

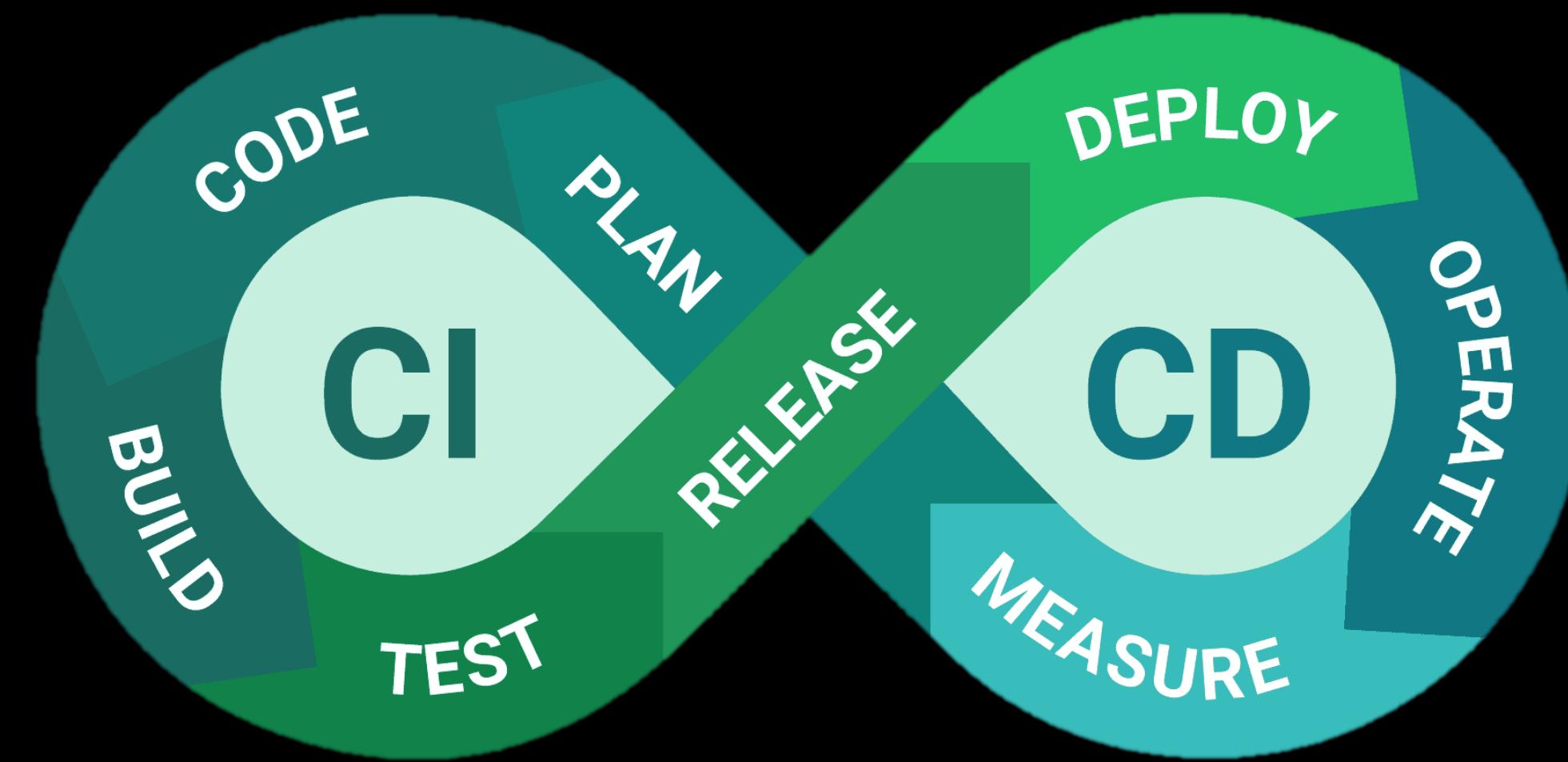
QA와 지속적 통합/배포



CI/CD의 정의

CI(Continuous Integration)

코드 변경 사항을 정기적으로 통합하고 자동화된 빌드 및 테스트를 수행



CD(Continuous Deployment)

변경된 코드를 자동으로 배포하여 사용자에게 제공



CI/CD의 주요 프로세스



1. 코드 변경 사항을 저장소에 커밋



2. 자동으로 빌드 및 테스트 실행



3. 테스트 성공 시 배포 자동화

CI/CD 파이프라인: 코드를 통합하고 배포하는 일련의 단계
빌드(Build), 테스트(Test), 배포(Deploy)



지속적 통합(CI)의 개념

지속적 통합(CI)

개발자들이 작성한 코드를 정기적으로 병합하고, 자동 빌드 및 테스트 실행

주요 역할

- 코드 변경 시 품질 문제를 조기에 발견
- 개발팀 간 코드 충돌 최소화



Gitlab CI/CD



Jenkins



Circle CI



CI의 주요 장점



코드 충돌 조기 해결



자동화된 테스트로 품질 보증



개발 속도 향상

효과적인 CI 구현

- 작은 단위로 자주 커밋
- 테스트 자동화 스크립트 작성



지속적 배포(CD)의 개념

지속적 배포(CD)

