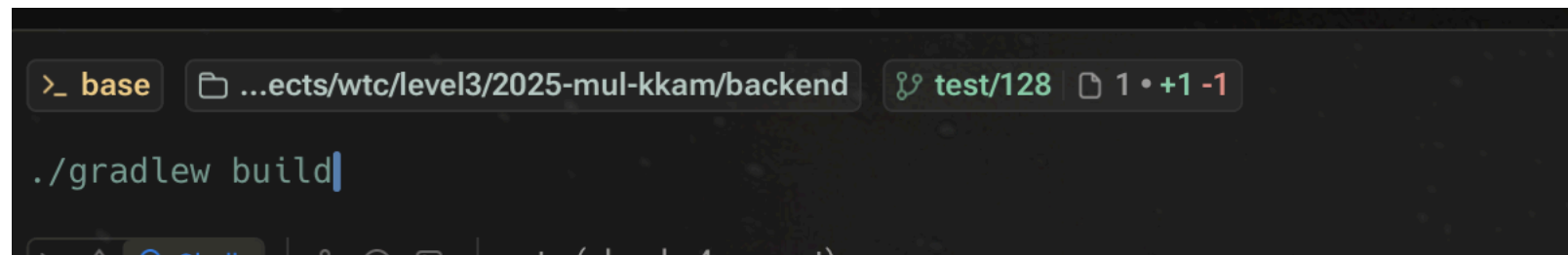
```
ssh -i "key.pem" ubuntu@EC2_IP
  cd backend
  ./gradlew build
  cd build/libs/
  PID=$(lsof -t -i:8080)
  if [ -n "$PID" ]; then
    kill -9 $PID
  fi
  java -jar mulkkam-0.0.1-SNAPSHOT.jar
```

<deploy.sh>



# 1.2. CD란

```
ssh -i "key.pem" ubuntu@EC2_IP
cd backend
./gradlew build
cd build/libs/
PID=$(lsof -t -i:8080)
if [ -n "$PID" ]; then
kill -9 $PID
fi
java -jar mulkkam-0.0.1-SNAPSHOT.jar
```



<deploy.sh>

```
#!/bin/bash
set -e

# 1. 이전 빌드 백업
backup_previous_build() {
    if [ -f "$PROJECT_PATH/build/libs/$JAR_NAME" ]; then
        log_info "이전 빌드를 백업합니다..."
        TIMESTAMP=$(date +%Y%m%d_%H%M%S)
        cd "$PROJECT_PATH/build/libs/$JAR_NAME"
        log_info "백업 경로: $JAR_NAME_$TIMESTAMP"
    fi
}

# 2. 소스코드 최신화
build_application() {
    log_info "소스코드를 업데이트합니다..."

    cd $PROJECT_PATH

    # 기존 빌드 파일 삭제
    log_info "기존 빌드 파일을 정리합니다..."
    ./gradlew clean

    # 새로운 빌드 실행
    log_info "새로운 빌드를 시작합니다..."
    ./gradlew build && exit $? || { log_error "빌드 실패"; exit 1; }

    # 3. 빌드 결과 확인
    JAR_PATH="build/libs/$JAR_NAME"
    if [ ! -f "$JAR_PATH" ]; then
        log_error "JAR 파일을 찾을 수 없습니다: $JAR_PATH"
        exit 1
    fi

    # 4. 프로세스 ID 생성
    echo $! > "$JAR_PATH/$APP_NAME.pid"
    log_info "애플리케이션이 시작되었습니다. PID: $!"
}

# 5. 애플리케이션 상태 확인
check_application_status() {
    log_info "애플리케이션 상태를 확인합니다..."

    local max_attempts=10
    local attempt=1
    local wait_time=10

    # 애플리케이션 시작 대기
    log_info "애플리케이션 시작을 위해 $wait_time초 대기합니다..."
    sleep $wait_time

    # 포트가 열릴 때까지 대기 (최대 30초)
    log_info "포트 $PORT가 열릴 때까지 대기합니다..."
    for i in $(seq 1 30); do
        if lsof -ti:$PORT >/dev/null 2>&1; then
            log_info "🟢 포트 $PORT가 열렸습니다."
            break
        fi
        log_warn "🟡 포트 $PORT가 아직 열리지 않았습니다. 재시도 중 ($i/30)"
        sleep 3
    done

    while [ $attempt -le $max_attempts ]; do
        log_info "Health Check $attempt/$max_attempts"

        # HTTP Health Check
        HTTP_STATUS=$(curl -s -o /dev/null -w "%{http_code}" \
            "http://localhost:$PORT/actuator/health")

        log_info "HTTP 상태 코드: $HTTP_STATUS"

        if [ "$HTTP_STATUS" = "200" ]; then
            # 6. 로그 파일 확인
            HEALTH_RESPONSE=$(curl -s "http://localhost:$PORT/actuator/health")
            --connect-timeout 5 --max-time 10 2>/dev/null || echo ""

            if echo "$HEALTH_RESPONSE" | grep -q '"status":"UP"'; then
                log_info "🟢 Health Check 성공"
                log_info "응답: $HEALTH_RESPONSE"
                return 0
            else
                log_warn "응답에 UP 상태가 없습니다: $HEALTH_RESPONSE"
            fi
        elif [ "$HTTP_STATUS" = "502" ]; then
            log_warn "연결 실패 - 애플리케이션이 아직 시작 중일 수 있습니다."
        else
            log_warn "예상치 못한 HTTP 상태: $HTTP_STATUS"
        fi

        sleep 3
        ((attempt++))
    done

    # 실패 시 디버그 정보 출력
    log_error "Health Check 최종 실패"
    log_error "최종 응답: $HEALTH_RESPONSE"

    # 포트 상태 확인
    if lsof -ti:$PORT >/dev/null 2>&1; then
        log_error "포트 $PORT 사용 중이지만..."
    else
        log_error "포트 $PORT에 실행 중인 프로세스가 없습니다."
    fi

    return 1
}

# 7. 로그 관리
manage_logs() {
    log_info "오래된 로그 파일을 정리합니다..."

    # 1. 가장 최근 10 로그 파일 삭제
    find $JAR_PATH -name "*.log" -mtime +7 -delete

    # 2. $BACKUP_DIR -name "*.jar" -mtime +30 -delete
    log_info "로그 관리 완료"
}

# 메인 실행 부분
# =====
main() {
    log_info "===== 시작 ====="
    log_info "Spring Boot 애플리케이션 배포를 시작합니다"
    log_info "===== 종료 ====="

    # 1. 디렉토리 최신화
    update_git_branch

    # 2. 환경변수
    # load_env_file
    # validate_env

    # 3. 디렉토리 생성
    create_directories

    # 4. 기존 애플리케이션 종료
    stop_application

    # 5. 이전 빌드 백업
    backup_previous_build

    # 6. 소스코드 최신화
    build_application

    # 7. 애플리케이션 시작
    start_application

    # 8. 상태 확인
    check_application_status

    # 9. 로그 관리
}
```



# 1.2. CD란

자동화된 배포 수행 → 일관된 배포와 운영 환경 반영에 대한 확신

<CD 스크립트>

빌드

운영서버에서 pull 받아 띄우기

## 2. 우리 팀의 개발 및 운영 환경

- 도커의 활용 필요성을 크게 체감하지 못함
- 도메인이 작음 -> 일정의 여유 & 이슈 대응에 비교적 용이

=> 보수적으로 개발 방향을 잡아나갔음

## 2. 우리 팀의 개발 및 운영 환경

- 도커의 활용 필요성을 크게 체감하지 못함
- 도메인이 작음 -> 일정의 여유 & 이슈 대응에 비교적 용이

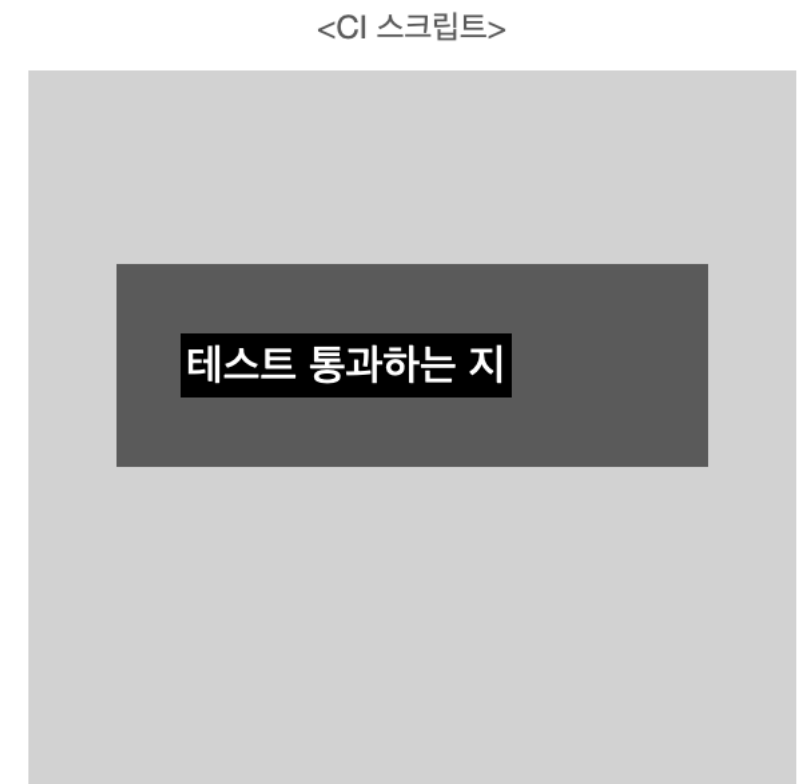
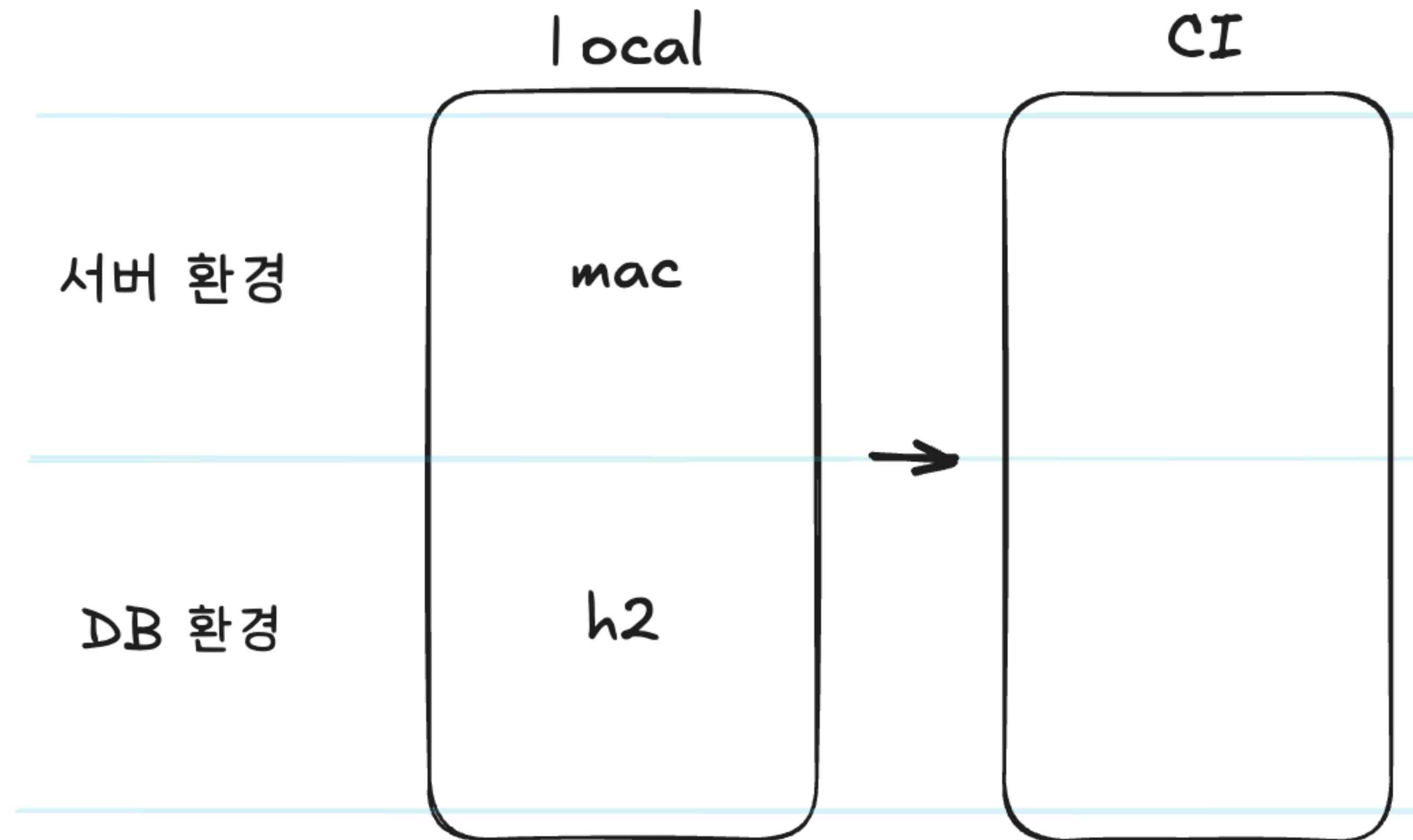
=> 보수적으로 개발 방향을 잡아나갔음

	local	운영 환경
운영체제	mac	ubuntu 24.04
아키텍처	arm64	arm64
DB	h2	mySQL

## 2.1. CI / CD 환경

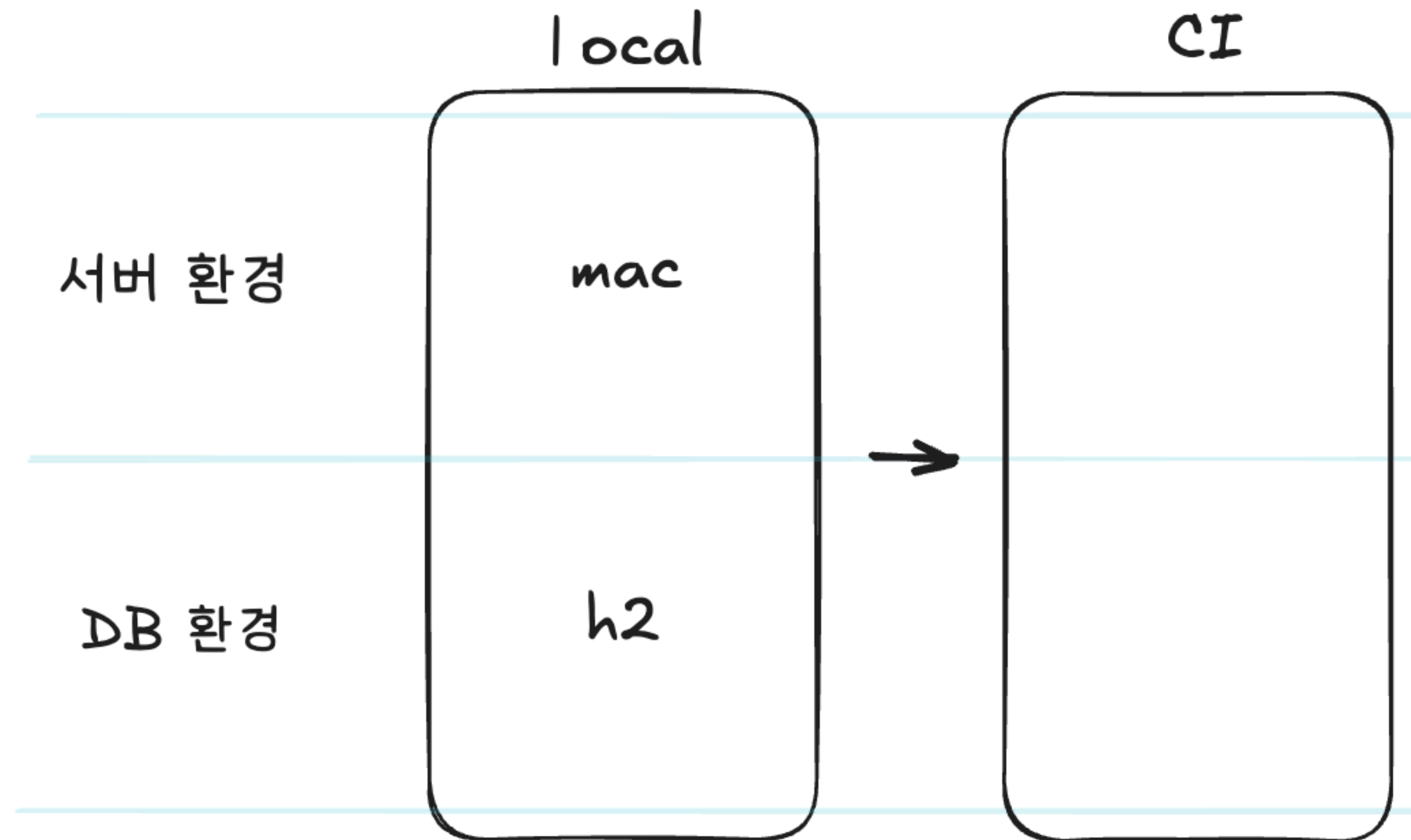


## 2.1. CI / CD 환경



(행위)  
테스트 통과

## 2.1. CI / CD 환경



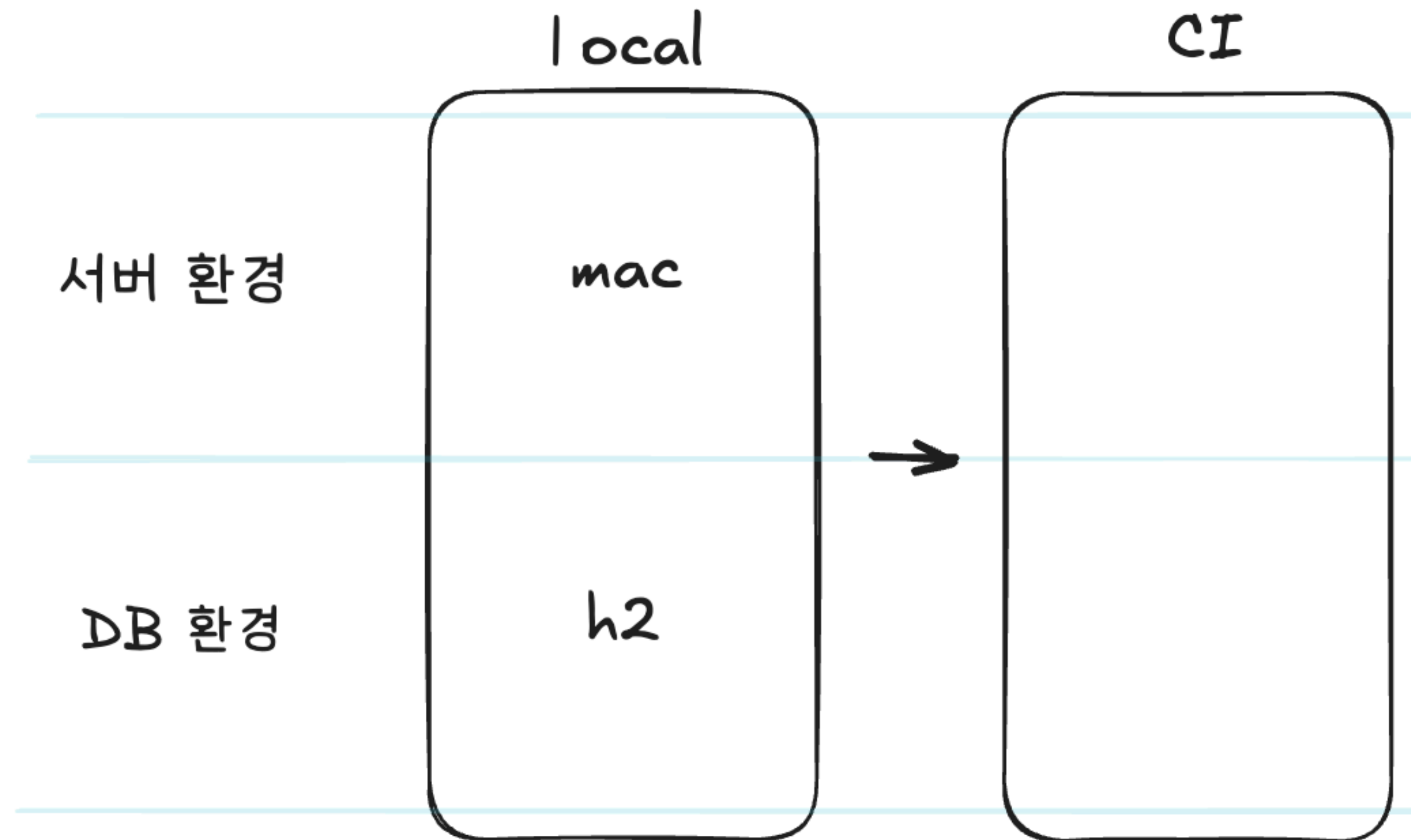
(목적) (행위)  
운영 서버에서의 안전성 테스트 통과



## 2.1. CI / CD 환경

<CI 스크립트>

테스트 통과하는 지



(목적)