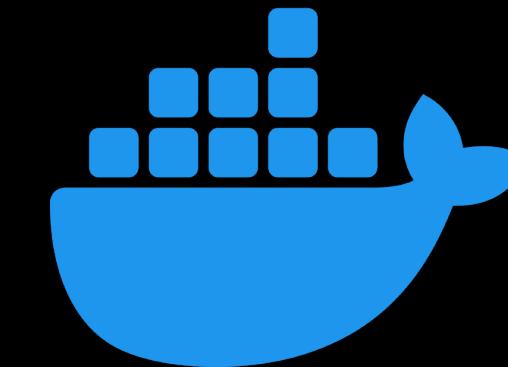
통합된 코드를 자동으로 프로덕션 환경에 배포

주요 역할

- 개발과 배포 주기를 단축
- 코드 변경 사항을 사용자에게 신속히 제공



Spinnaker



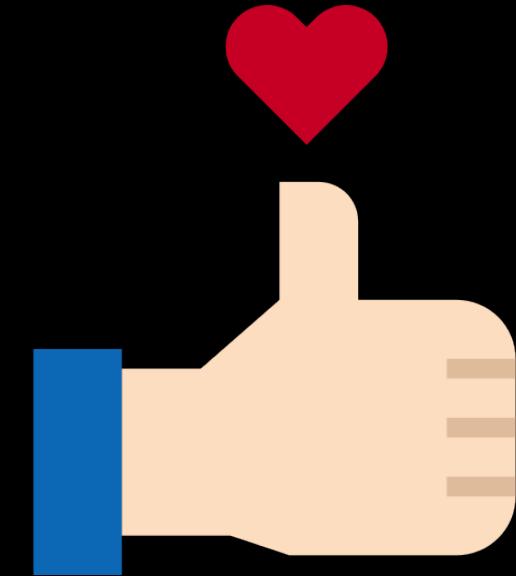
Docker



Kubernetes



CD의 주요 장점과 과제



주요 장점

- 배포 주기 단축
- 사용자 피드백 반영 속도 증가



과제

- 프로덕션 환경에서의 위험 관리
- 배포 후 문제 발생 시 롤백 계획 필요

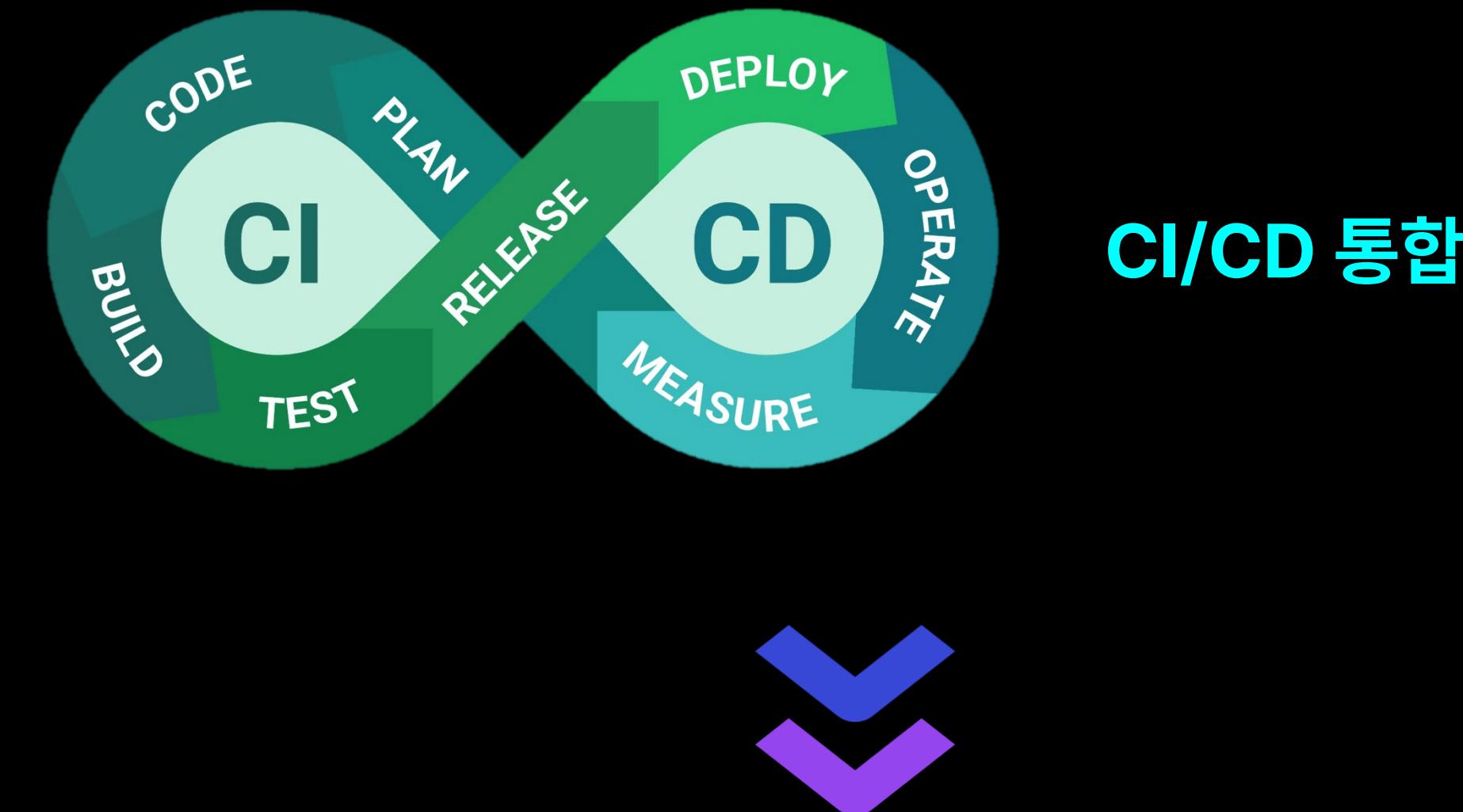


CI와 CD의 차이점 이해

구분	CI (Continuous Integration)	CD (Continuous Delivery/Deployment)
주요 목적	개발자 코드의 통합 및 자동화된 테스트	테스트된 코드를 배포 환경에 자동 전달
핵심 단계	개발 → 코드 병합 → 빌드 → 테스트	테스트 → 준비된 배포 → 자동 배포
자동화 범위	빌드 및 테스트 단계까지 자동화	배포(Delivery) 또는 배포 + 릴리스(Deployment)까지 자동화
주요 이점	코드 충돌 감소, 빠른 피드백 제공	지속적인 배포, 사용자에게 빠르게 기능 제공
사용 도구 예시	Jenkins, GitHub Actions, GitLab CI/CD	Jenkins, Spinnaker, CircleCI
결과물	테스트를 통과한 빌드	사용자에게 배포된 코드 또는 배포 준비 상태의 코드
적용 환경	개발 환경 (Dev Environment)	스테이징/프로덕션 환경 (Staging/Production)
흐름 예시	개발 → 빌드 → 테스트	테스트 → 배포 → 사용자 제공



CI/CD 통합의 효과



- 전체 개발-테스트-배포 프로세스의 자동화
- 신속한 코드 릴리스
- 사용자 피드백의 빠른 반영



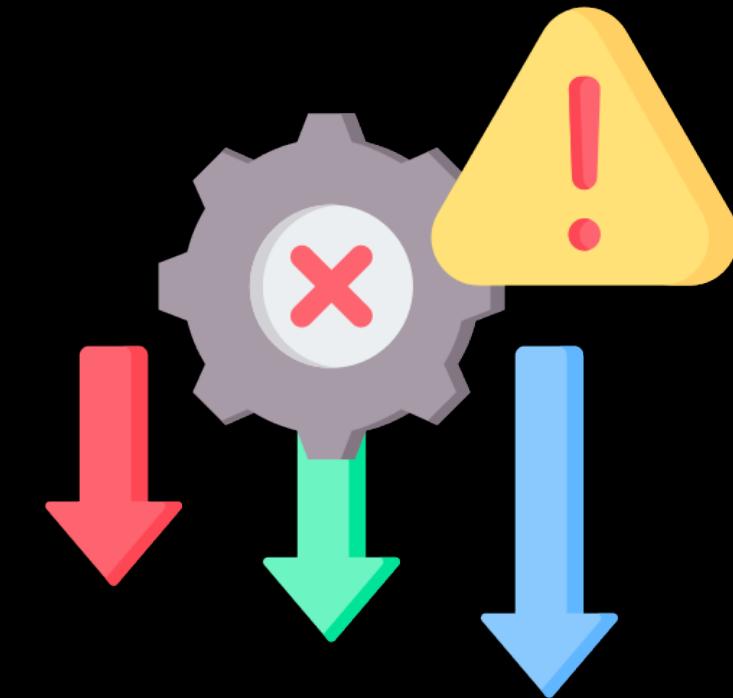
CI/CD의 핵심 목표



소프트웨어 품질 향상



개발 및 배포 속도 향상



오류 발생 가능성 감소



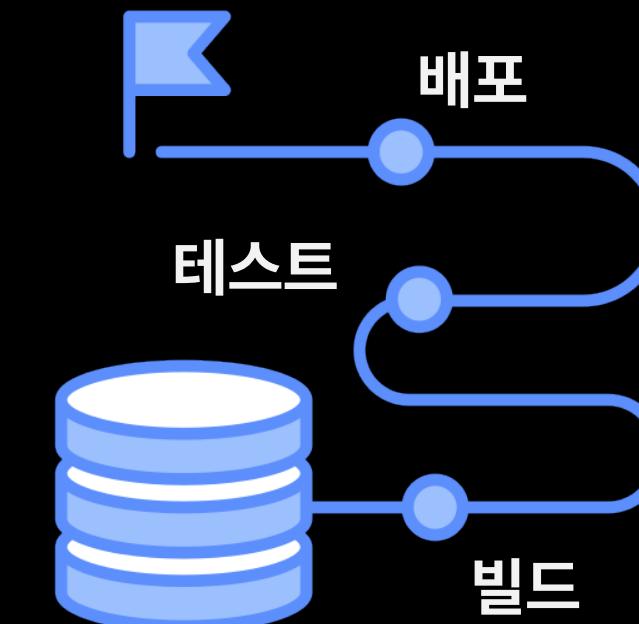
CI/CD 도입 시 유의 사항



-  자동화된 테스트 작성
-  지속적인 모니터링 도구 활용
-  팀원 간 CI/CD 파이프라인 이해 공유



CI/CD 파이프라인이란?



파이프라인

코드의 빌드(Build), 테스트(Test), 배포(Deploy) 과정을 자동화한 워크플로우

주요 역할

- 개발부터 배포까지의 과정을 자동화하여 일관성과 효율성 확보
- 반복 작업 제거로 생산성 향상

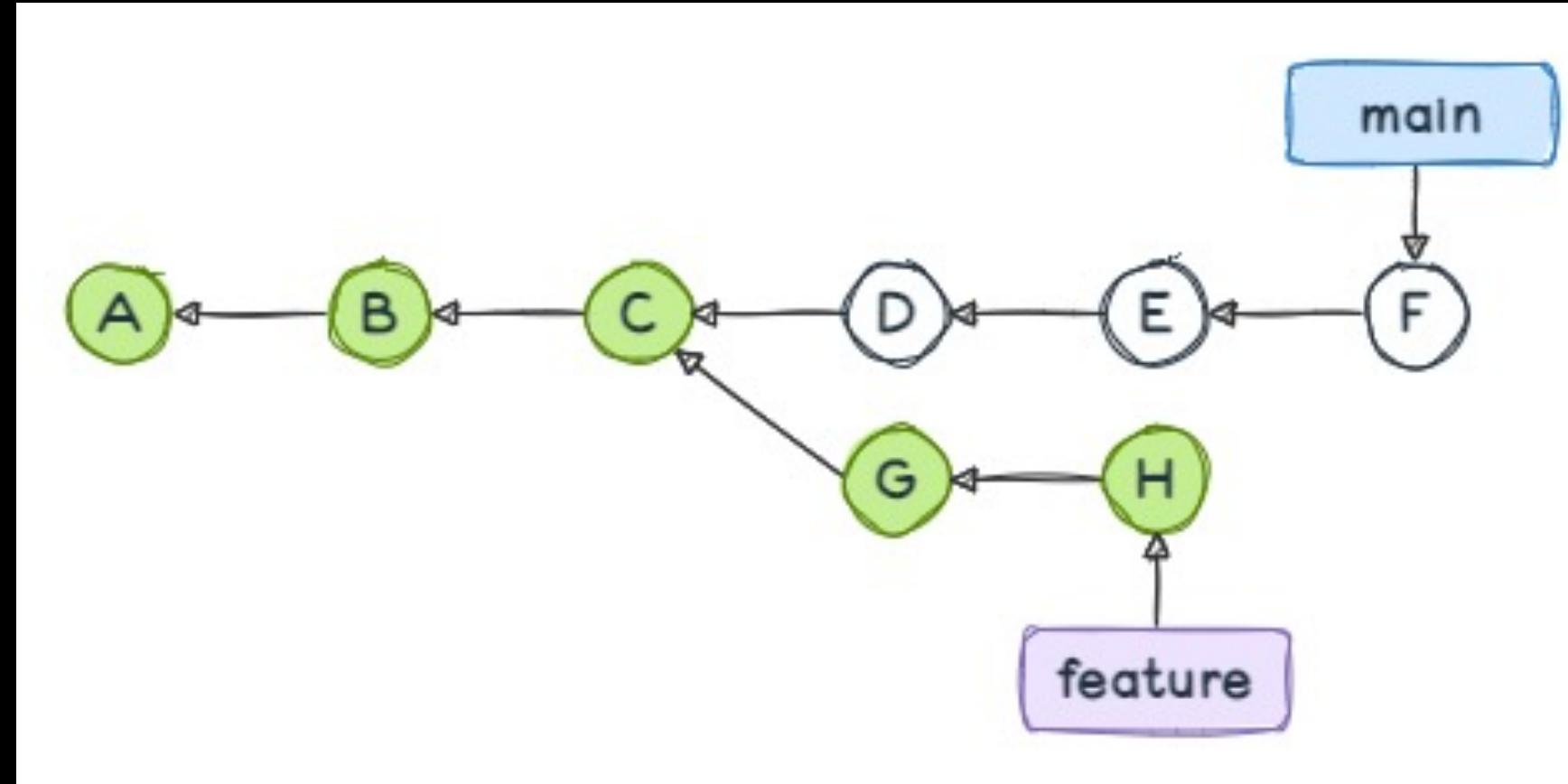


CI/CD 파이프라인의 기본 단계





코드 커밋이란?



Best



작은 단위로 자주 커밋



코드 리뷰 프로세스 도입

코드 커밋

개발자가 코드 저장소(Git, GitHub 등)에 변경 사항을 저장



빌드란 무엇인가?