운영 서버에서의 안전성

(행위)

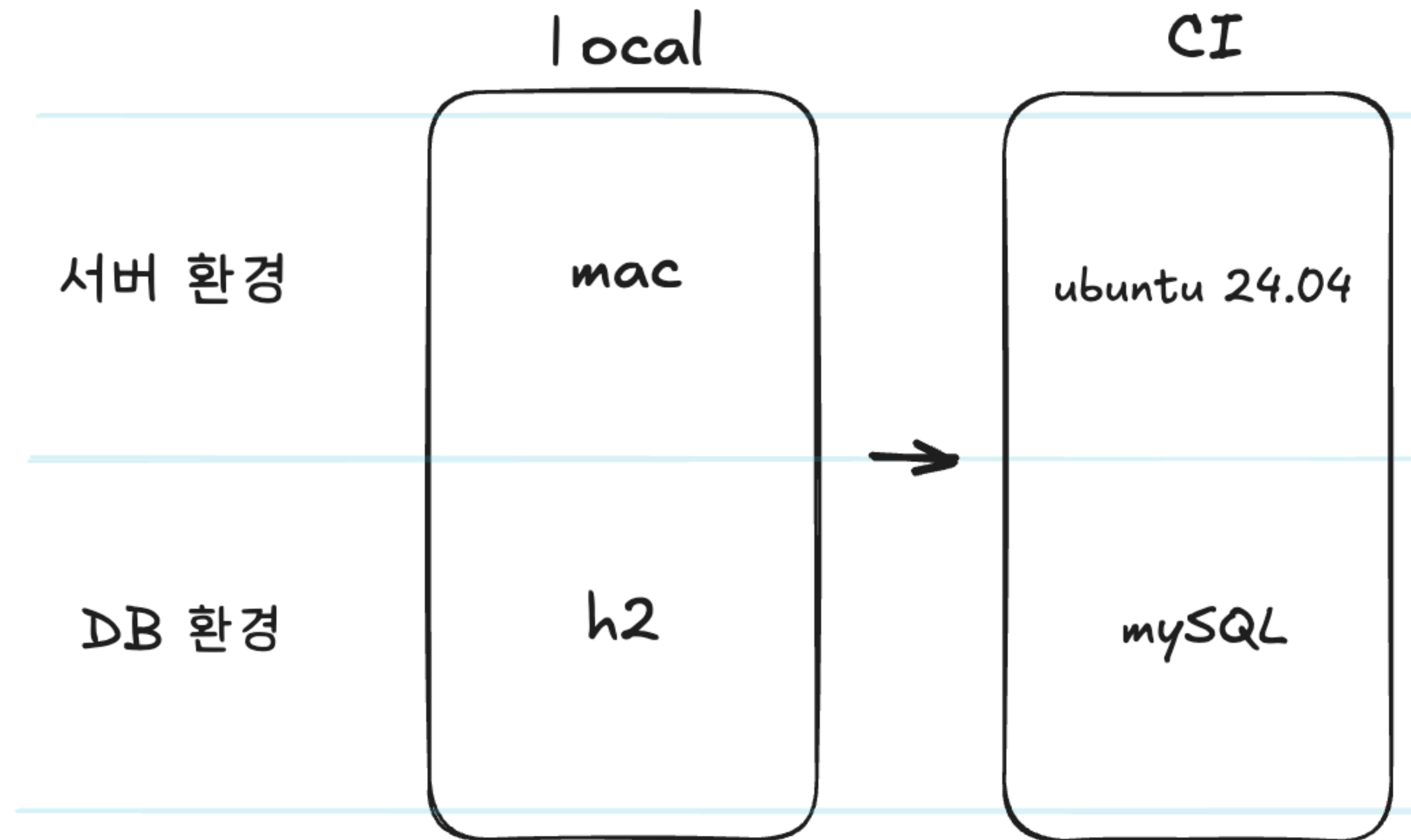
테스트 통과

=> 운영 서버와 동일한 환경으로 테스트

## 2.1. CI / CD 환경

<CI 스크립트>

테스트 통과하는 지

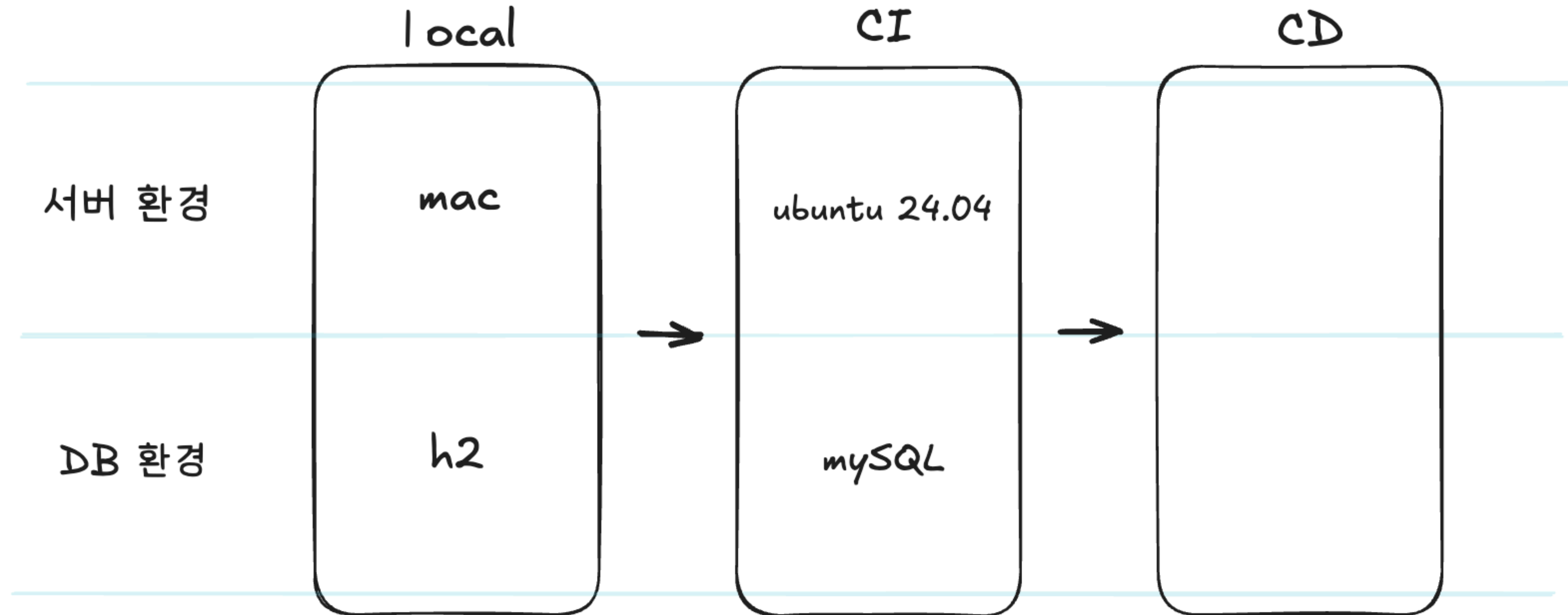


(목적) 운영 서버에서의 안전성  
(행위) 테스트 통과

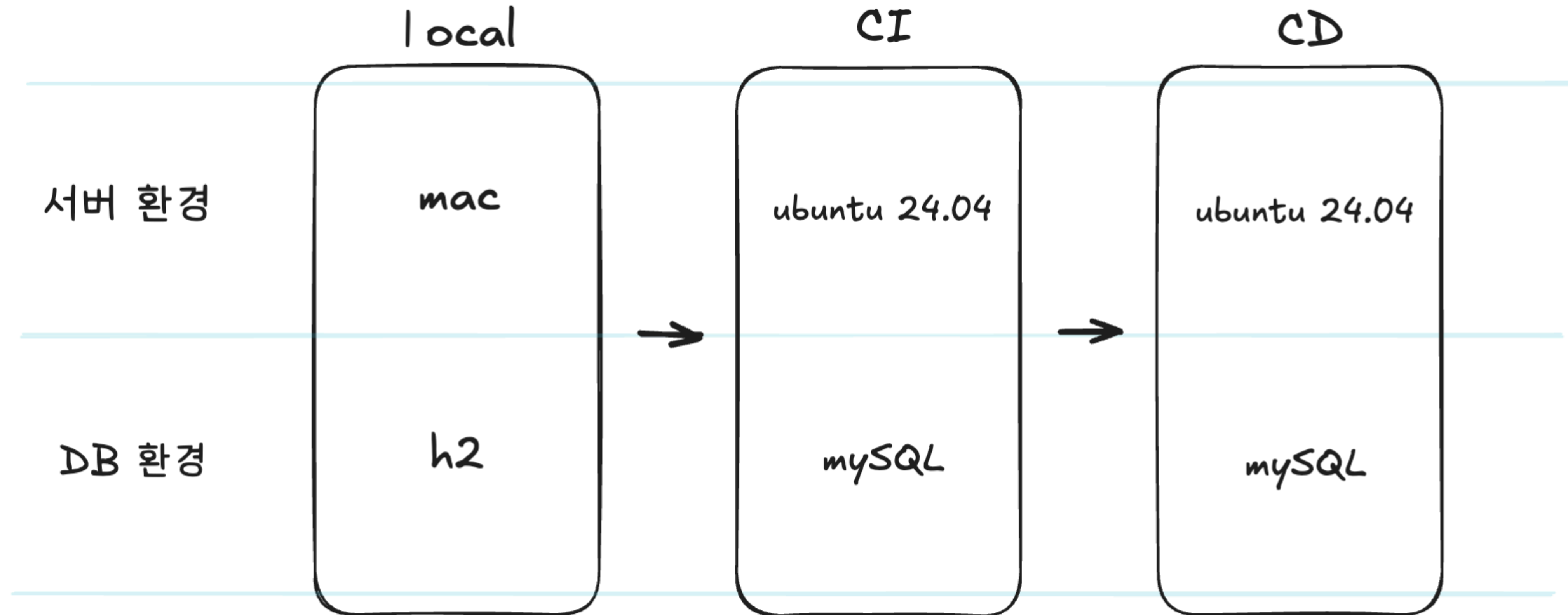
=> 운영 서버와 동일한 환경으로 테스트



## 2.1. CI / CD 환경



## 2.1. CI / CD 환경



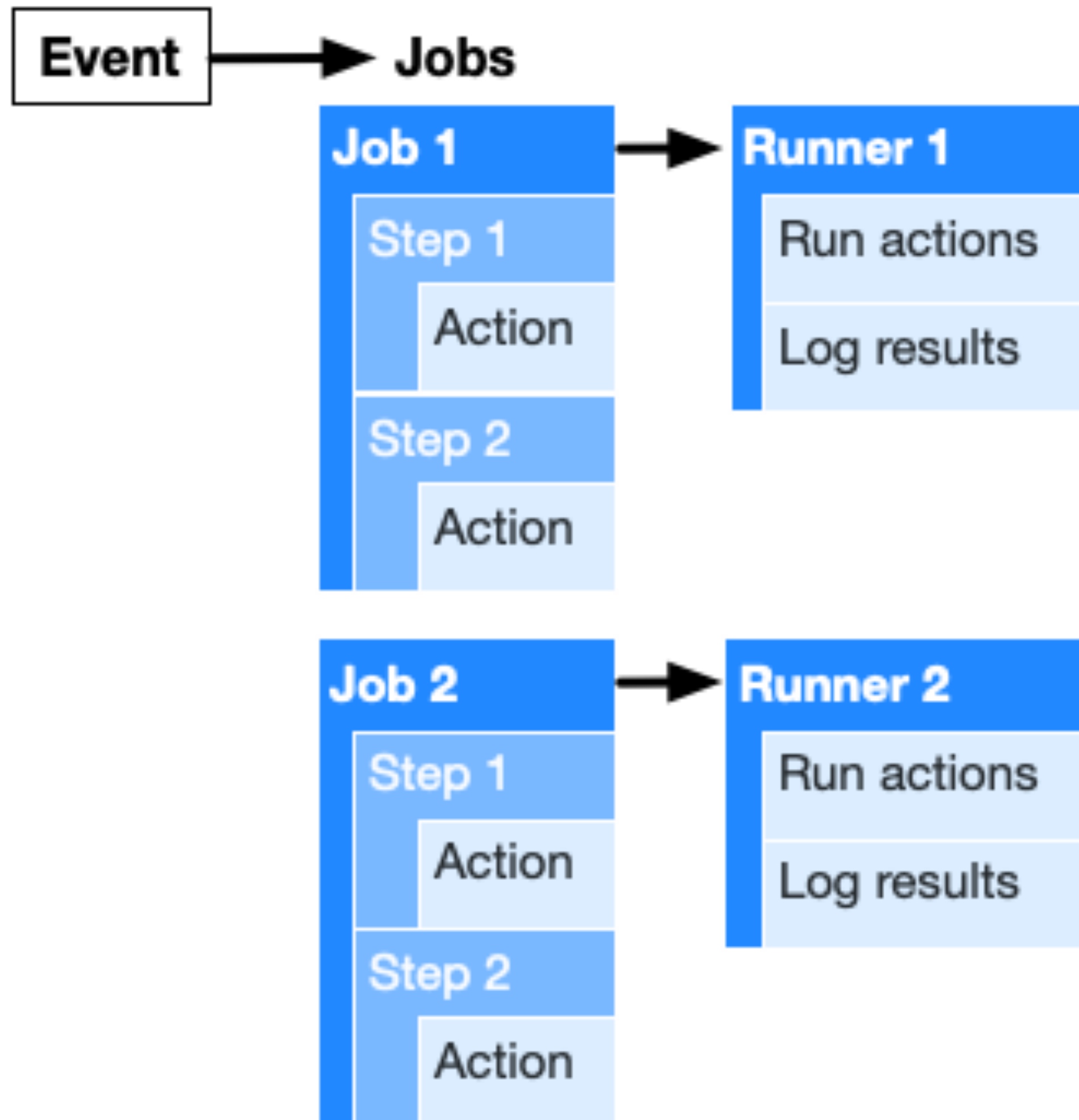
## 2.2. Github Actions란?

## 2.2. Github Actions란?

특정한 이벤트가 발생했을 때 내가 원하는 일을 자동으로 수행할 수 있도록 만들어주는 툴

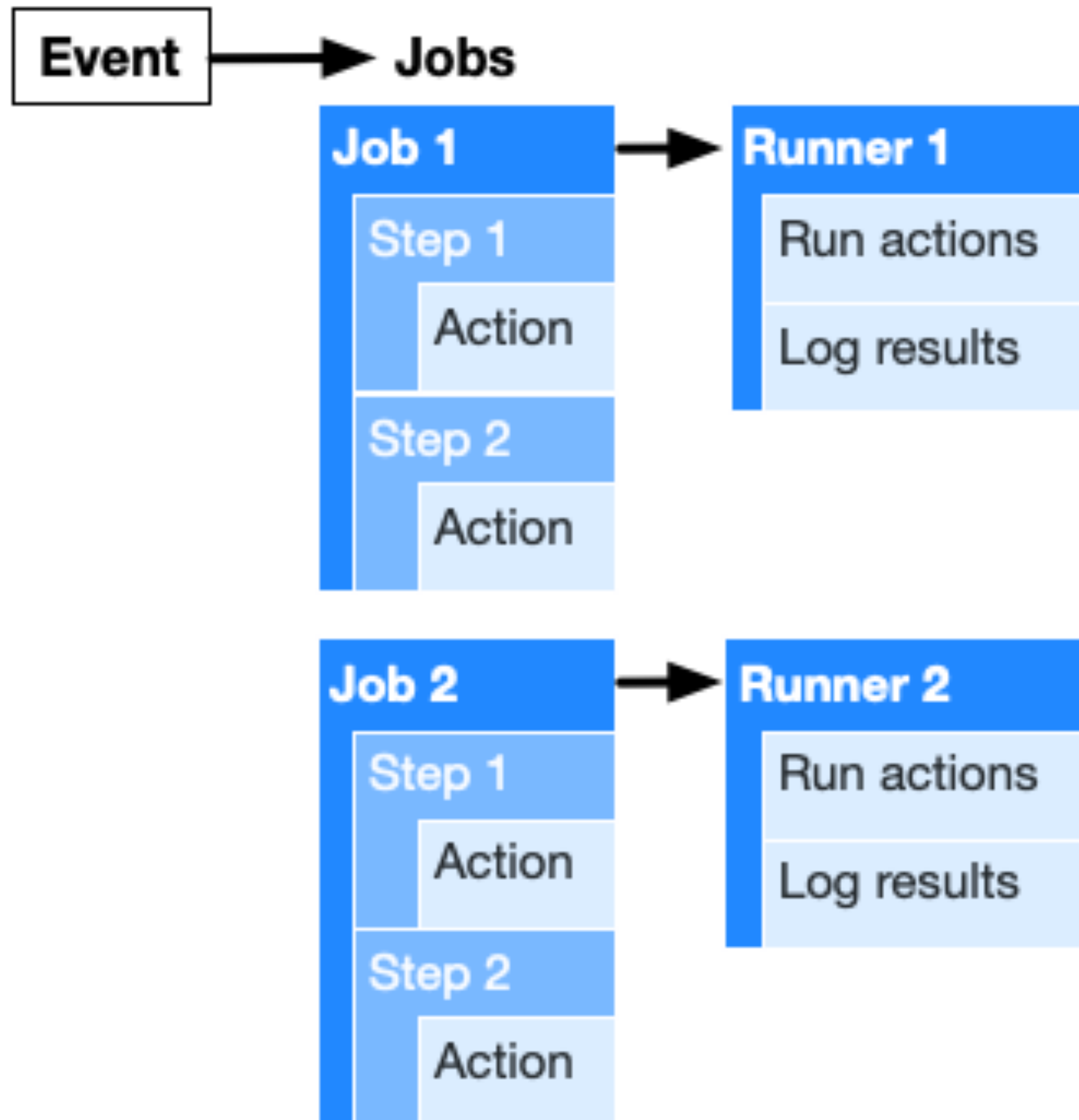
## 2.2. Github Actions란?