빌드

소스 코드를 실행 가능한 상태로 컴파일
의존성 패키지 설치 및 코드 병합

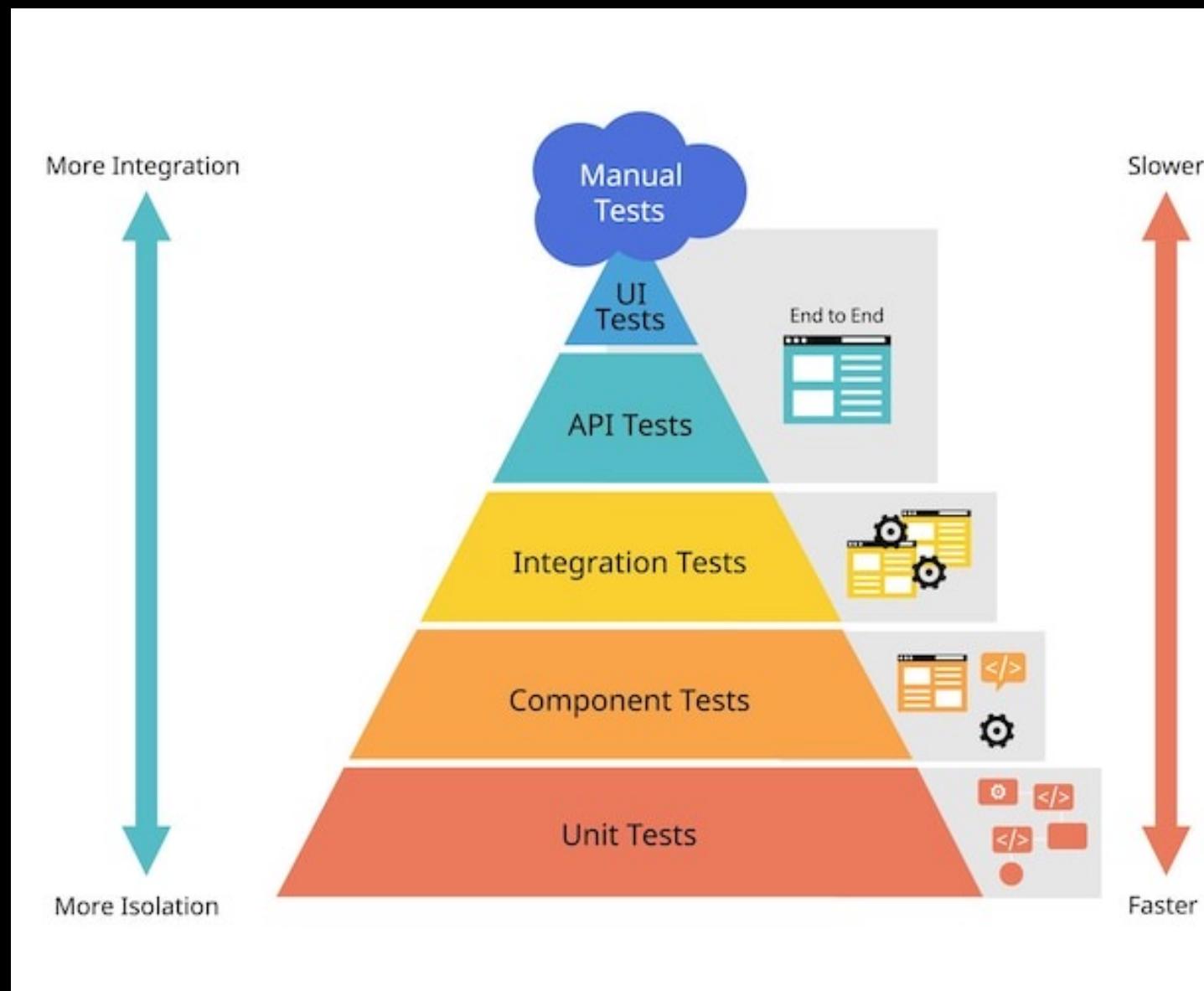
실행 가능한 코드를 생성하여
배포 준비 완료





테스트 단계에서 하는 일

코드 변경으로 인한 오류 및 품질 문제 조기 발견

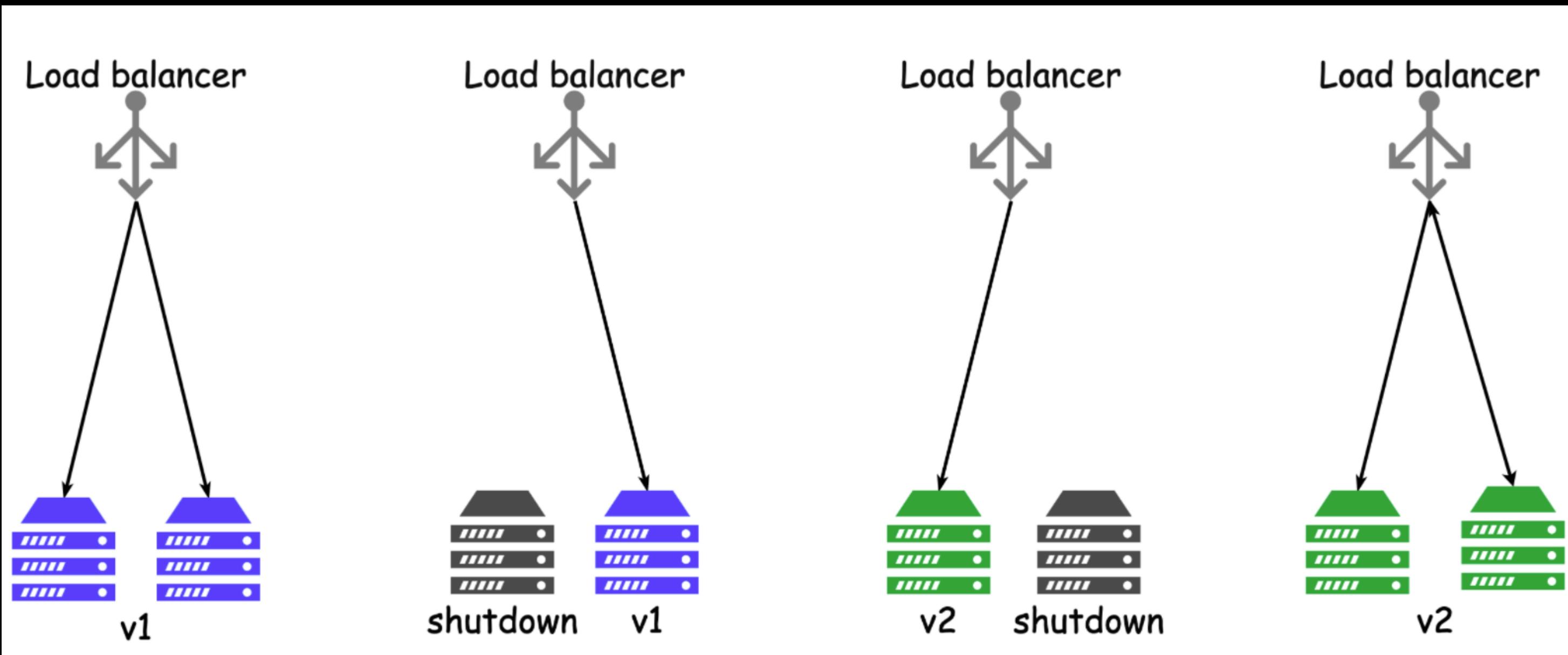


- **단위 테스트(Unit Test):** 개별 모듈 테스트
- **통합 테스트(Integration Test):** 모듈 간 상호작용 테스트
- **종단 간 테스트(End-to-End Test):** 전체 워크플로우 테스트



배포 단계에서 하는 일

프로덕션 환경에 코드를 자동으로 배포
프로덕션 배포 전 스테이징 환경에서 테스트 수행

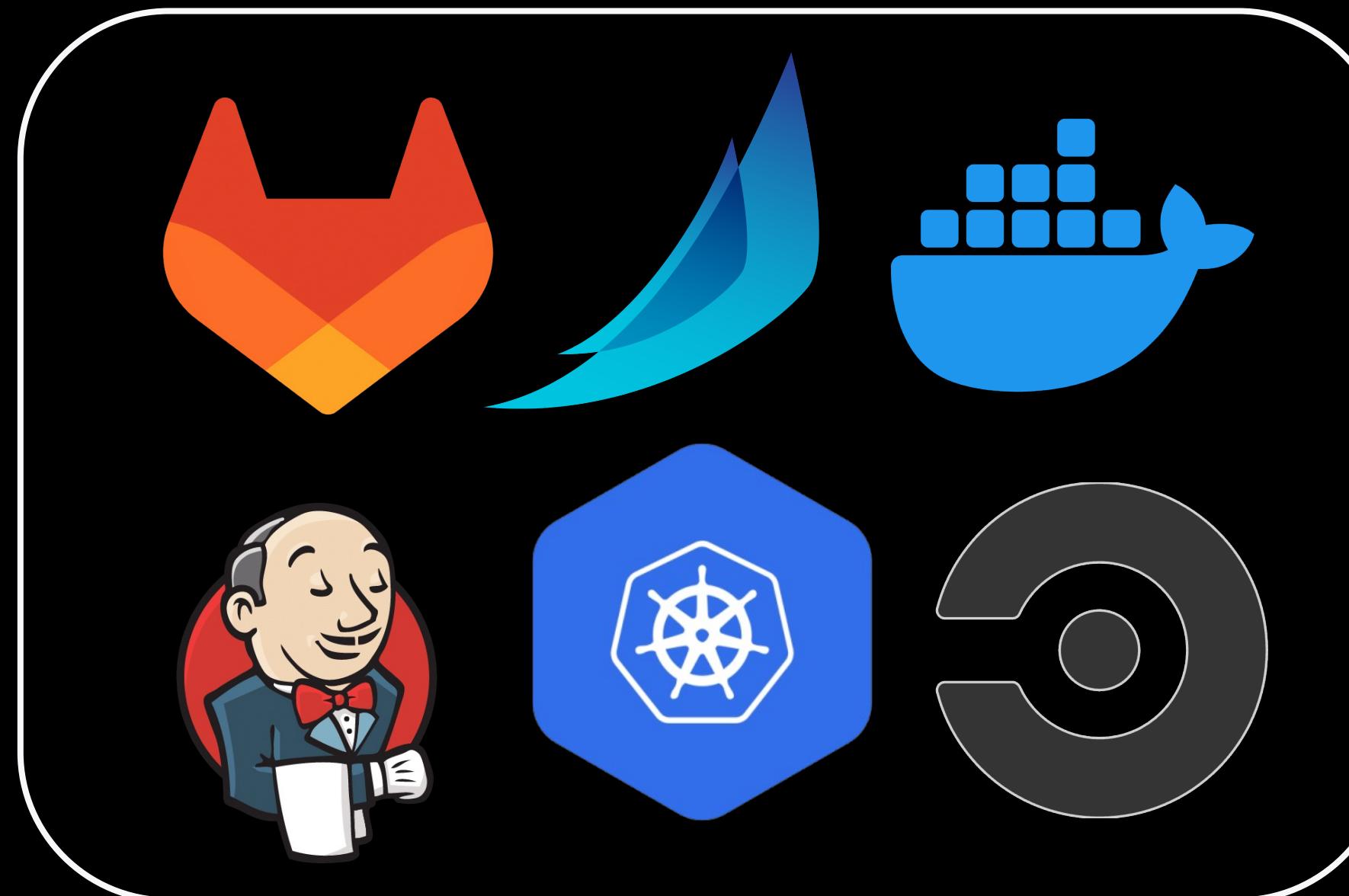


<https://velog.io/@jingrow/%EB%B8%94%EB%A3%A8%EA%B7%B8%EB%A6%B0-%EB%A1%A4%EB%A7%81-%EC%B9%B4%EB%82%98%EB%A6%AC-%EB%B0%B0%ED%8F%AC%EC%9D%98-%EC%B0%A8%EC%9D%B4%EC%A0%90%EA%B3%BC-%EC%96%B4%EB%96%A4-%EA%B2%BD%EC%9A%B0%EC%97%90-%EA%B0%81%EA%B0%81%EC%9D%98-%EB%B0%B0%ED%8F%AC%EB%B0%A9%EC%8B%9D%EC%9D%84-%EC%8B%9C%EB%8F%84%ED%95%98%EB%8A%94%EC%A7%80-%EC%A1%B0%EC%82%AC%ED%95%B4%EB%B3%B4%EC%84%B8%EC%9A%94>



CI/CD 도구란?