특정한 이벤트가 발생했을 때 내가 원하는 일을 자동으로 수행할 수 있도록 만들어주는 툴



## 2.2. Github Actions란?

특정한 이벤트가 발생했을 때 내가 원하는 일을 자동으로 수행할 수 있도록 만들어주는 툴



용어

1. event: 어떤 일이 발생할 때
2. Job: step의 그룹화
3. step: 스크립트 or action
4. actions : 다양한 명령들
5. runner : job을 실행

## 2.3. 운영 환경에 따른 Github Actions



## 2.3. 운영 환경에 따른 Github Actions



언제?



## 2.3. 운영 환경에 따른 Github Actions

CI

언제? pr을 올릴 때마다

ubuntu 24.04

mysql

## 2.3. 운영 환경에 따른 Github Actions

CI

언제? pr을 올릴 때마다 -> 이벤트

ubuntu 24.04

mysql

## 2.3. 운영 환경에 따른 Github Actions

CI

ubuntu 24.04

mySQL

언제? pr을 올릴 때마다 -> 이벤트

어떻게?

1. runner 내부에 mysql 깔아서 테스트 하기
2. Docker 컨테이너로 MySQL 띄우기

## 2.3. 운영 환경에 따른 Github Actions

CI

ubuntu 24.04

mysql

언제? pr을 올릴 때마다 -> 이벤트

어떻게?

1. runner 내부에 mysql 깔아서 테스트 하기
2. Docker 컨테이너로 MySQL 띄우기

## 2.3. 운영 환경에 따른 Github Actions

CI



언제? pr을 올릴 때마다 -> 이벤트

어떻게? Docker 컨테이너로 MySQL 띄우기

## 2.3. 운영 환경에 따른 Github Actions

CI

ubuntu 24.04

mysql

언제? pr을 올릴 때마다 -> 이벤트

어떻게? Docker 컨테이너로 MySQL 띄우기

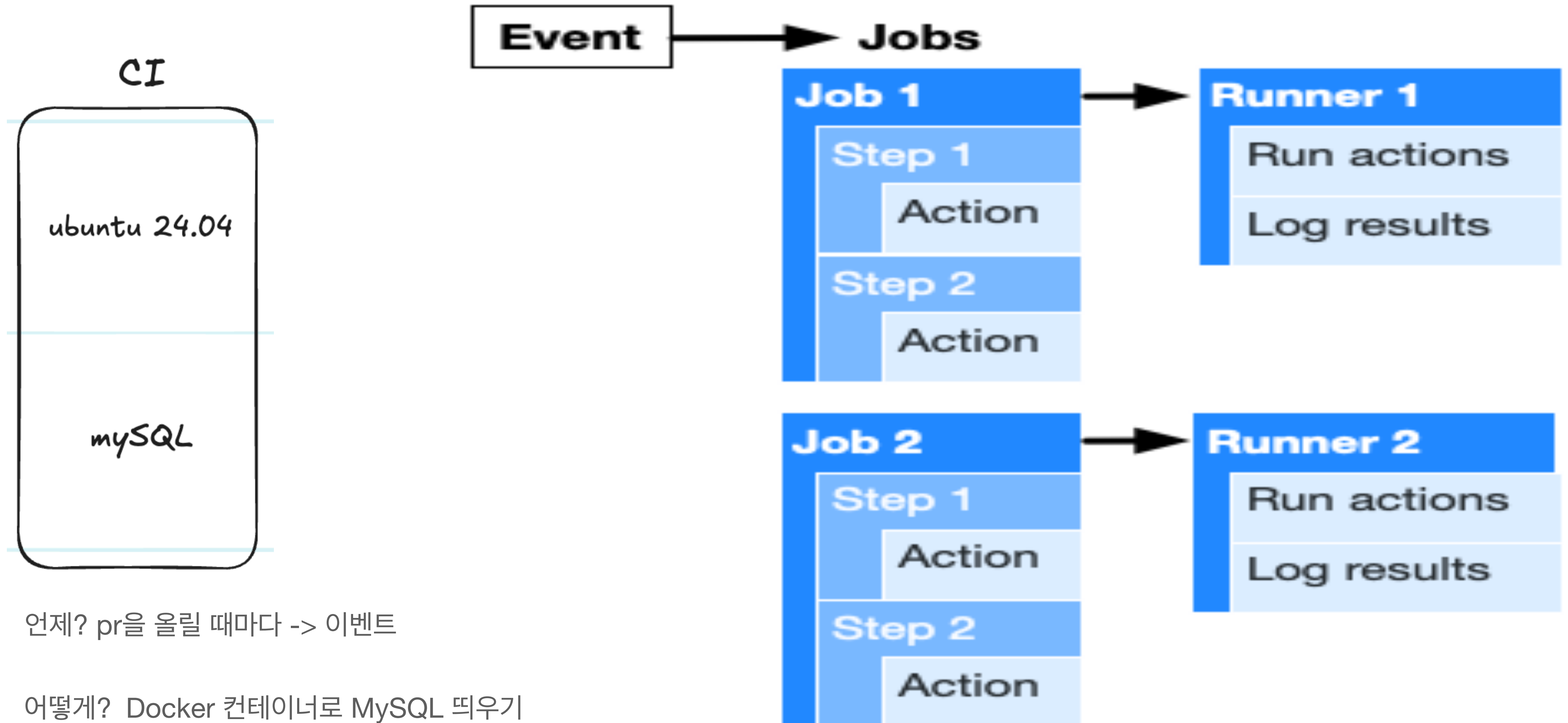
## 2.3. 운영 환경에 따른 Github Actions



언제? pr을 올릴 때마다 -> 이벤트

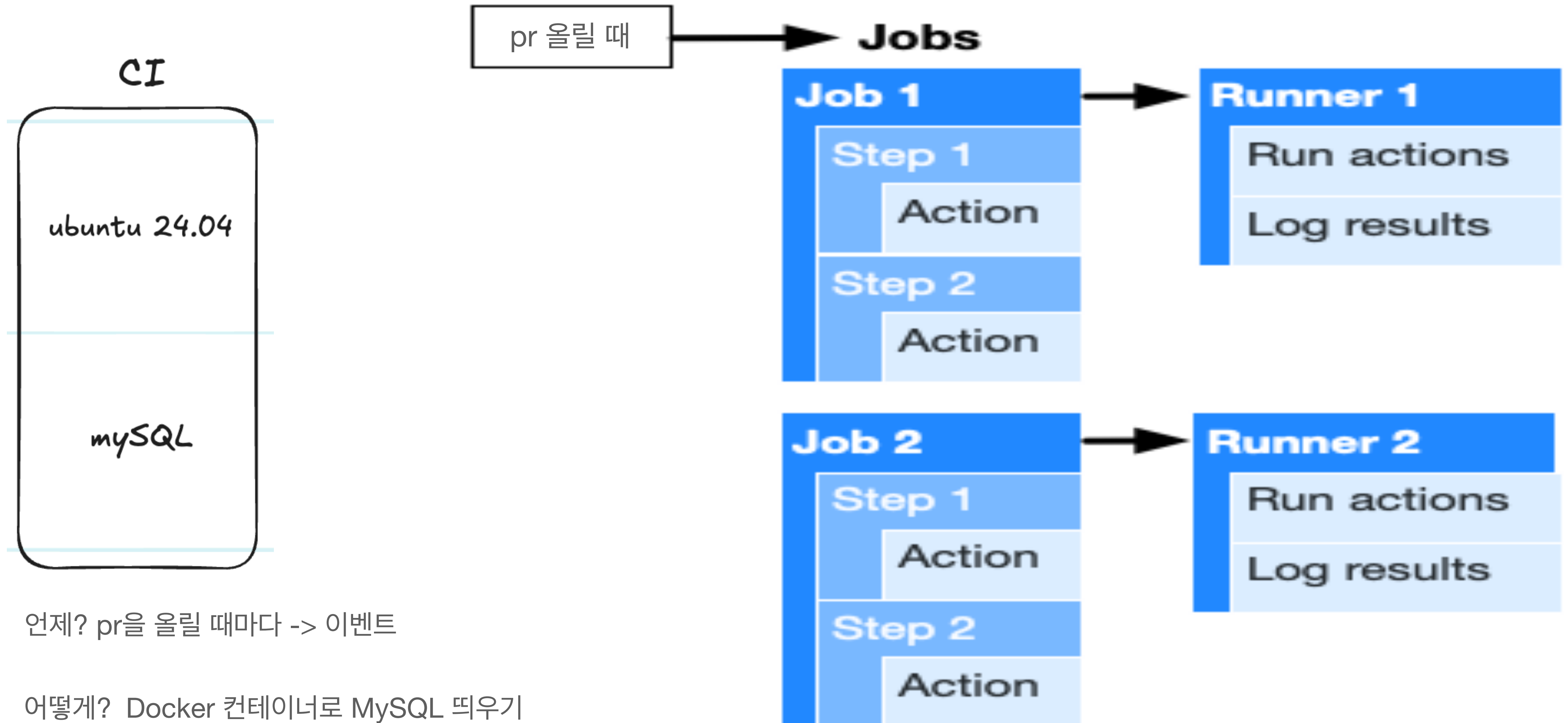
어떻게? Docker 컨테이너로 MySQL 띄우기

## 2.3. 운영 환경에 따른 Github Actions

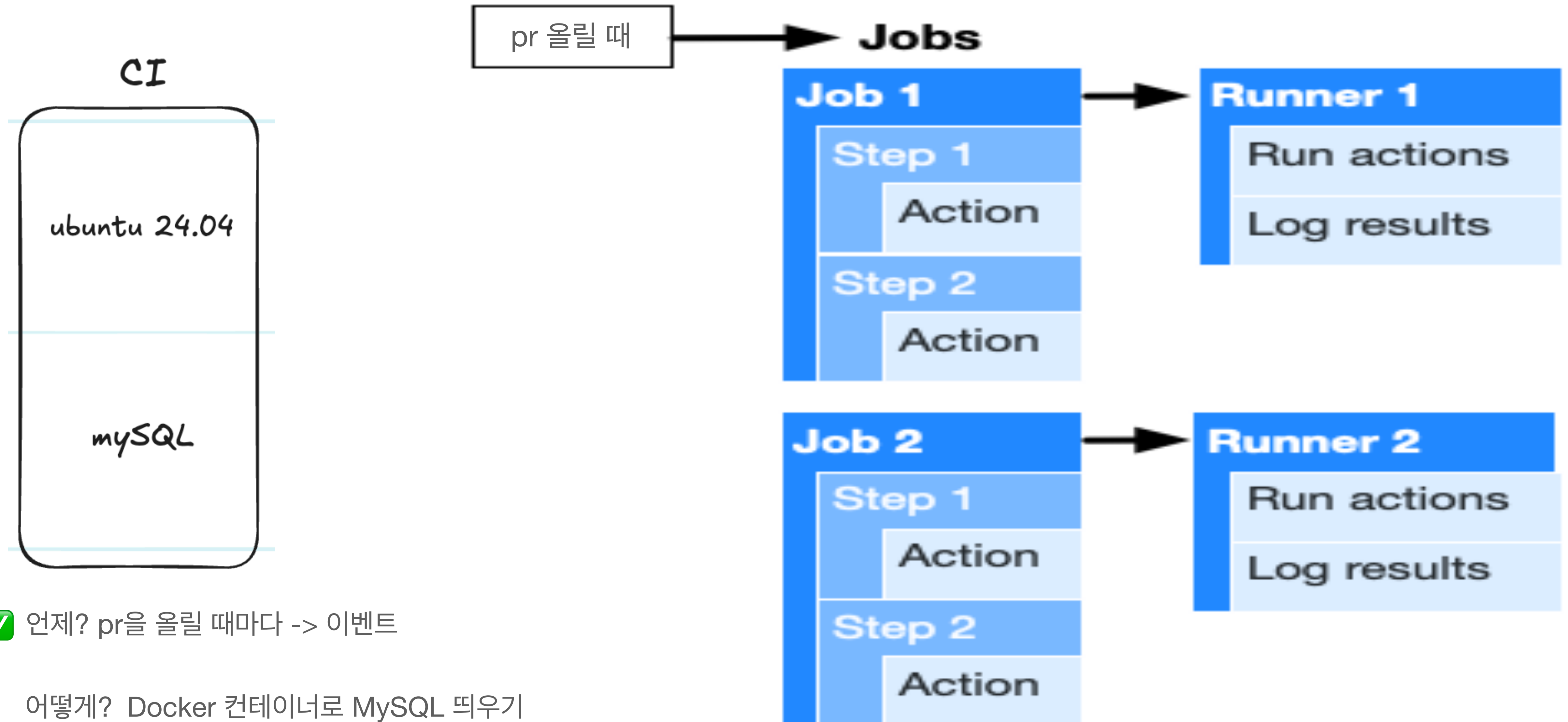




## 2.3. 운영 환경에 따른 Github Actions



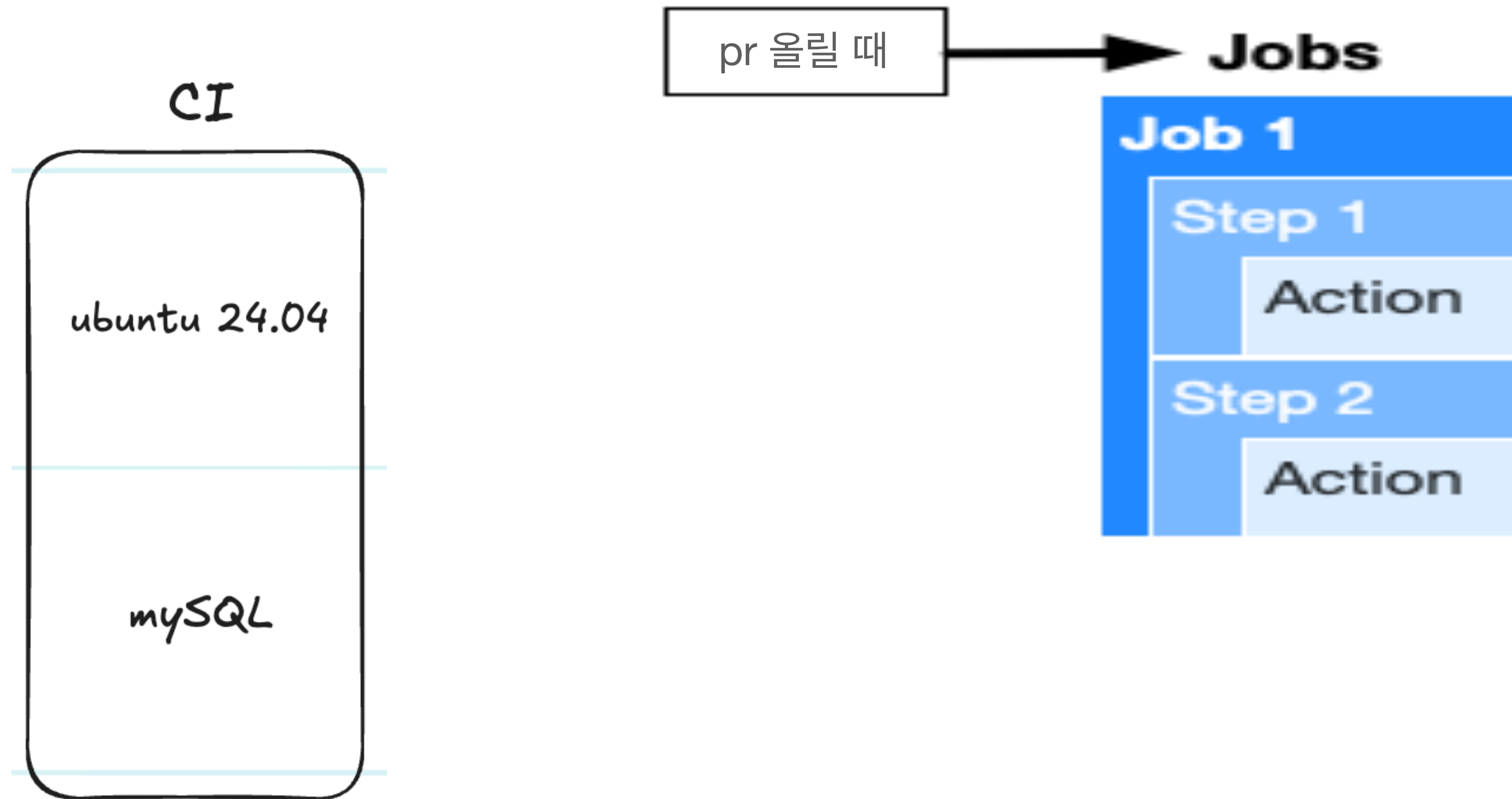
## 2.3. 운영 환경에 따른 Github Actions



✓ 언제? pr을 올릴 때마다 -> 이벤트

어떻게? Docker 컨테이너로 MySQL 띄우기

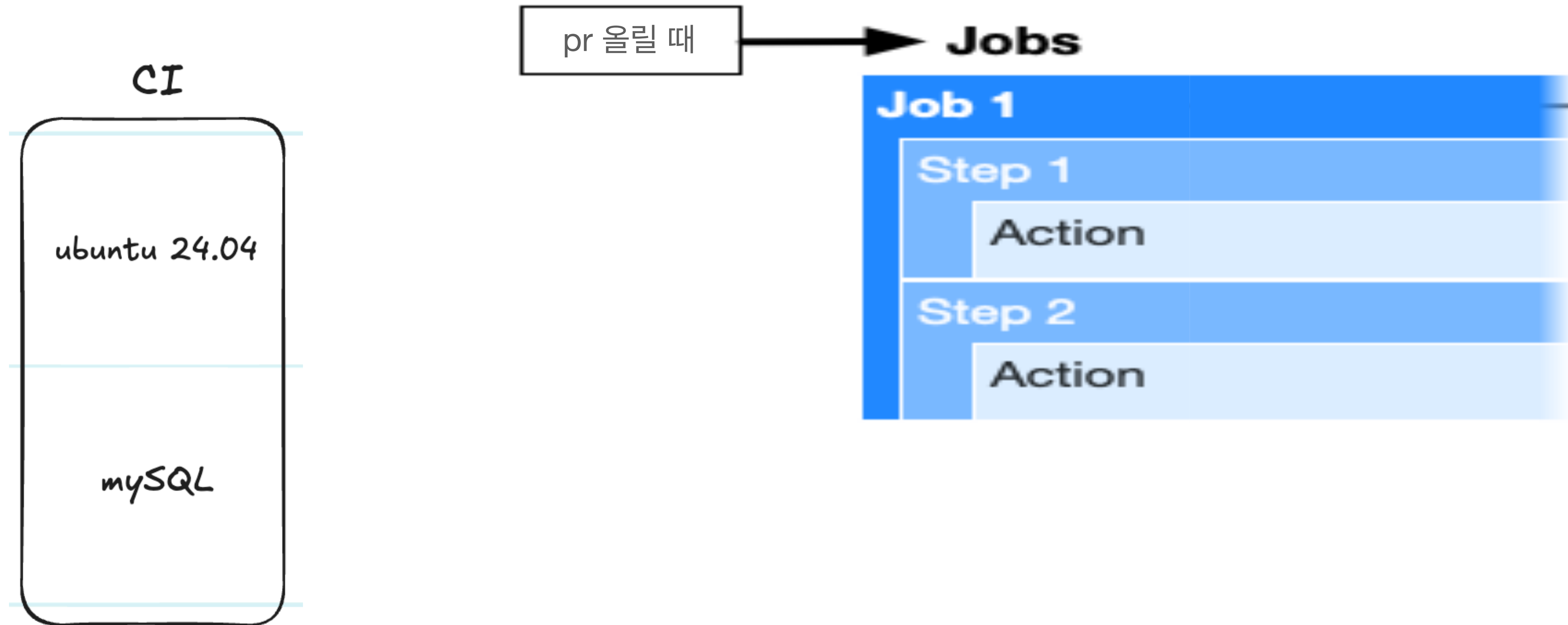
## 2.3. 운영 환경에 따른 Github Actions



✓ 언제? pr을 올릴 때마다 -> 이벤트

어떻게? Docker 컨테이너로 MySQL 띄우기

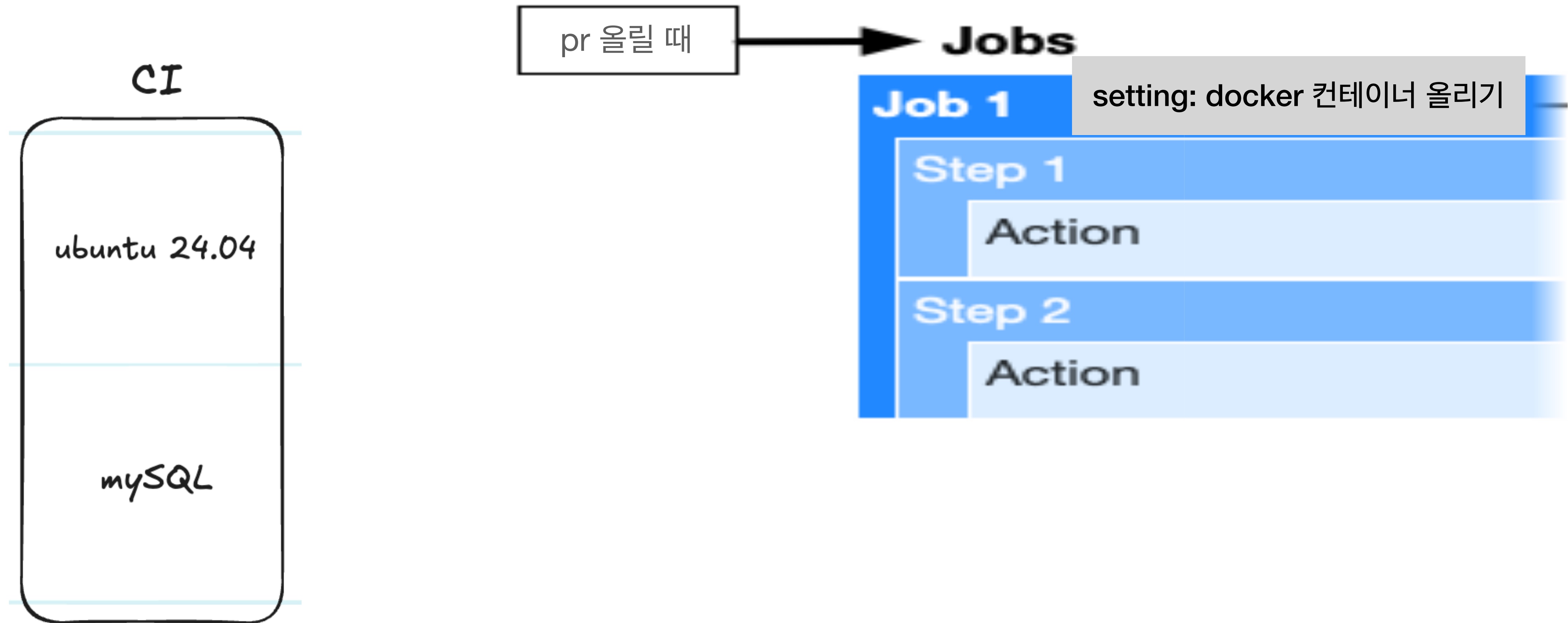
## 2.3. 운영 환경에 따른 Github Actions



✓ 언제? pr을 올릴 때마다 -> 이벤트

어떻게? Docker 컨테이너로 MySQL 띄우기

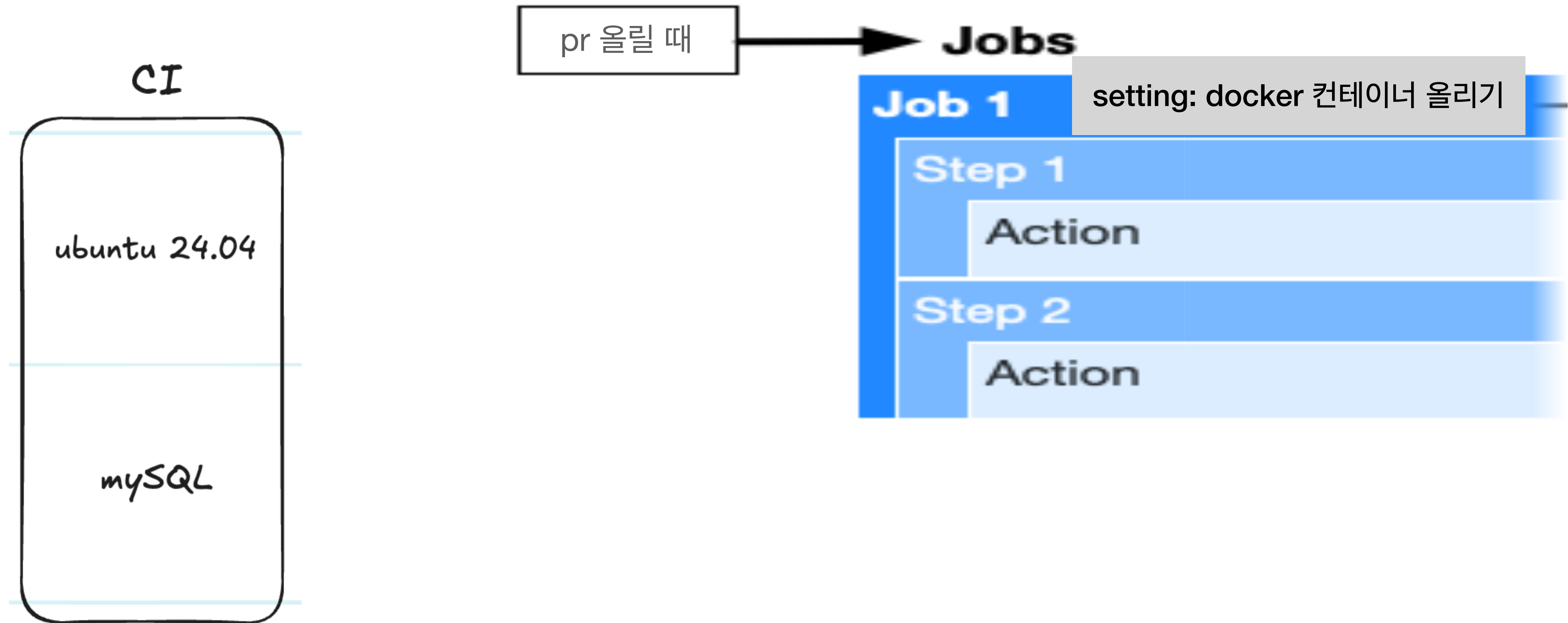
## 2.3. 운영 환경에 따른 Github Actions



✓ 언제? pr을 올릴 때마다 -> 이벤트

어떻게? Docker 컨테이너로 MySQL 띄우기

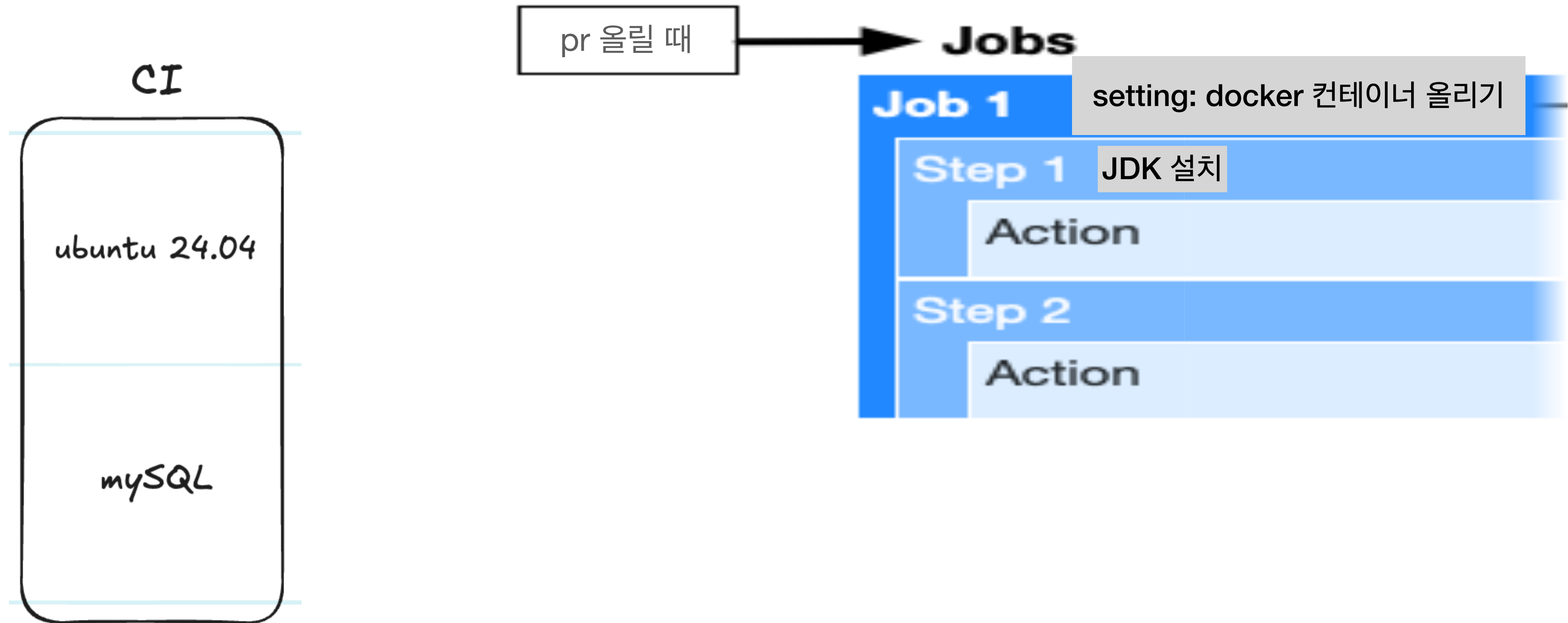
## 2.3. 운영 환경에 따른 Github Actions



✓ 언제? pr을 올릴 때마다 -> 이벤트

✓ 어떻게? Docker 컨테이너로 MySQL 띄우기

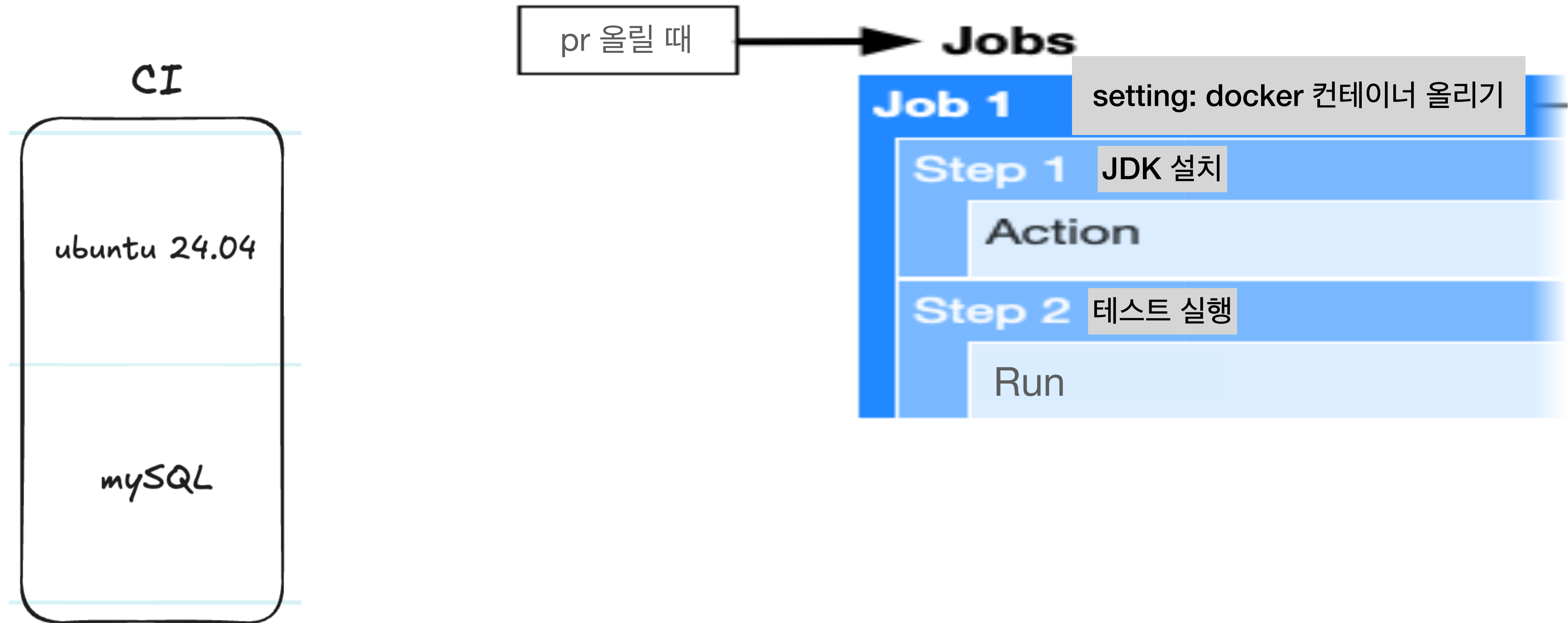
## 2.3. 운영 환경에 따른 Github Actions



✓ 언제? pr을 올릴 때마다 -> 이벤트

✓ 어떻게? Docker 컨테이너로 MySQL 띄우기

## 2.3. 운영 환경에 따른 Github Actions

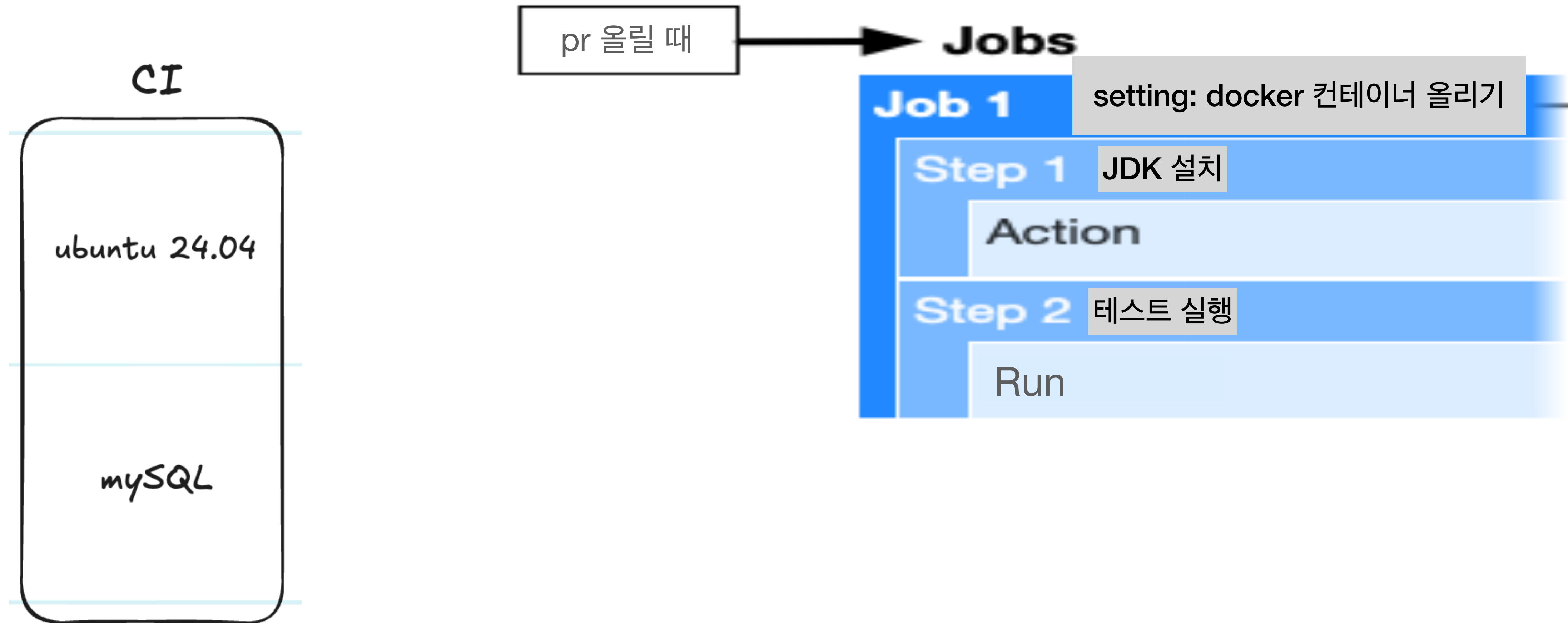


✓ 언제? pr을 올릴 때마다 -> 이벤트

✓ 어떻게? Docker 컨테이너로 MySQL 띄우기



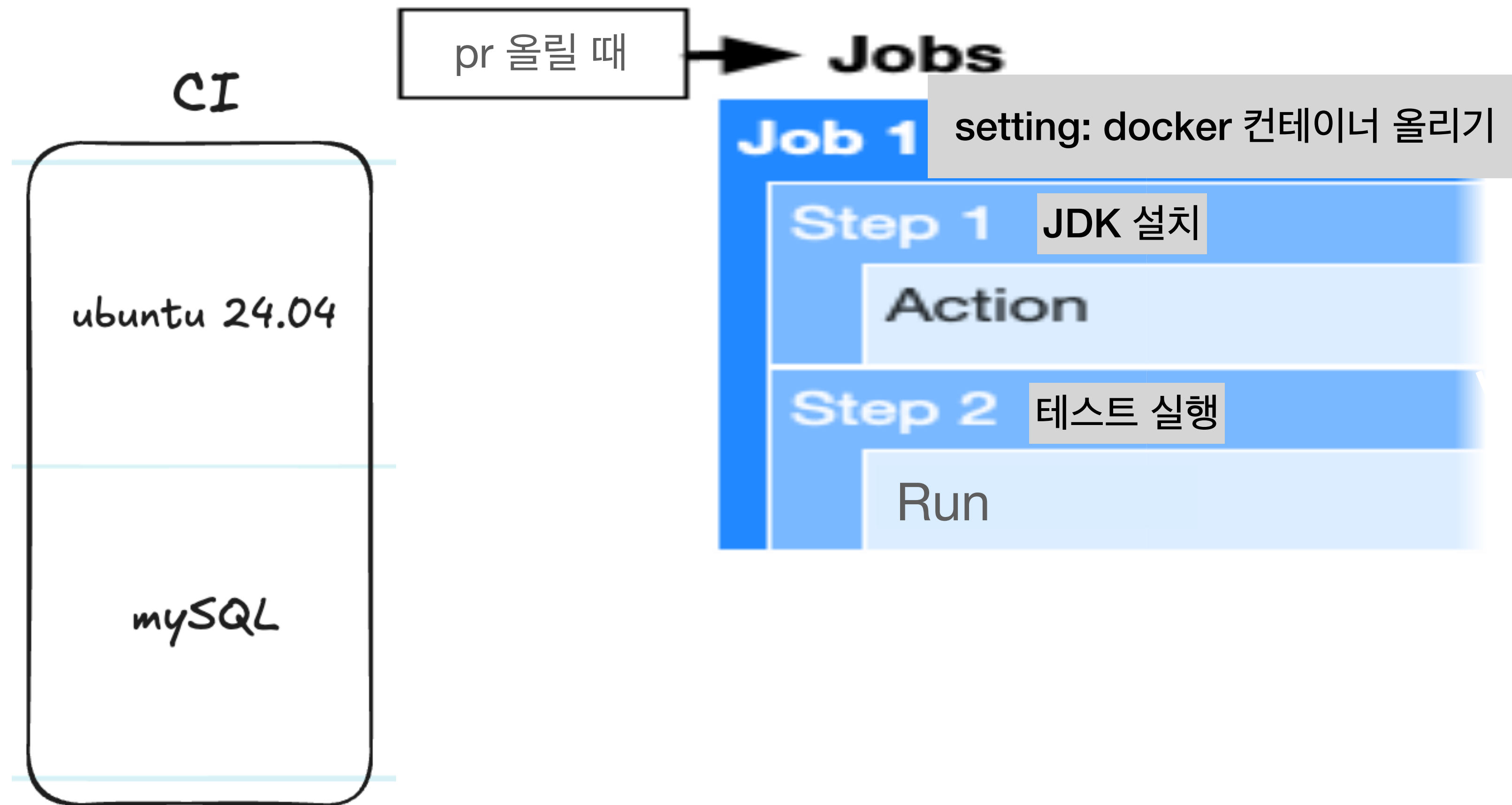
## 2.3. 운영 환경에 따른 Github Actions



✓ 언제? pr을 올릴 때마다 -> 이벤트

✓ 어떻게? Docker 컨테이너로 MySQL 띄우기

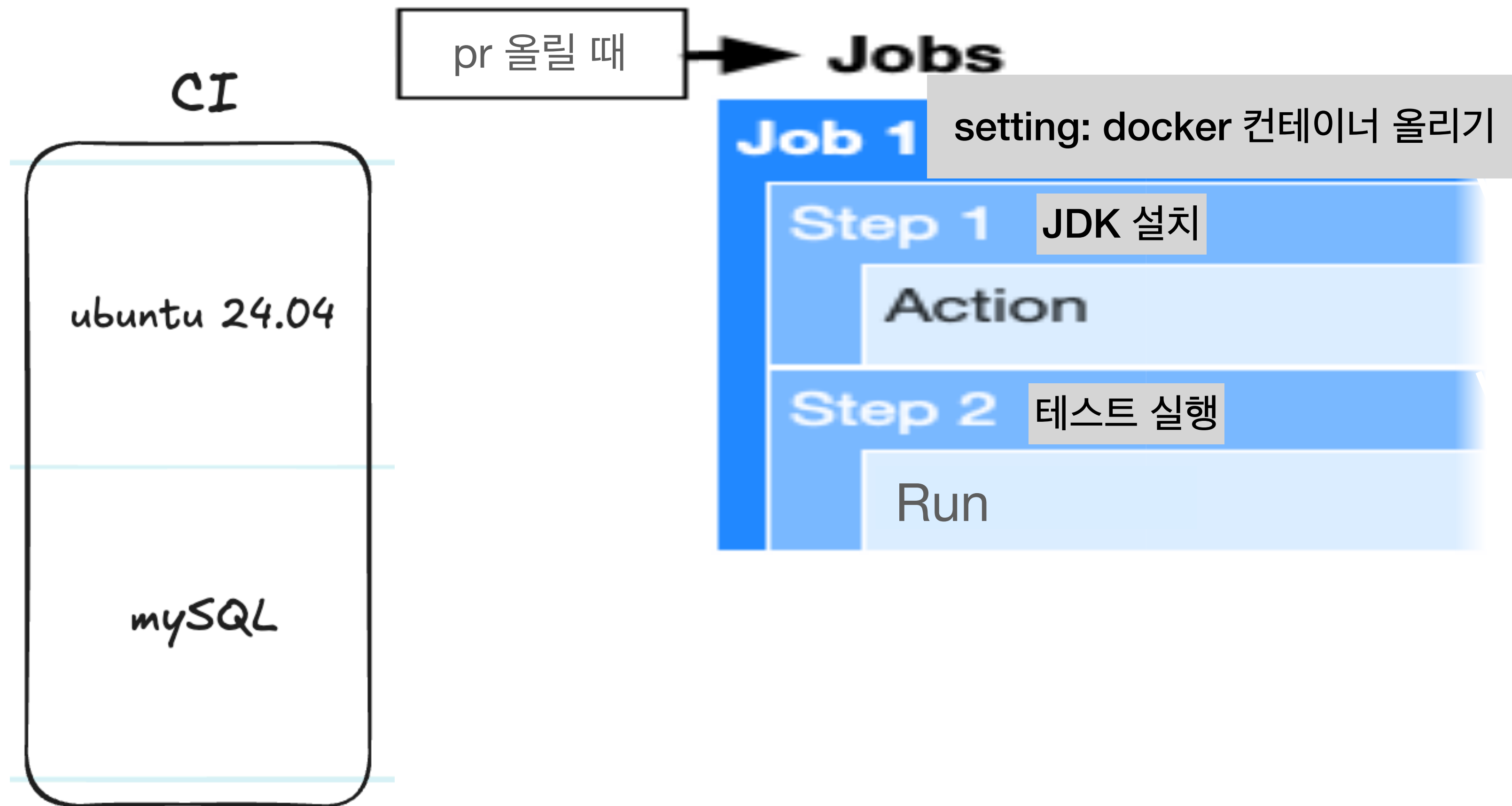
## 2.3. 운영 환경에 따른 Github Actions



✓ 언제? pr을 올릴 때마다 -> 이벤트

✓ 어떻게? Docker 컨테이너로 MySQL 띄우기

## 2.3. 운영 환경에 따른 Github Action

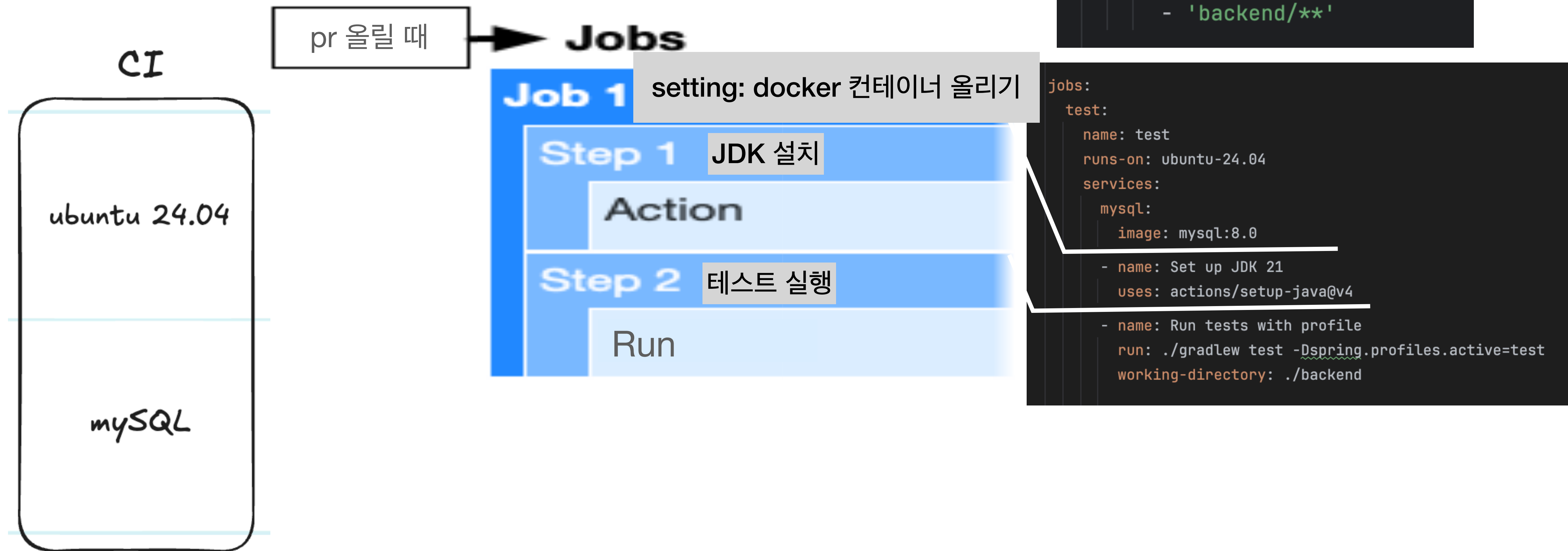


```
on:
  pull_request:
    branches: [ "develop" ]
    paths:
      - 'backend/**'
```

✓ 언제? pr을 올릴 때마다 -> 이벤트

✓ 어떻게? Docker 컨테이너로 MySQL 띄우기

## 2.3. 운영 환경에 따른 Github Action



✓ 언제? pr을 올릴 때마다 -> 이벤트

✓ 어떻게? Docker 컨테이너로 MySQL 띄우기

## 2.3. 운영 환경에 따른 Github Actions



## 2.3. 운영 환경에 따른 Github Actions



## 2.3. 운영 환경에 따른 Github Actions



언제? pr이 merge 될 때마다 -> 이벤트