CI/CD 도구



빌드, 테스트, 배포를 자동화
파이프라인 상태 및 실행 결과 시각화

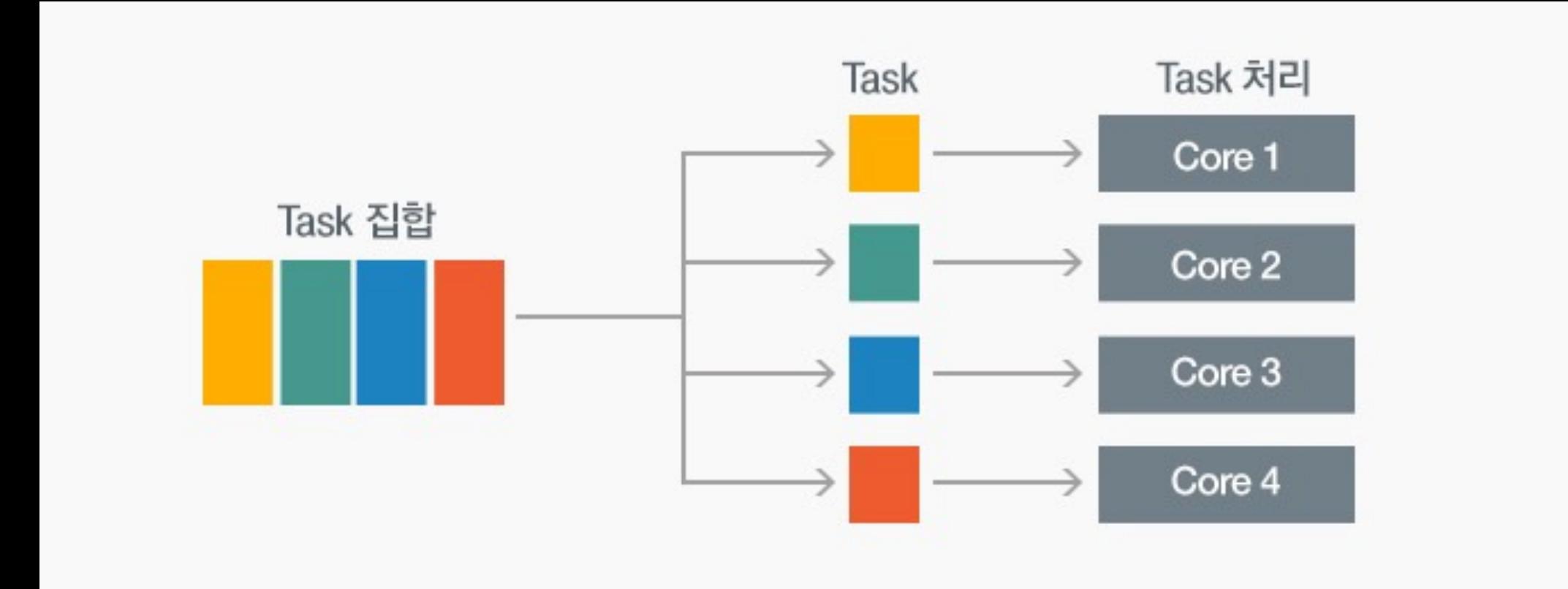
도구 선택 기준

- 팀 규모와 프로젝트 요구사항
- 호환성 및 확장성

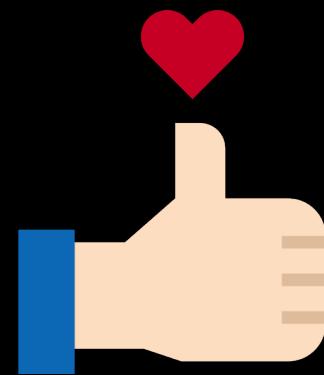


파이프라인의 병렬 처리

병렬 처리



여러 작업을 동시에 실행하여 빌드와 테스트 속도 개선



- 시간 절약
- 병목 현상 최소화



조건부 실행과 환경 변수

조건부 실행

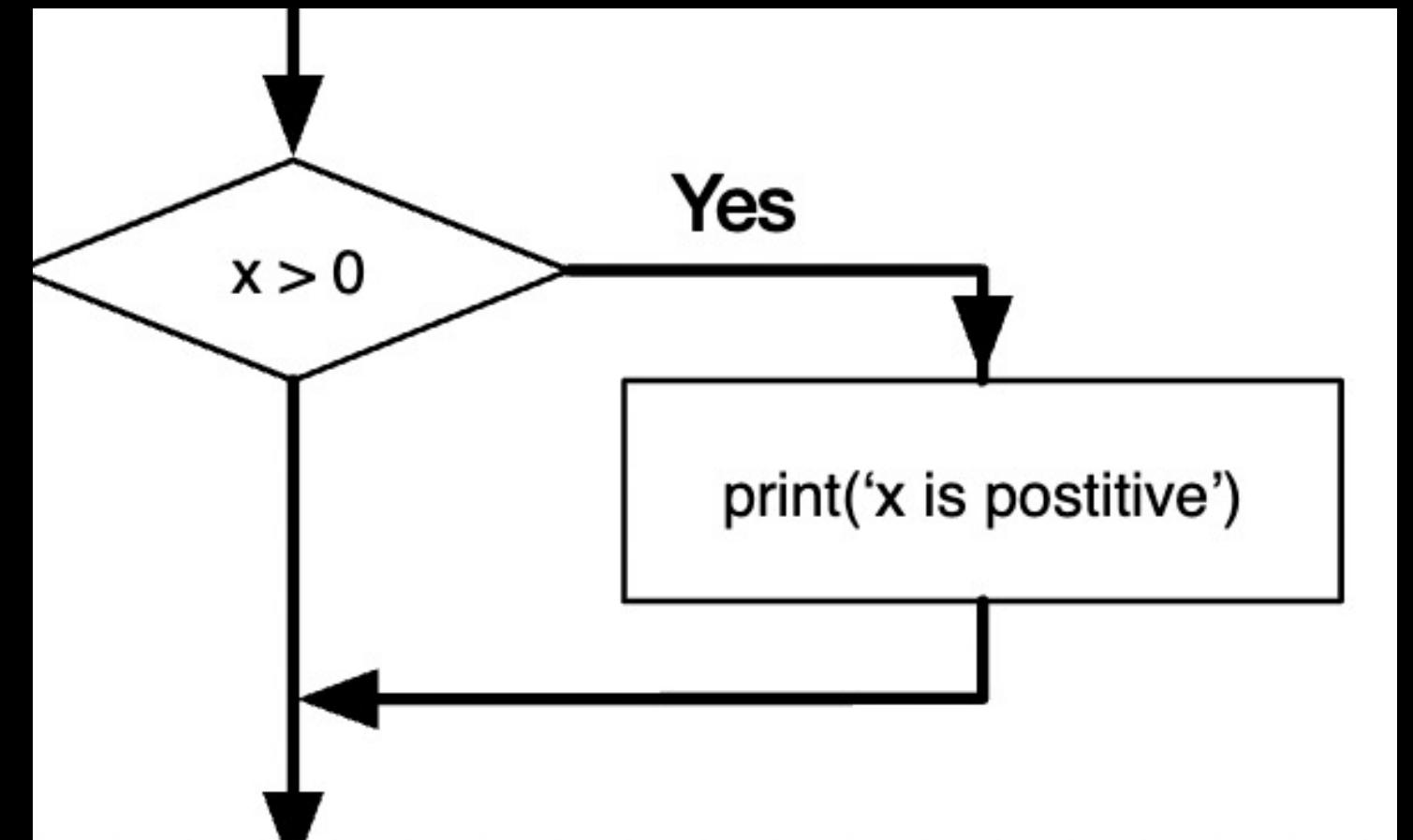
특정 조건에서만 단계 실행

예: 브랜치 이름이 main일 때만 배포 단계 실행

환경 변수(Environment Variables)