## 2.3. 운영 환경에 따른 Github Actions



언제? pr이 merge 될 때마다 -> 이벤트

어떻게?



## 2.3. 운영 환경에 따른 Github Actions



언제? pr이 merge 될 때마다 -> 이벤트

어떻게?

1. 직접 jar 파일 ec2 내에서 실행하기
2. 도커 이미지로 관리하기

## 2.3. 운영 환경에 따른 Github Actions



언제? pr이 merge 될 때마다 -> 이벤트

어떻게?

1. 직접 jar 파일 ec2 내에서 실행하기
2. 도커 이미지로 관리하기

1. RDS
2. 도커 이미지로 관리하기
3. 직접 ec2 내에 mysql 실행하기

## 2.3. 운영 환경에 따른 Github Actions



언제? pr이 merge 될 때마다 -> 이벤트

어떻게?

1. 직접 jar 파일 ec2 내에서 실행하기
2. 도커 이미지로 관리하기

1. RDS
2. 도커 이미지로 관리하기
3. 직접 ec2 내에 mysql 실행하기

## 2.3. 운영 환경에 따른 Github Actions



언제? pr이 merge 될 때마다 -> 이벤트

어떻게?

1. 직접 jar 파일 ec2 내에서 실행하기
2. 도커 이미지로 관리하기

1. RDS
2. 도커 이미지로 관리하기
3. 직접 ec2 내에 mysql 실행하기

## 2.3. 운영 환경에 따른 Github Actions



언제? pr이 merge 될 때마다 -> 이벤트

어떻게?

1. 직접 jar 파일 ec2 내에서 실행하기
2. ~~도커 이미지로 관리하기~~

1. ~~RDS~~
2. ~~도커 이미지로 관리하기~~
3. 직접 ec2 내에 ~~mysql~~ 실행하기

## 2.3. 운영 환경에 따른 Github Actions

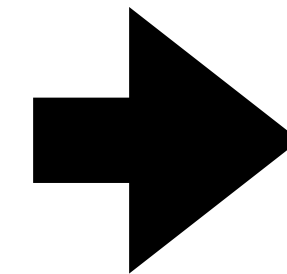


언제? pr이 merge 될 때마다 -> 이벤트

어떻게?

1. 직접 jar 파일 ec2 내에서 실행하기
2. ~~도커 이미지로 관리하기~~

1. ~~도커 이미지로 관리하기~~ RDS
2. ~~도커 이미지로 관리하기~~
3. ~~직접 ec2 내에 mysql 실행하기~~



ec2 터짐

## 2.3. 운영 환경에 따른 Github Actions

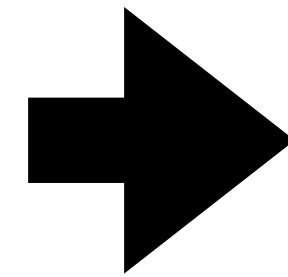


언제? pr이 merge 될 때마다 -> 이벤트

어떻게?

1. 직접 jar 파일 ec2 내에서 실행하기
2. 도커 이미지로 관리하기

1. RDS
2. 도커 이미지로 관리하기
3. 직접 ec2 내에 mysql 실행하기



ec2 터짐

원인 추측: ec2의 용량 부족

## 2.3. 운영 환경에 따른 Github Actions

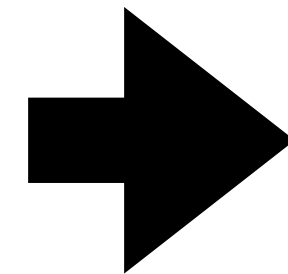


언제? pr이 merge 될 때마다 -> 이벤트

어떻게?

1. 직접 jar 파일 ec2 내에서 실행하기
2. 도커 이미지로 관리하기

1. RDS
2. 도커 이미지로 관리하기
3. 직접 ec2 내에 mysql 실행하기



ec2 터짐

원인 추측: ec2의 용량 부족

ec2 용량 증가 (8gb -> 30gb)  
swap 메모리 용량 증가 (2gb -> 8gb)



## 2.3. 운영 환경에 따른 Github Actions

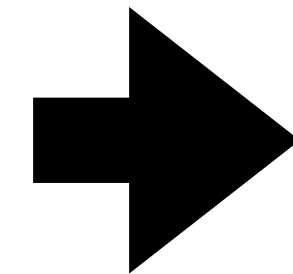


언제? pr이 merge 될 때마다 -> 이벤트

어떻게?

1. 직접 jar 파일 ec2 내에서 실행하기
2. 도커 이미지로 관리하기

1. RDS
2. 도커 이미지로 관리하기
3. 직접 ec2 내에 mysql 실행하기



ec2 터짐

원인 추측: ec2의 용량 부족

ec2 용량 증가 (8gb -> 30gb)  
swap 메모리 용량 증가 (2gb -> 8gb)

-> 해결 됨

## 2.3. 운영 환경에 따른 Github Actions



언제? pr이 merge 될 때마다 -> 이벤트

어떻게?

1. 직접 jar 파일 ec2 내에서 실행하기
2. 도커 이미지로 관리하기

1. RDS
2. 도커 이미지로 관리하기
3. 직접 ec2 내에 mysql 실행하기

## 2.3. 운영 환경에 따른 Github Actions



언제? pr이 merge 될 때마다 -> 이벤트

어떻게?