빌드 및 배포에 필요한 설정을 동적으로 전달

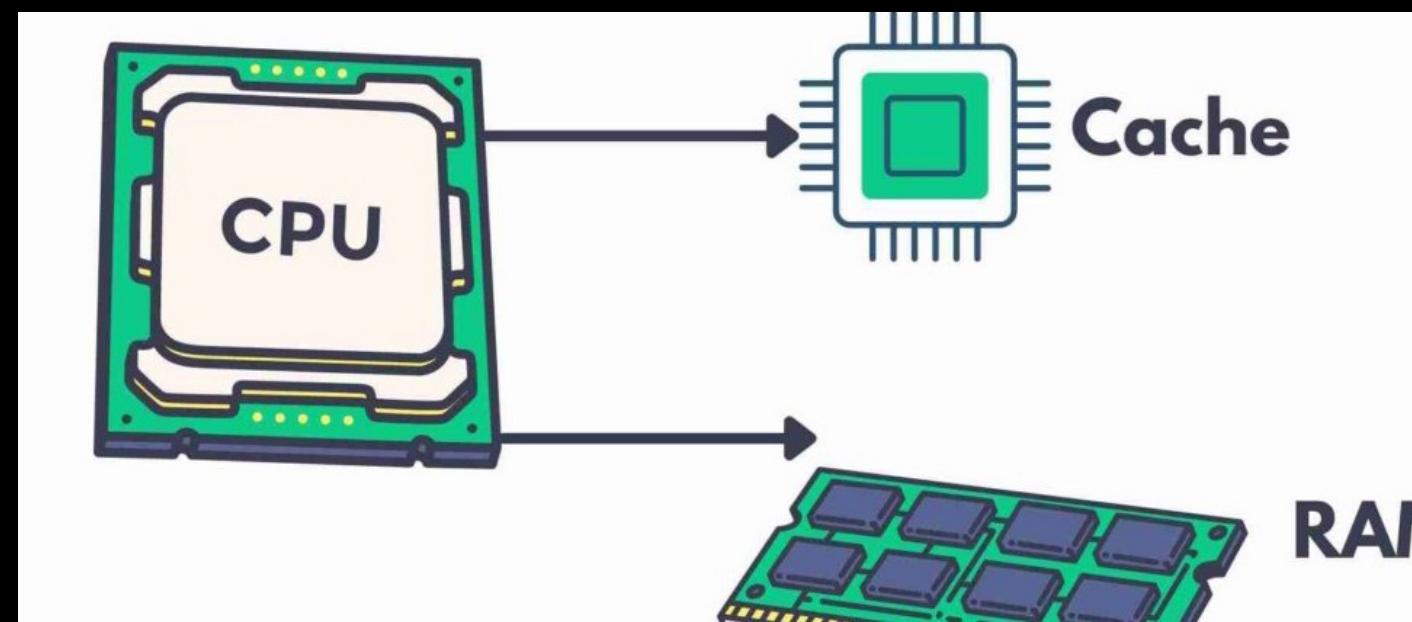
예: 데이터베이스 연결 정보, API 키





캐싱을 활용한 파이프라인 최적화

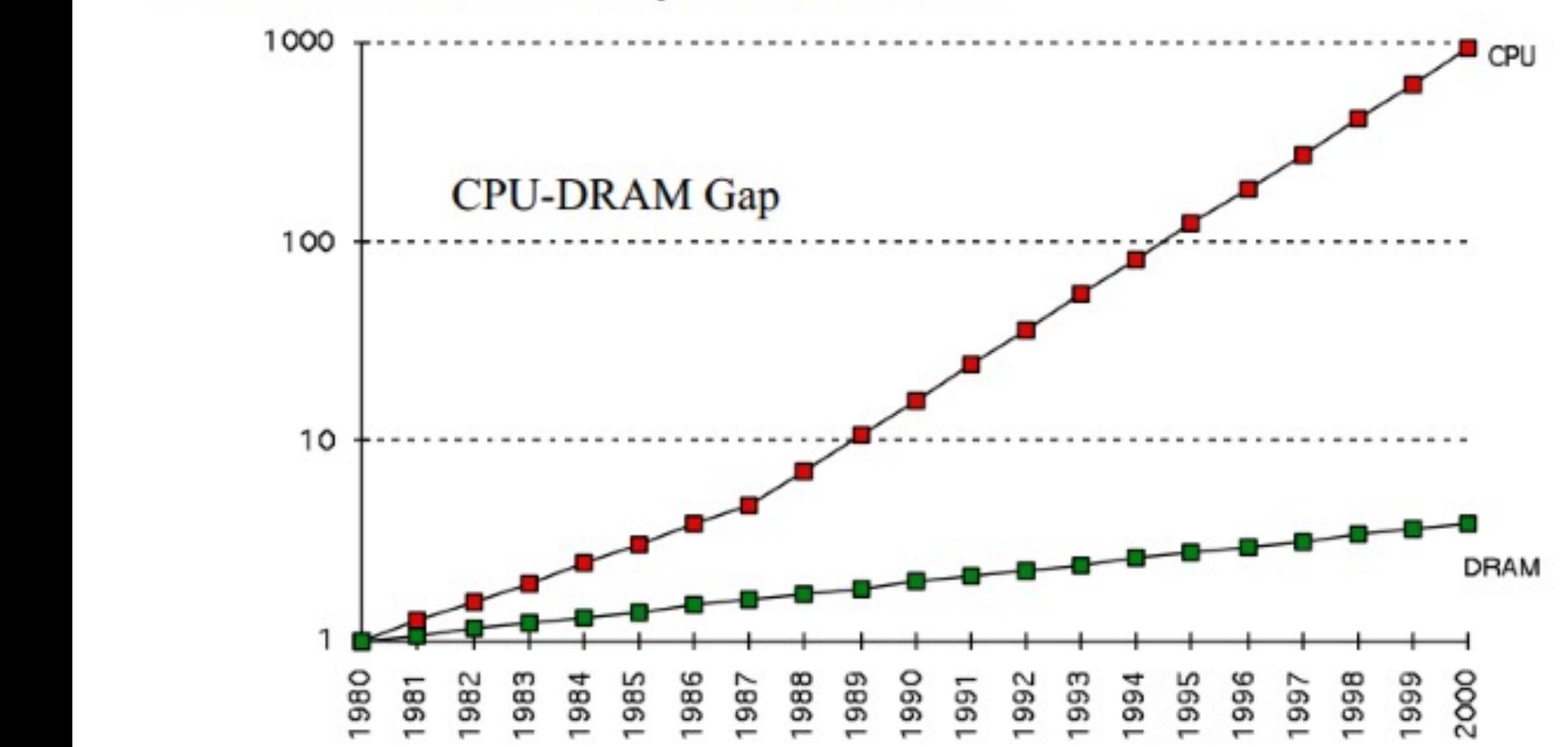
캐싱



- 동일한 작업에 대해 중복 실행 방지
- 빌드 속도 개선

유형 : 의존성 캐싱, 아티팩트 캐싱 등

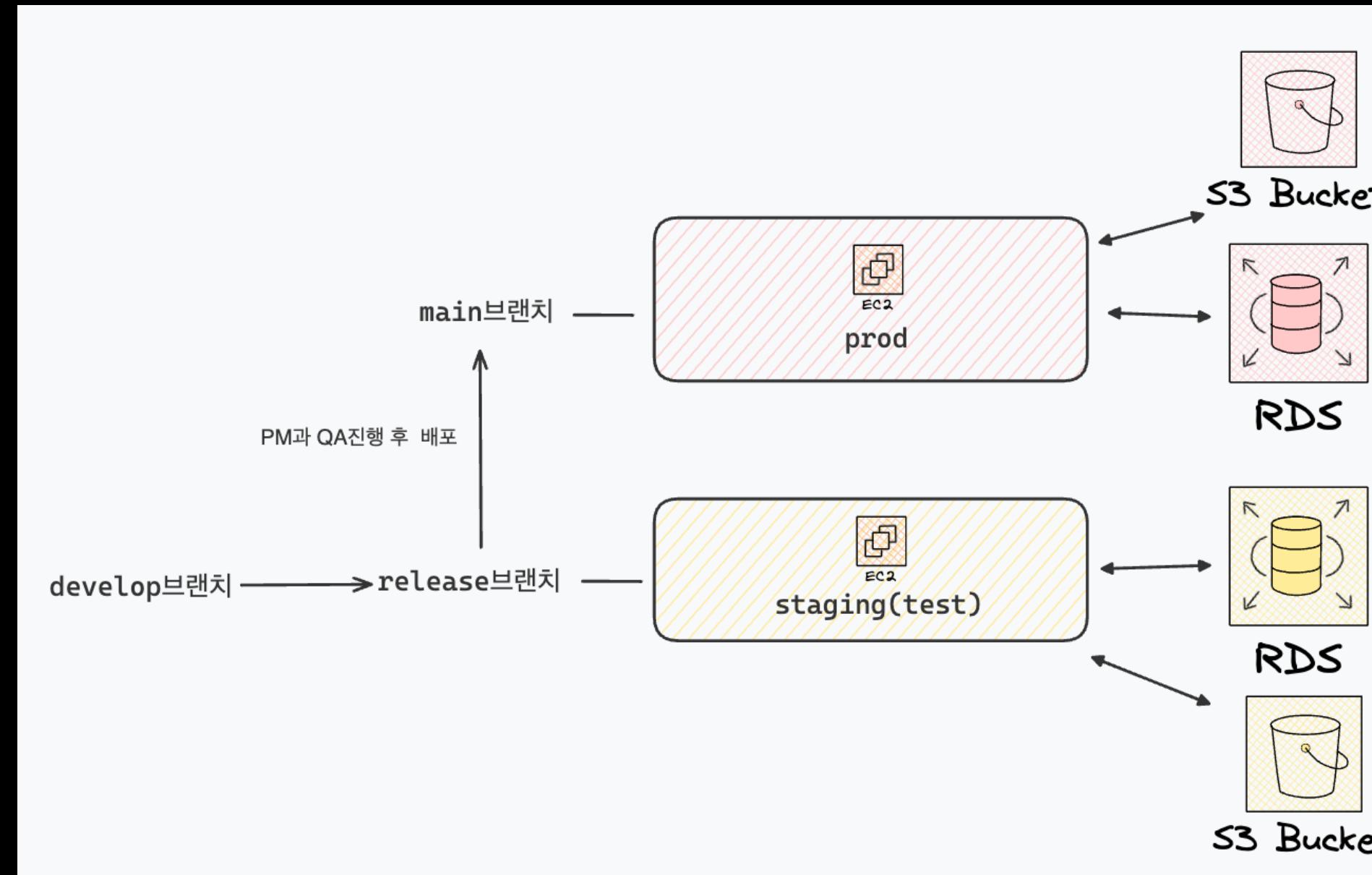
Processor vs Memory Performance



1980: no cache in microprocessor;
1995 2-level cache



스테이징 환경 테스트의 중요성



Best

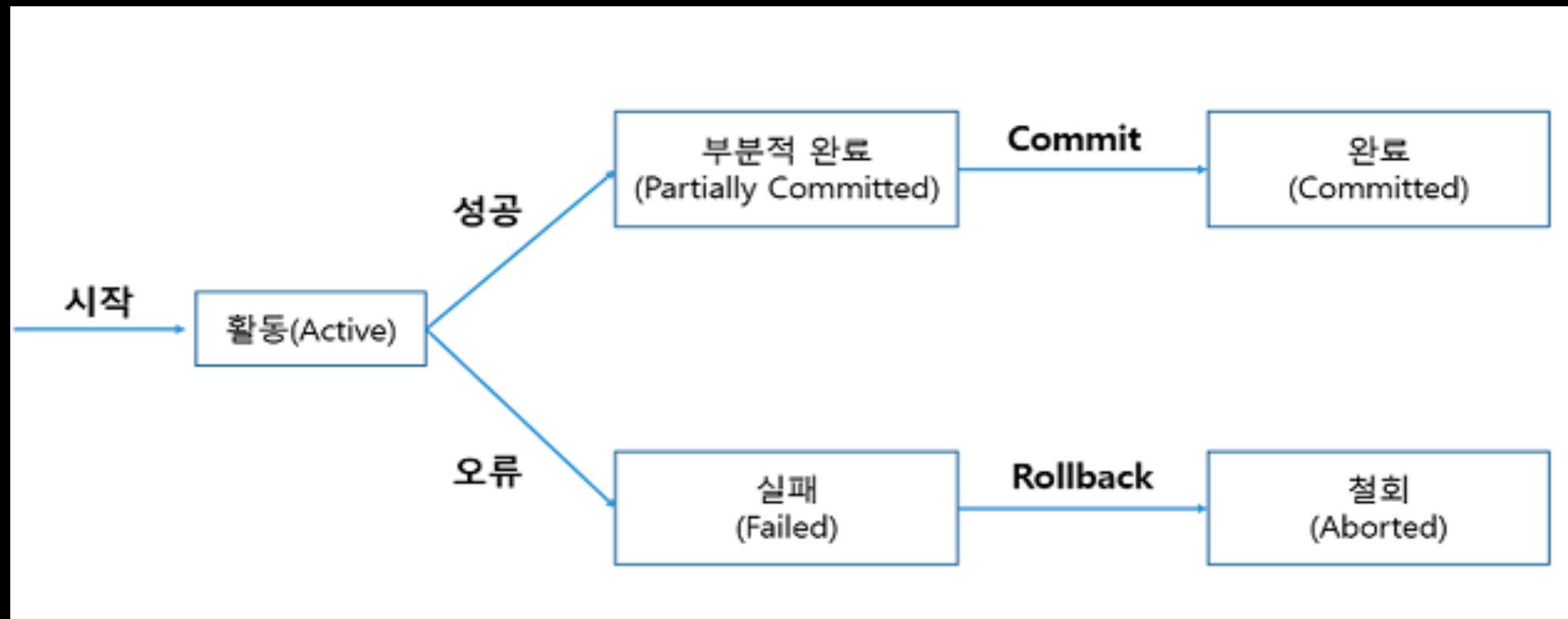
✓ 스테이징 환경을
프로덕션 환경과 최대한
동일하게 유지

스테이징 환경

- 실제 프로덕션 환경과 유사하게 설정된 테스트 환경
- 프로덕션 환경에서 발생할 수 있는 오류를 사전 검출



배포 실패 시 롤백 계획



롤백 계획

롤백

배포 후 문제가 발생했을 때 이전 버전으로 복구하는 작업

- 배포 이전 상태 스냅샷 저장
- 배포 실패 조건 정의



파이프라인 오류 처리 전략

```
2020-10-21 11:04:40.753 INFO 22020 --- [main] o.s.batch.core.step.AbstractStep : Step: [step1] executed in 58ms
2020-10-21 11:04:40.757 ERROR 22020 --- [main] c.e.b.JobCompletionNotificationListener : !!! JOB FINISHED! Time to verify the results
2020-10-21 11:04:40.759 INFO 22020 --- [main] c.e.b.JobCompletionNotificationListener : Found <firstName: JILL, lastName: DOE> in the database.
2020-10-21 11:04:40.759 INFO 22020 --- [main] c.e.b.JobCompletionNotificationListener : Found <firstName: JOE, lastName: DOE> in the database.