도커 이미지로 관리하기

도커 이미지로 관리하기

## 2.3. 운영 환경에 따른 Github Actions



언제? pr이 merge 될 때마다 -> 이벤트

어떻게? 소스 코드 + mysql 모두 도커 이미지로 관리하기

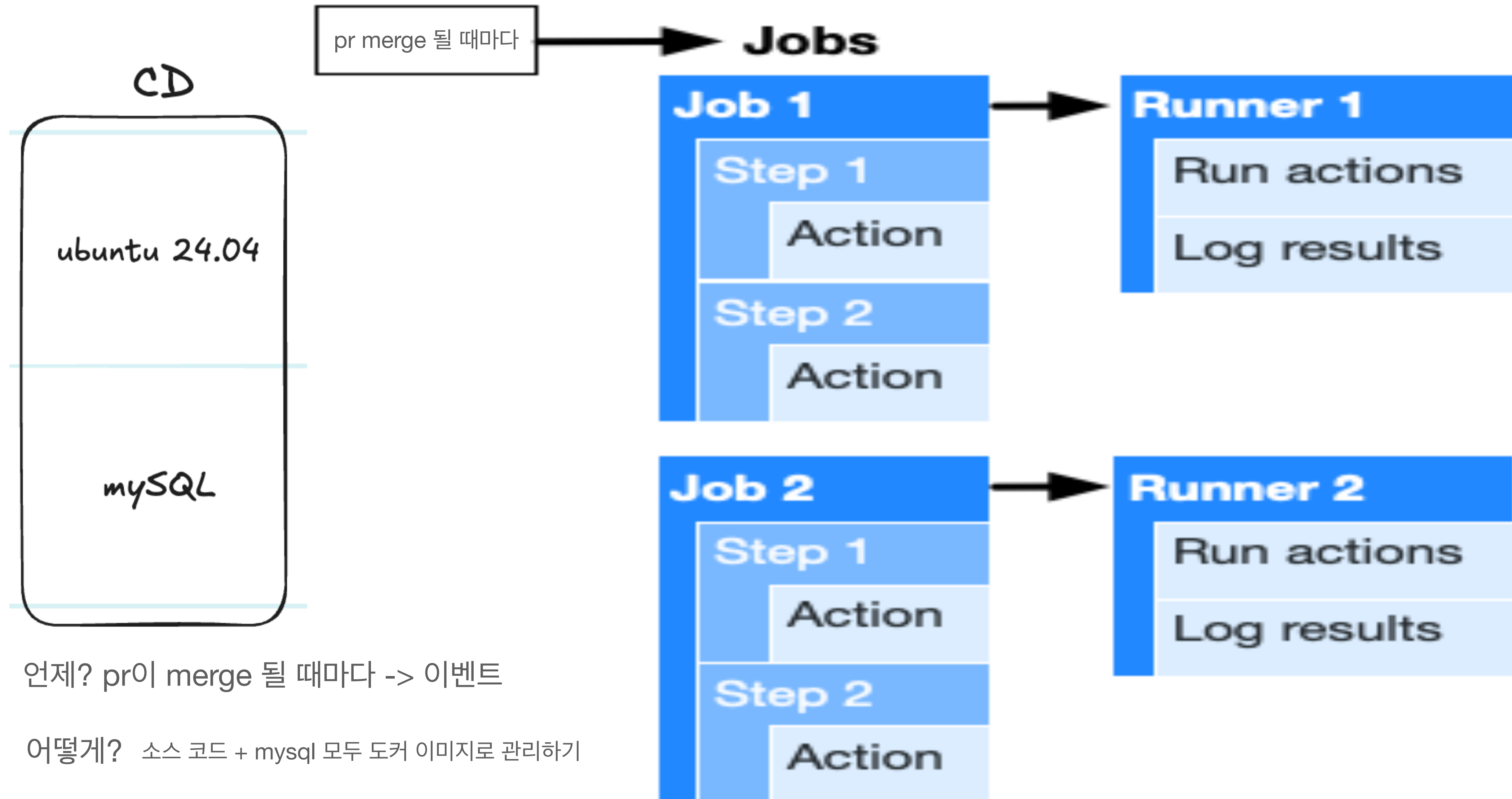
## 2.3. 운영 환경에 따른 Github Actions



언제? pr이 merge 될 때마다 -> 이벤트

어떻게? 소스 코드 + mysql 모두 도커 이미지로 관리하기

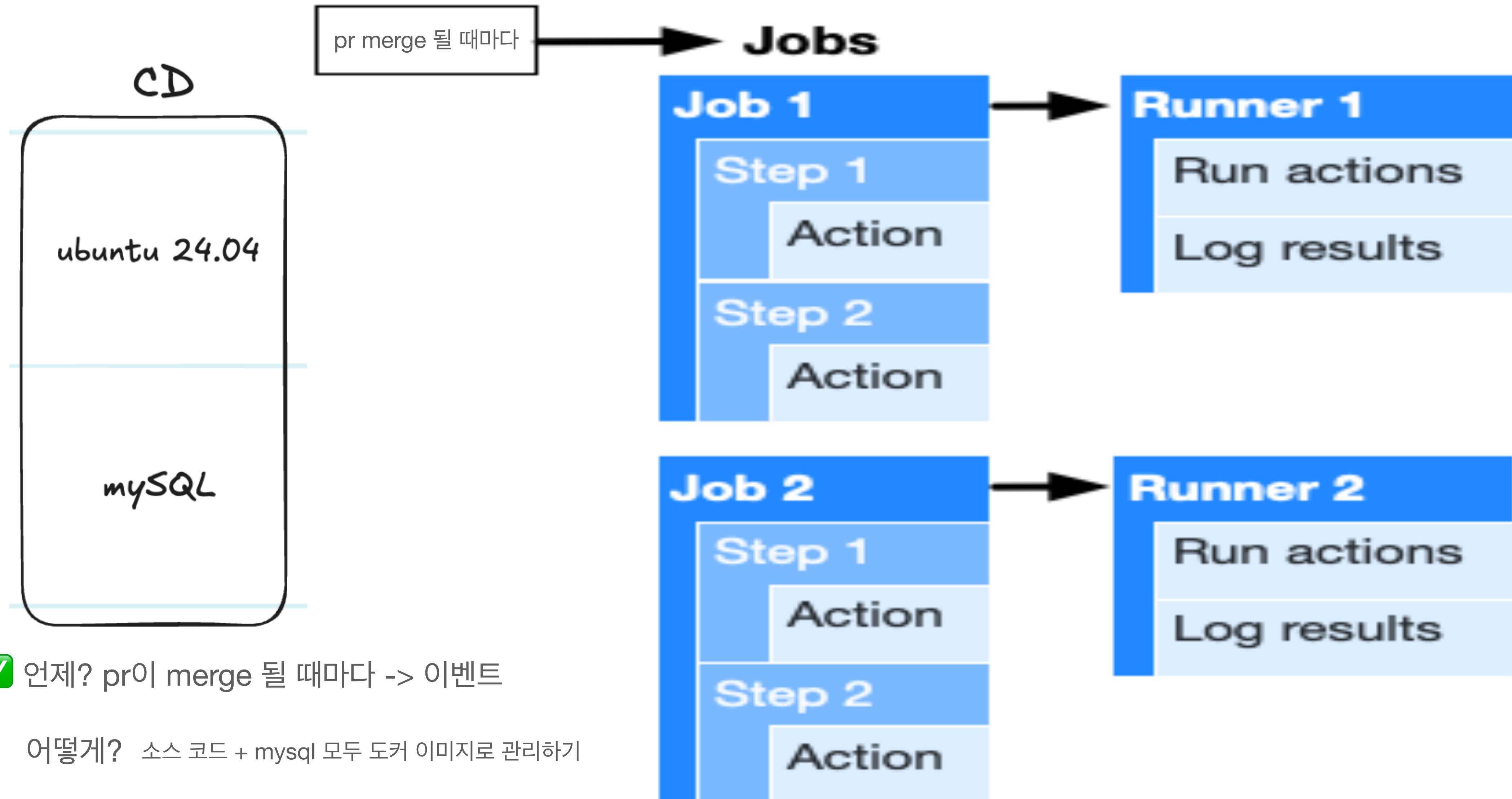
## 2.3. 운영 환경에 따른 Github Actions



언제? pr이 merge 될 때마다 -> 이벤트

어떻게? 소스 코드 + mysql 모두 도커 이미지로 관리하기

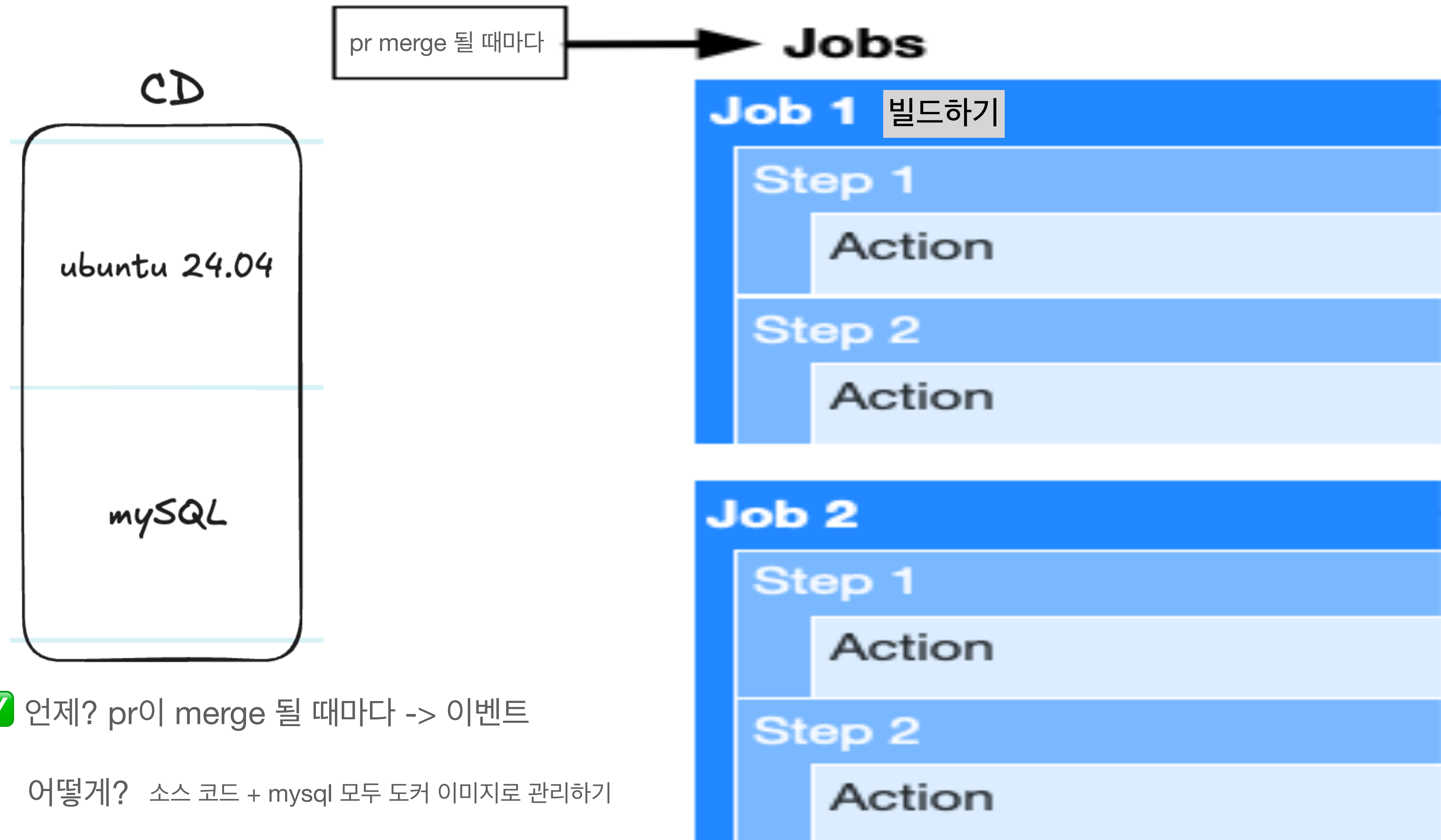
## 2.3. 운영 환경에 따른 Github Actions



✓ 언제? pr이 merge 될 때마다 -> 이벤트

어떻게? 소스 코드 + mysql 모두 도커 이미지로 관리하기

## 2.3. 운영 환경에 따른 Github Actions

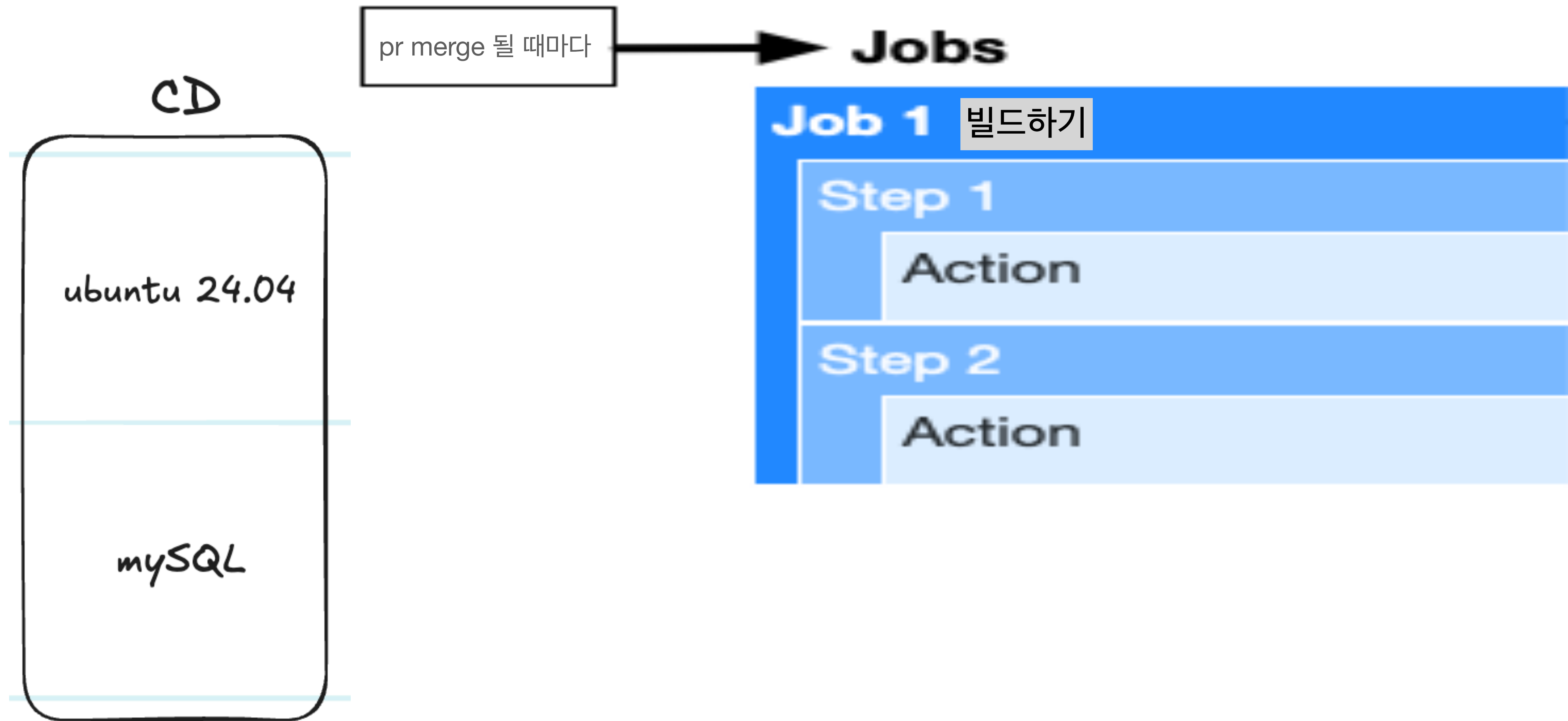


✓ 언제? pr이 merge 될 때마다 -> 이벤트

어떻게? 소스 코드 + mysql 모두 도커 이미지로 관리하기



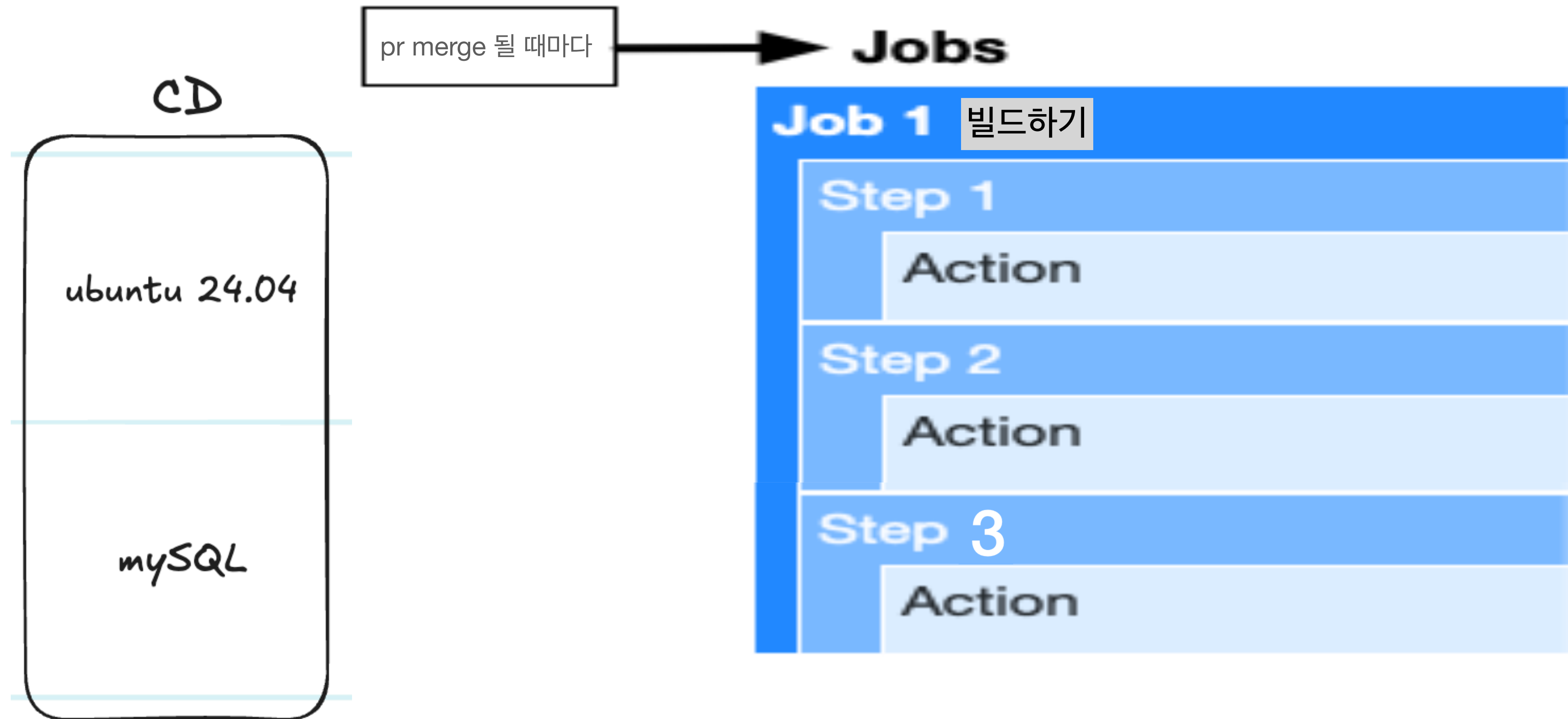
## 2.3. 운영 환경에 따른 Github Actions



✓ 언제? pr이 merge 될 때마다 -> 이벤트

어떻게? 소스 코드 + mysql 모두 도커 이미지로 관리하기

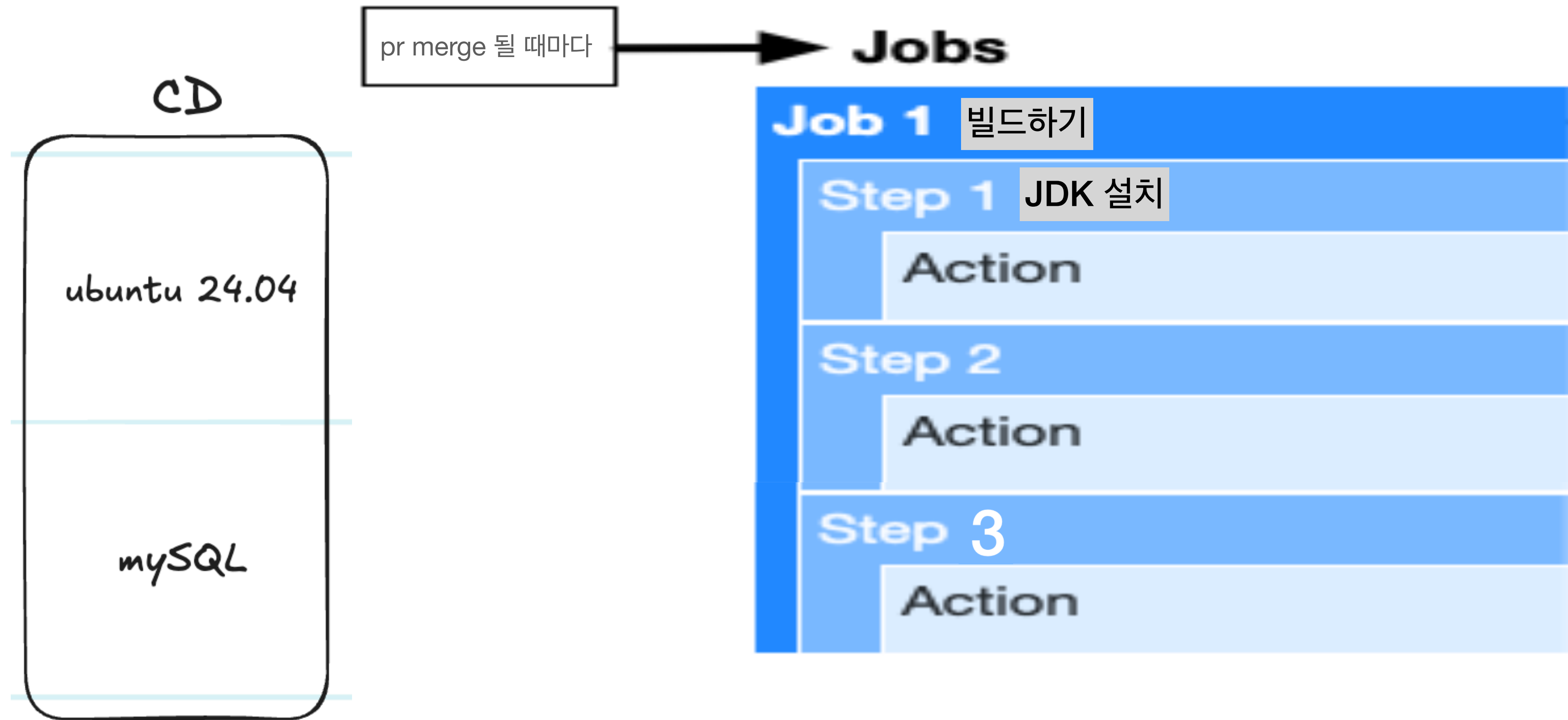
## 2.3. 운영 환경에 따른 Github Actions



✓ 언제? pr이 merge 될 때마다 -> 이벤트

어떻게? 소스 코드 + mysql 모두 도커 이미지로 관리하기

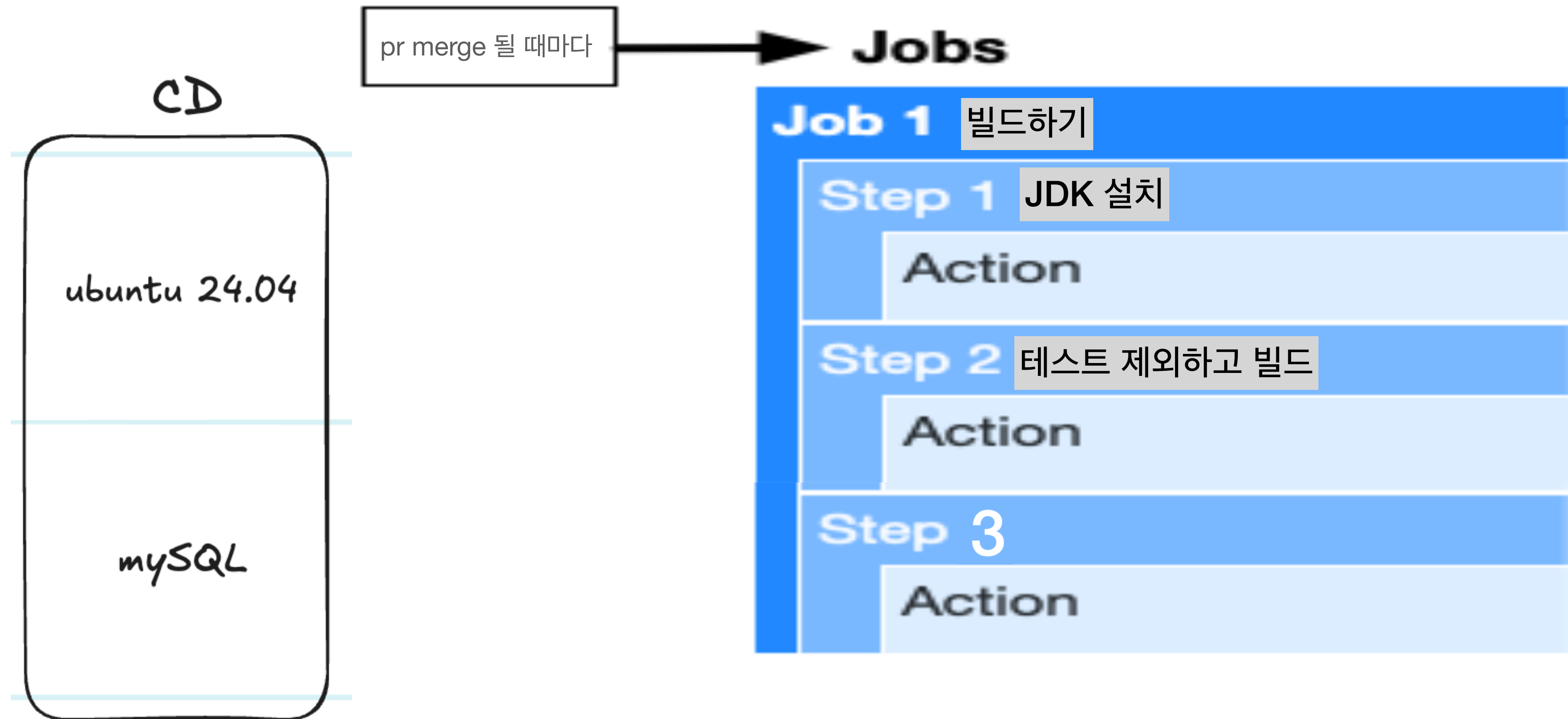
## 2.3. 운영 환경에 따른 Github Actions



✓ 언제? pr이 merge 될 때마다 -> 이벤트

어떻게? 소스 코드 + mysql 모두 도커 이미지로 관리하기

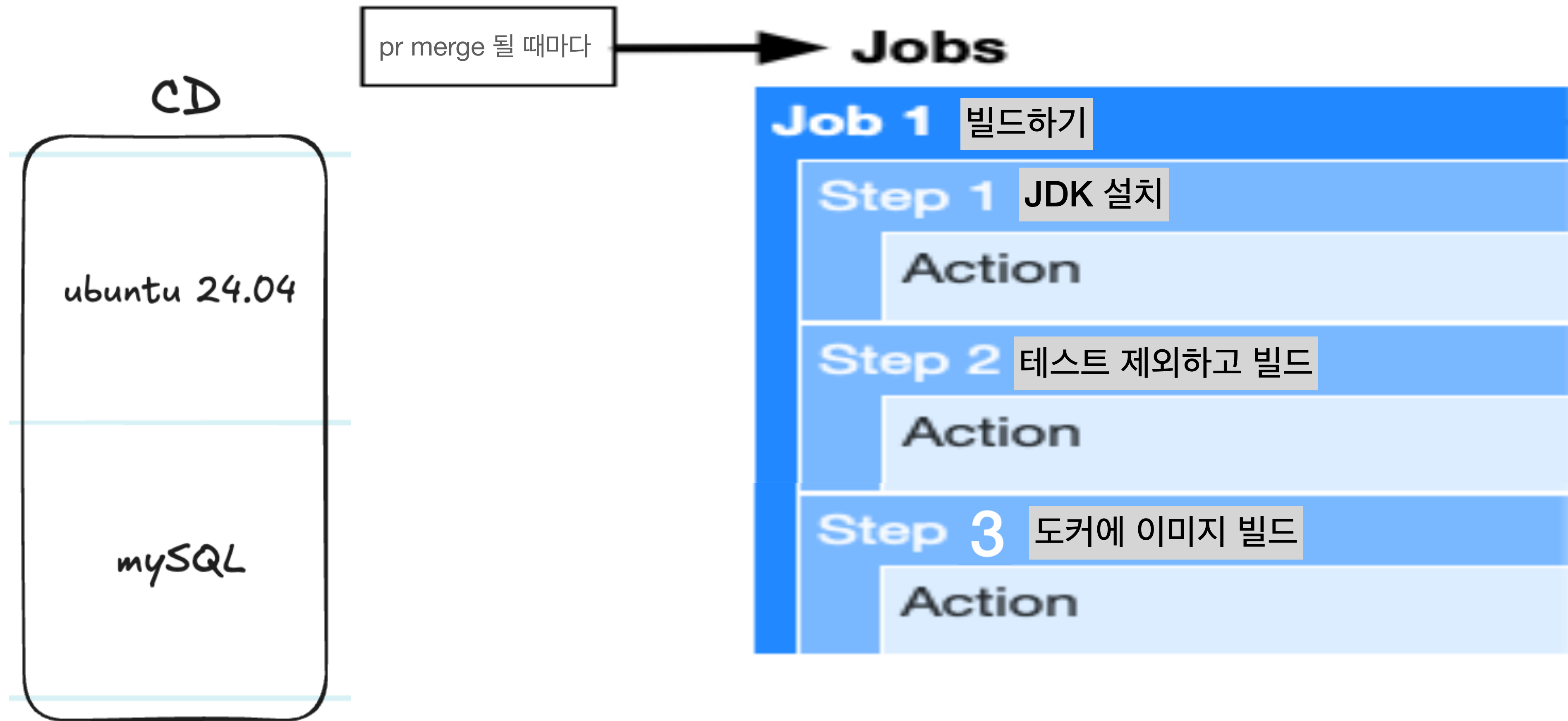
## 2.3. 운영 환경에 따른 Github Actions



✓ 언제? pr이 merge 될 때마다 -> 이벤트

어떻게? 소스 코드 + mysql 모두 도커 이미지로 관리하기

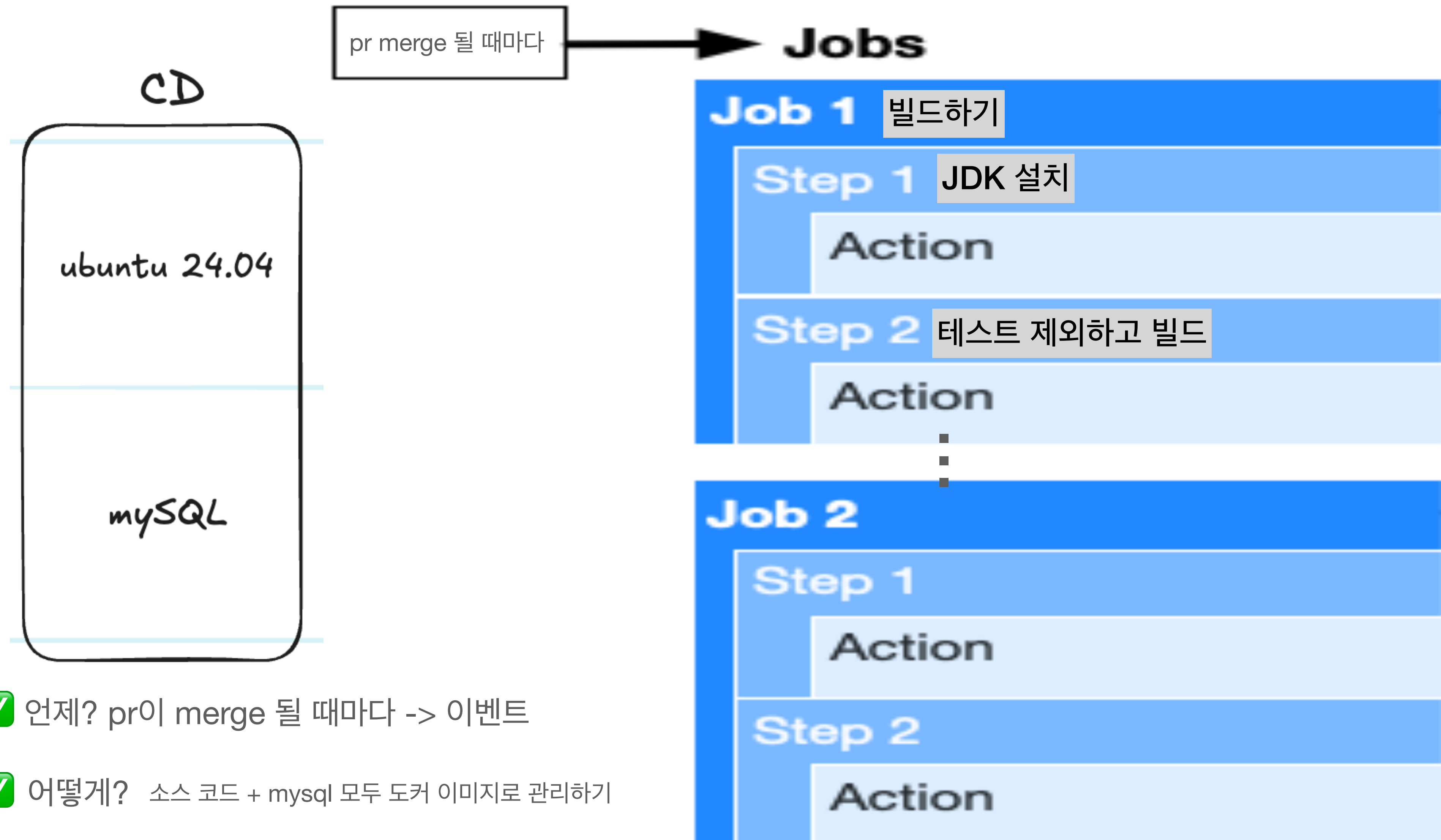
## 2.3. 운영 환경에 따른 Github Actions



✓ 언제? pr이 merge 될 때마다 -> 이벤트

✓ 어떻게? 소스 코드 + mysql 모두 도커 이미지로 관리하기

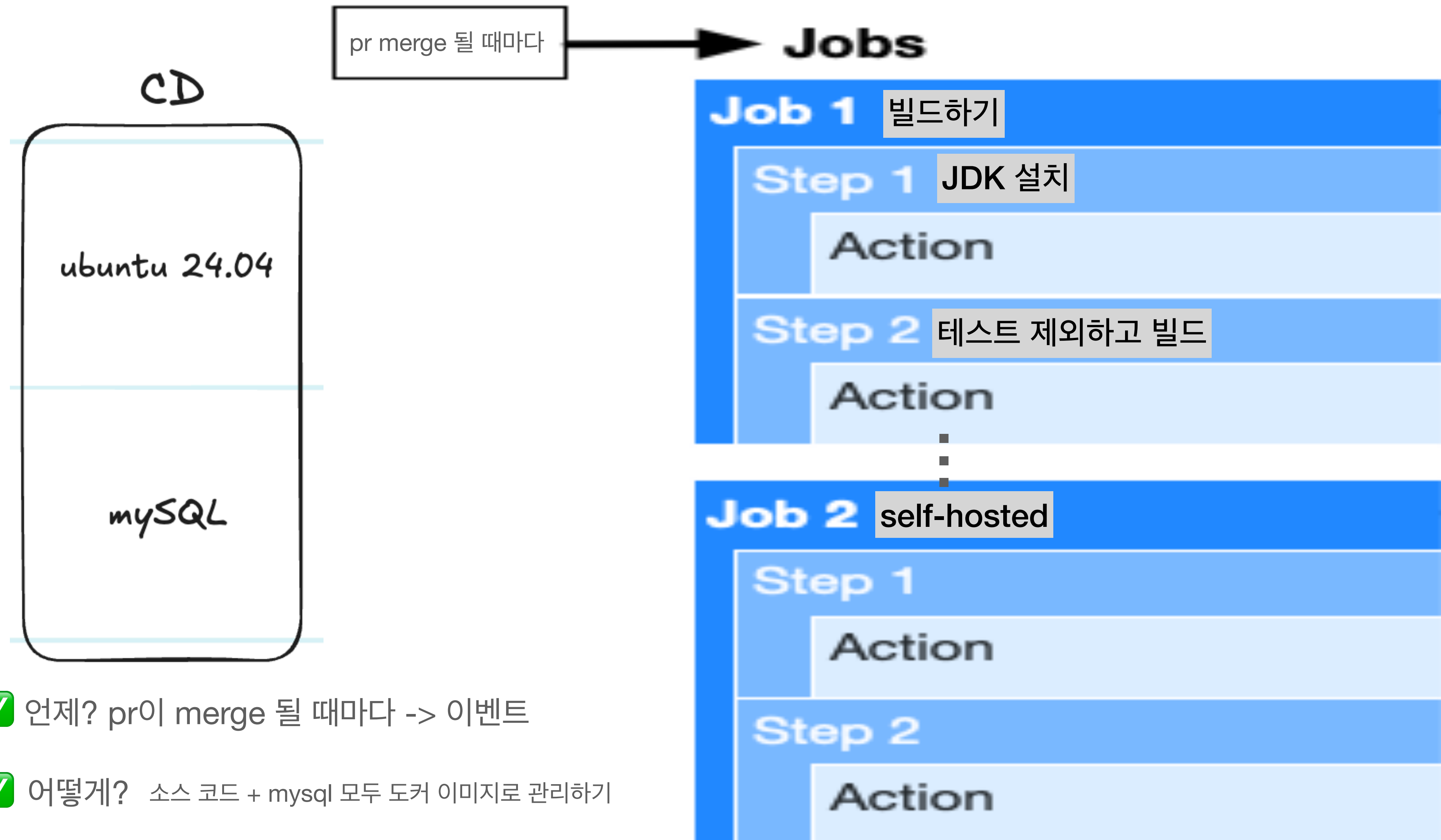
## 2.3. 운영 환경에 따른 Github Actions



✓ 언제? pr이 merge 될 때마다 -> 이벤트

✓ 어떻게? 소스 코드 + mysql 모두 도커 이미지로 관리하기

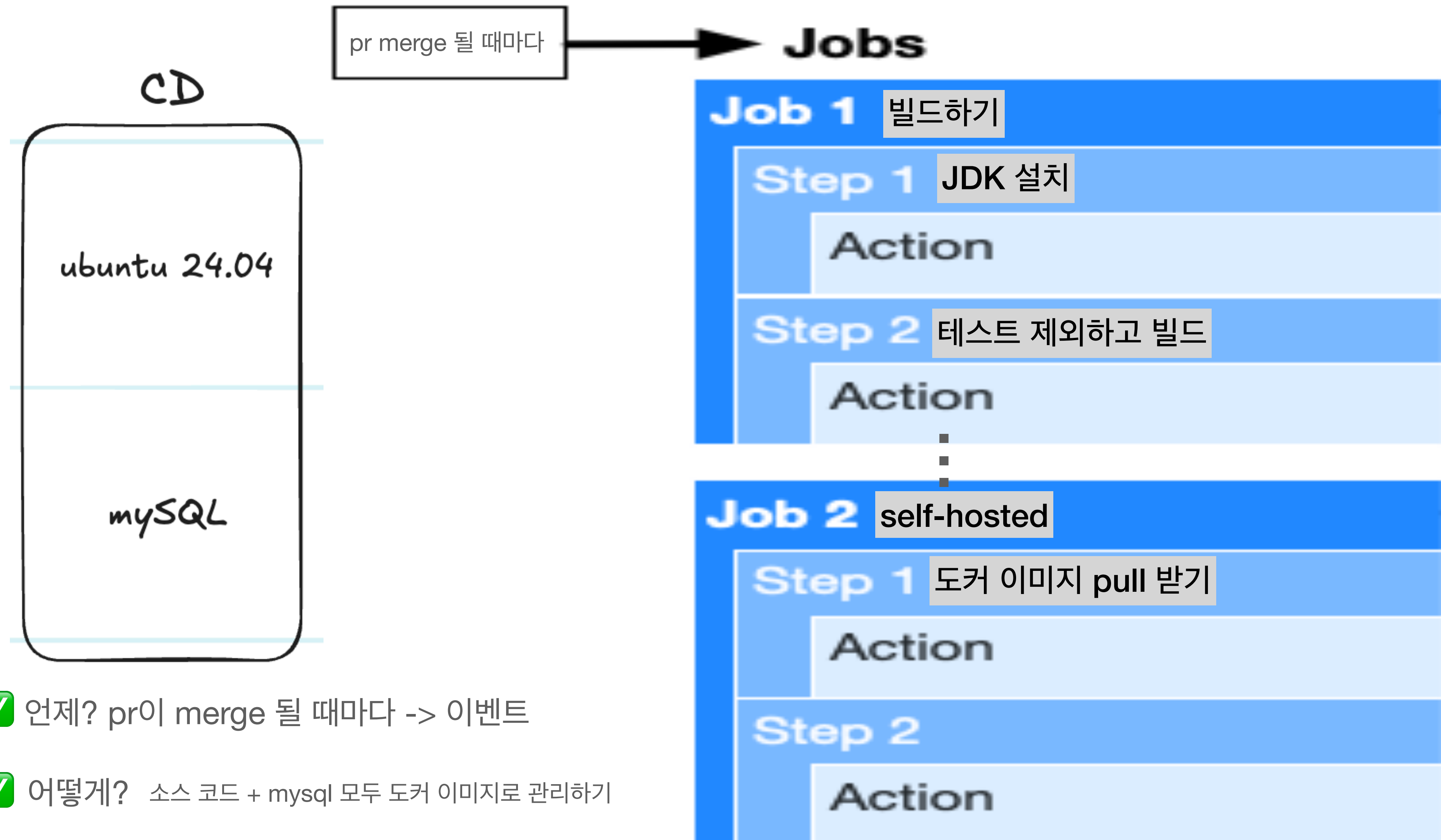
## 2.3. 운영 환경에 따른 Github Actions



✓ 언제? pr이 merge 될 때마다 -> 이벤트

✓ 어떻게? 소스 코드 + mysql 모두 도커 이미지로 관리하기

## 2.3. 운영 환경에 따른 Github Actions

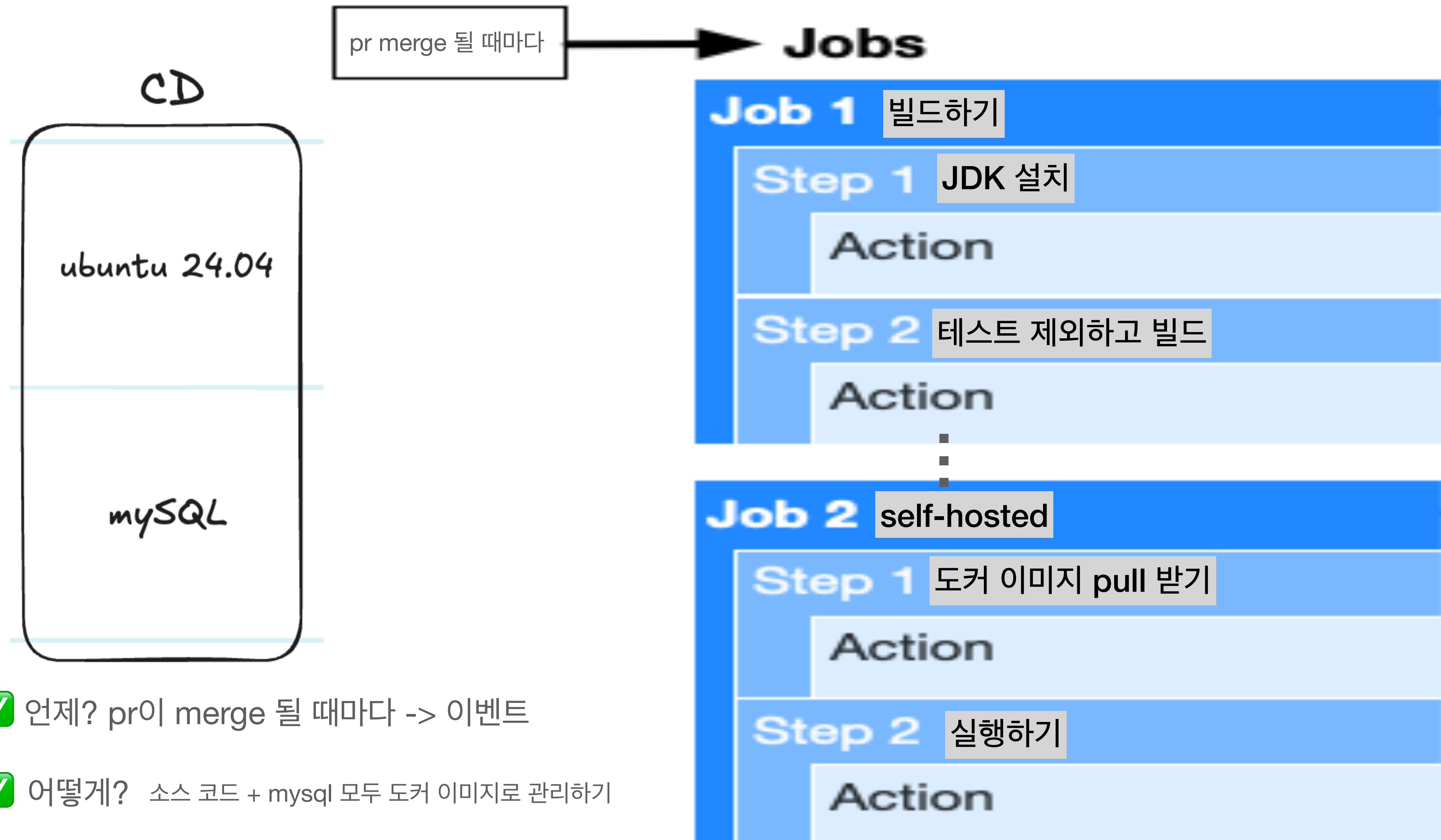


✓ 언제? pr이 merge 될 때마다 -> 이벤트

✓ 어떻게? 소스 코드 + mysql 모두 도커 이미지로 관리하기