2020-10-21 11:04:40.759 INFO 22020 --- [main] c.e.b.JobCompletionNotificationListener : Found <firstName: JUSTIN, lastName: DOE> in the database.
2020-10-21 11:04:40.759 INFO 22020 --- [main] c.e.b.JobCompletionNotificationListener : Found <firstName: JANE, lastName: DOE> in the database.
2020-10-21 11:04:40.759 INFO 22020 --- [main] c.e.b.JobCompletionNotificationListener : Found <firstName: JOHN, lastName: DOE> in the database.
2020-10-21 11:04:40.761 INFO 22020 --- [main] o.s.b.c.l.support.SimpleJobLauncher : Job: [FlowJob: [name=importUserJob]] completed with the following parameters:
[COMPLETED] in 88ms
2020-10-21 11:04:40.765 INFO 22020 --- [main] com.zaxxer.hikari.HikariDataSource : HikariPool-1 - Shutdown initiated...
2020-10-21 11:04:40.767 INFO 22020 --- [main] com.zaxxer.hikari.HikariDataSource : HikariPool-1 - Shutdown completed.
Picked up JAVA_TOOL_OPTIONS: -Djava.net.preferIPv4Stack=true

C:\Windows\System32\config\systemprofile\.m2\repository\com\example\batch-processing\0.0.1-SNAPSHOT>exit 0
Build step 'Console output (build log) parsing' changed build result to FAILURE
[Checks API] No suitable checks publisher found.
Finished: FAILURE
```