## 2.3. 운영 환경에 따른 Github Actions

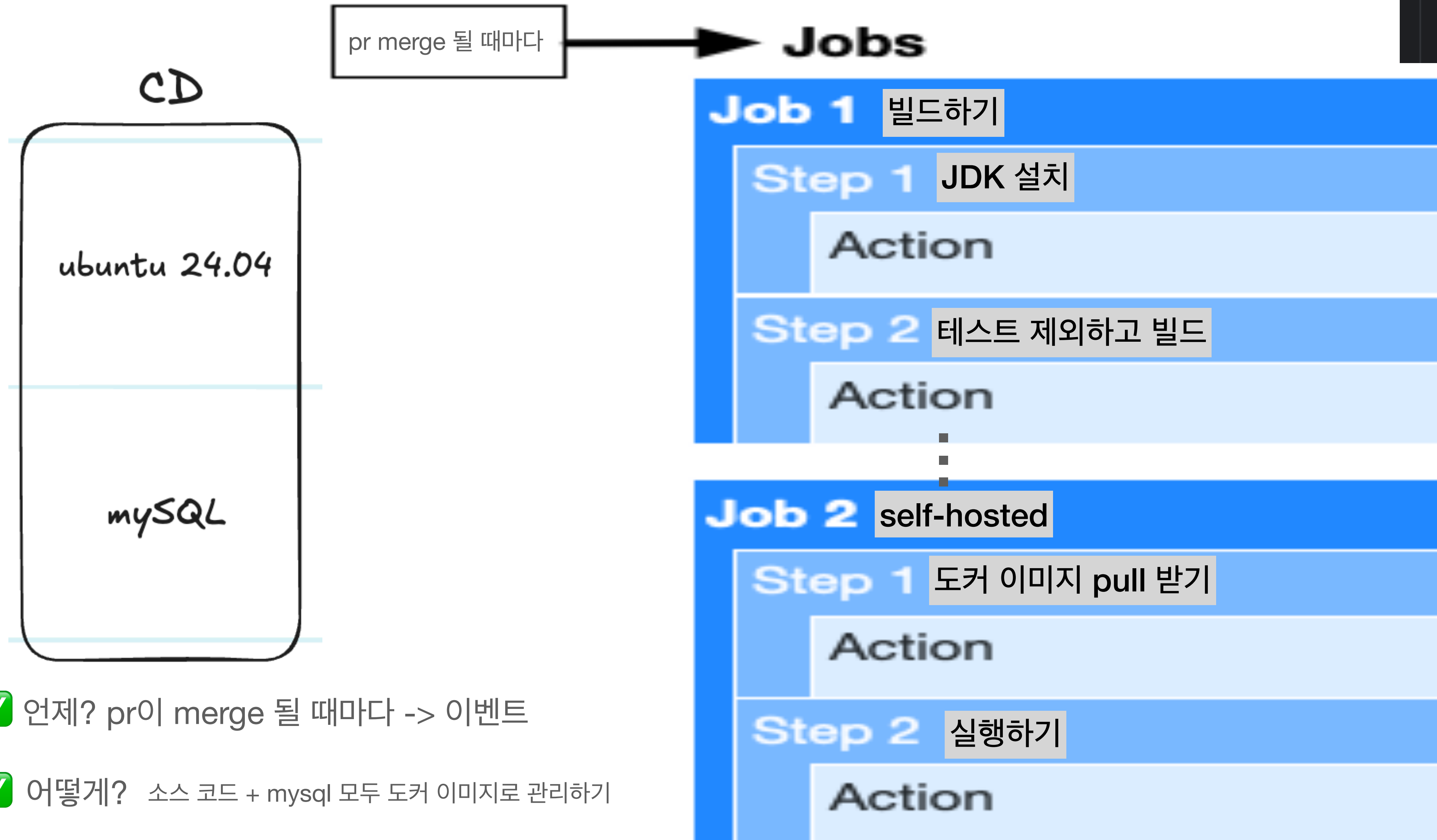


✓ 언제? pr이 merge 될 때마다 -> 이벤트

✓ 어떻게? 소스 코드 + mysql 모두 도커 이미지로 관리하기

## 2.3. 운영 환경에 따른 Github Actions

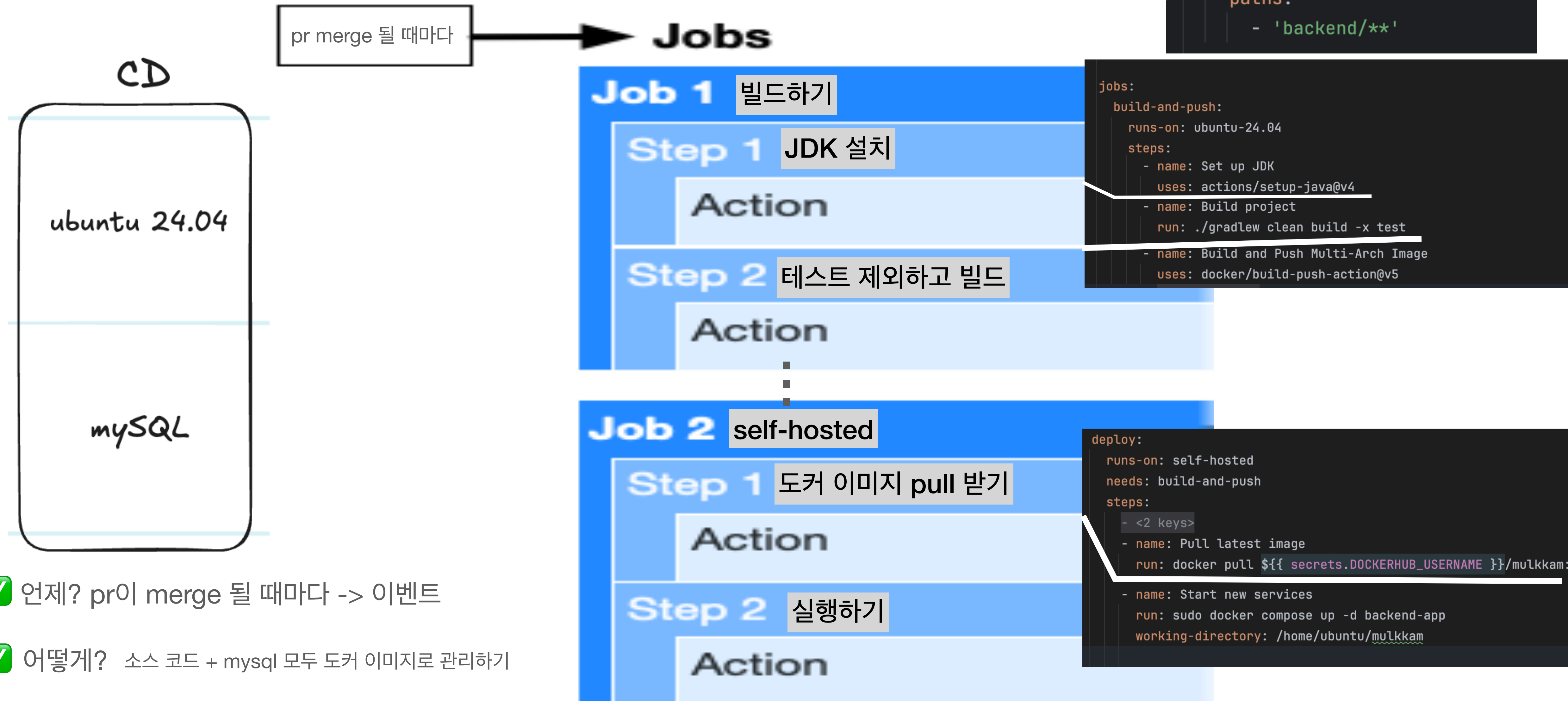
```
on:
  push:
    branches: [ "develop" ]
    paths:
      - 'backend/**'
```



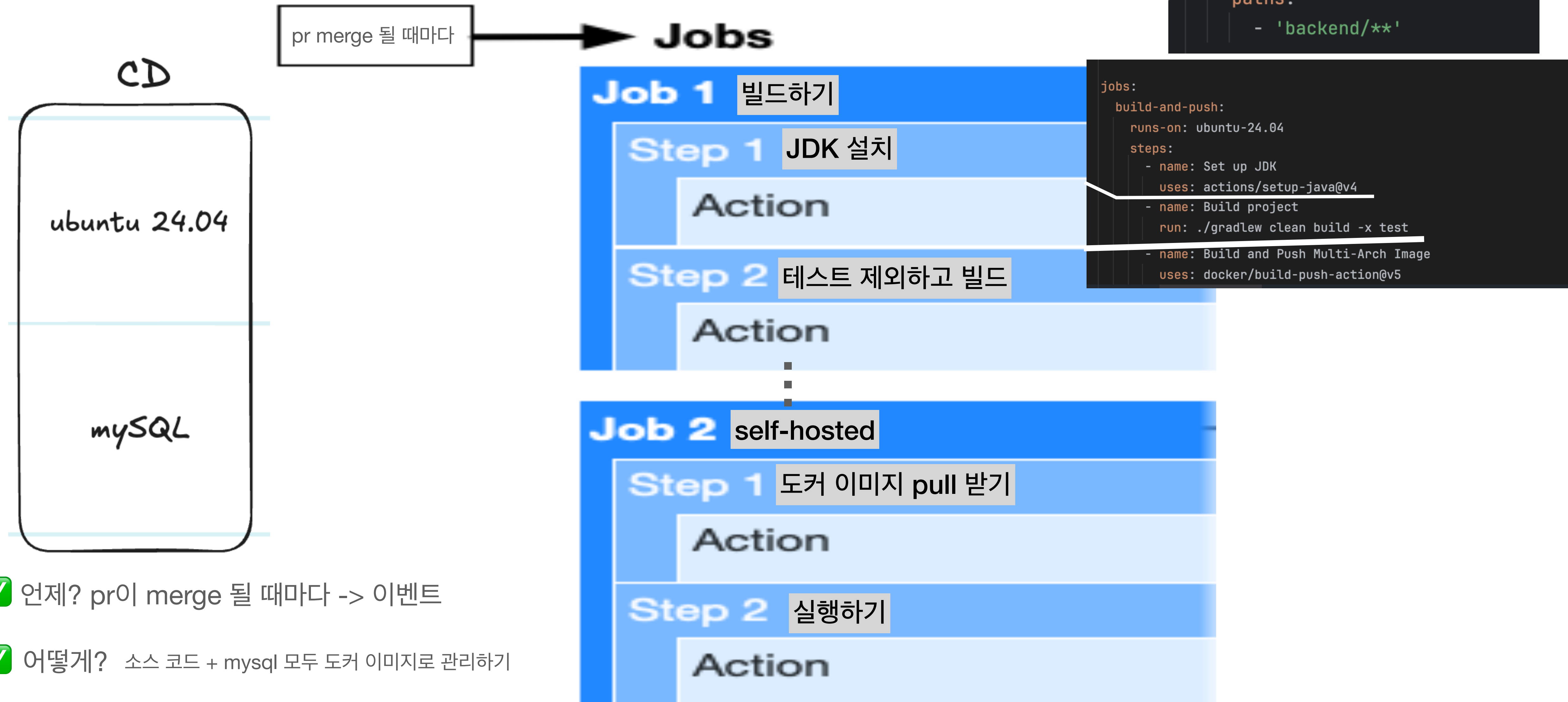
✓ 언제? pr이 merge 될 때마다 -> 이벤트

✓ 어떻게? 소스 코드 + mysql 모두 도커 이미지로 관리하기

## 2.3. 운영 환경에 따른 Github Actions



## 2.3. 운영 환경에 따른 Github Actions



✓ 언제? pr이 merge 될 때마다 -> 이벤트

✓ 어떻게? 소스 코드 + mysql 모두 도커 이미지로 관리하기

# 3. 운영 환경 안정화를 위한 인프라 개선 사항

## 1 merge 직전, PR이 승인된 상태에서 dev 브랜치의 최신 변경 사항을 병합 시도

- 충돌 없으면 그대로 merge 진행
- 충돌 발생 시 IntelliJ로 수동 해결

## 2 멀티플랫폼 이미지 사용 제거

- 팀원 전원이 macOS 환경 사용
- 단일 플랫폼으로도 충분하여 멀티플랫폼 설정 불필요

## 3 Docker 최대 메모리 사용량 제한 설정

- 운영환경 메모리 전체 사용 방지 목적
- 컨테이너별 메모리 한도 지정으로 안정성 확보

## 4 EC2 인스턴스 새로 생성

- 기존 인스턴스에 메모리 및 swap 메모리 과도하게 할당됨
- 실제 사용량 기준으로 적절한 메모리 설정 후 재구성

# 감사합니다.