오류 발생 원인

빌드 실패, 테스트 실패, 배포 환경 문제 등



오류 처리 방법

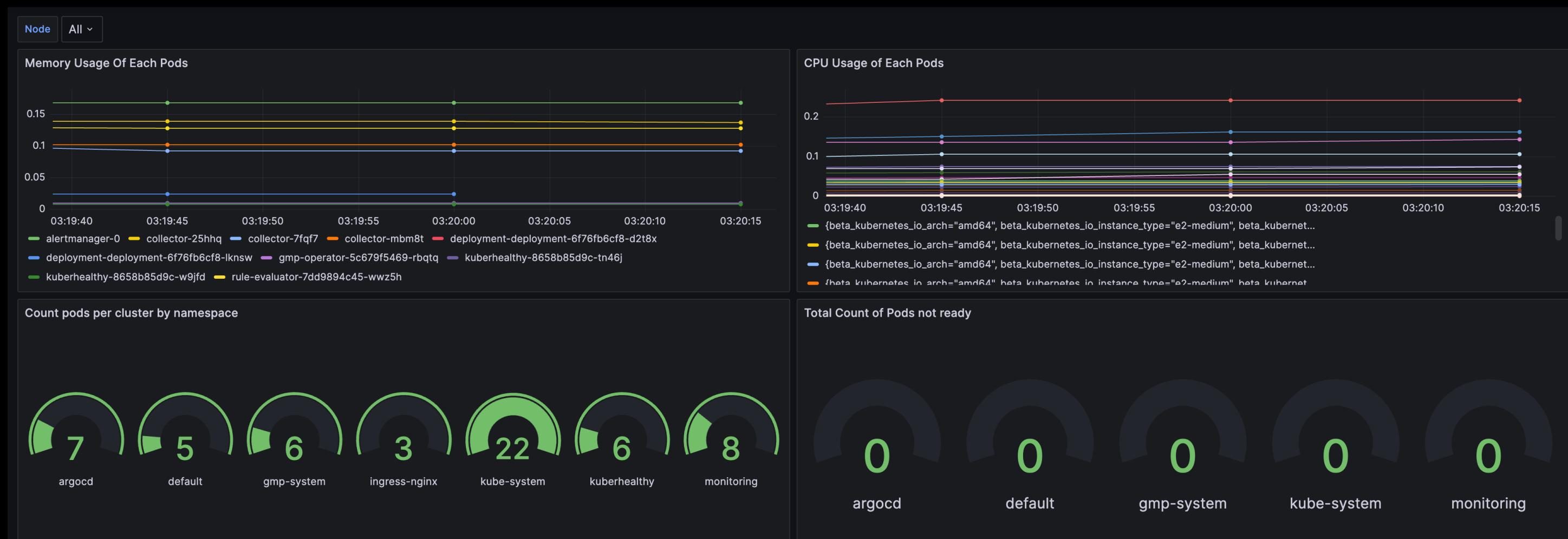
- 로그 분석을 통한 원인 파악
- 알림 설정(Slack, 이메일)
- 자동화된 재실행 설정



CI/CD 모니터링의 중요성

파이프라인의 성능 및 문제를 실시간으로 추적

도구: Prometheus, Grafana, New Relic



주요 모니터링 요소

- 빌드 시간
- 테스트 성공률
- 배포 성공률

<https://velog.io/@mdev97/K8S-CICD-Pipeline-%EB%AA%A8%EB%8B%88%ED%84%B0%EB%A7%81-%EB%A1%9C%EA%B9%85-%EC%8B%9C%EC%8A%A4%ED%85%9C-%EA%B5%AC%EC%B6%95-%ED%94%84%EB%A1%9C%EC%A0%9D%ED%8A%B8-%EB%AA%A8%EB%8B%88%ED%84%B0%EB%A7%81-%EC%8B%9C%EC%8A%A4%ED%85%9C-%EA%B5%AC%EC%B6%95>



파이프라인 최적화 전략



- 불필요한 단계 제거
- 병렬 처리 강화
- 캐싱 적극 활용
- 파이프라인 실행 트리거 조건 설정



퀴즈 타임



QUIZ 1번부터 7번을 풀어봐요!



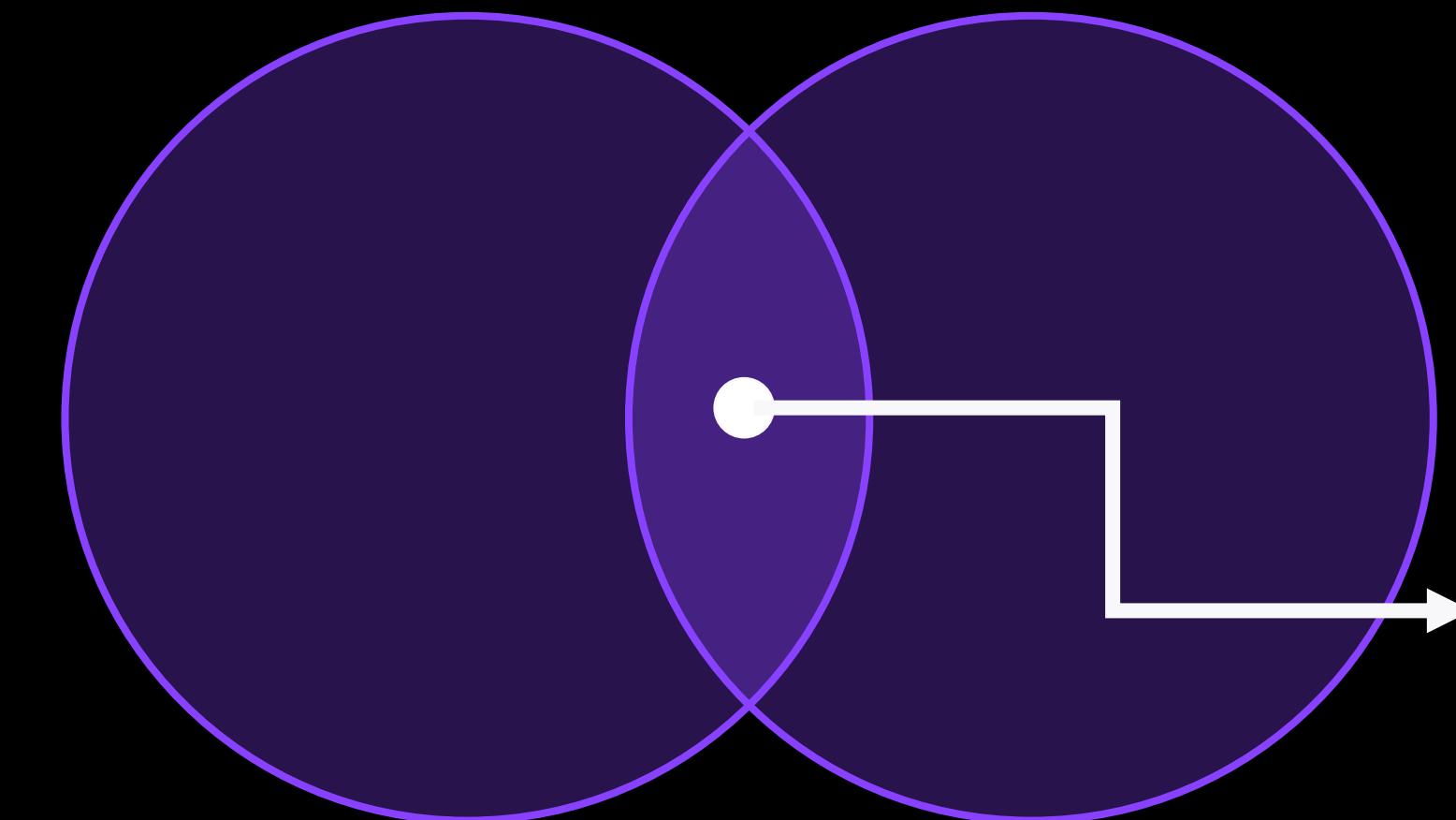
QA와 DevOps의 정의

QA(Quality Assurance)

- 소프트웨어 개발 전 과정에서 품질 기준을 보장
- 오류를 사전에 방지하는 예방 활동 중심

DevOps(Development + Operations)

- 개발과 운영 간의 협업을 통해 빠르고 안정적인 소프트웨어 배포
- CI/CD를 통한 자동화와 통합 프로세스 강조



안정성, 민첩성, 지속적인 개선 목표



QA와 DevOps의 역할 분담

QA



- 자동화 테스트 작성 및 유지
- 품질 기준 설정 및 모니터링
- 테스트 결과 분석 및 리포트

품질 중심

DevOps



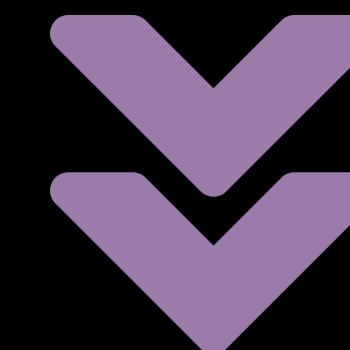
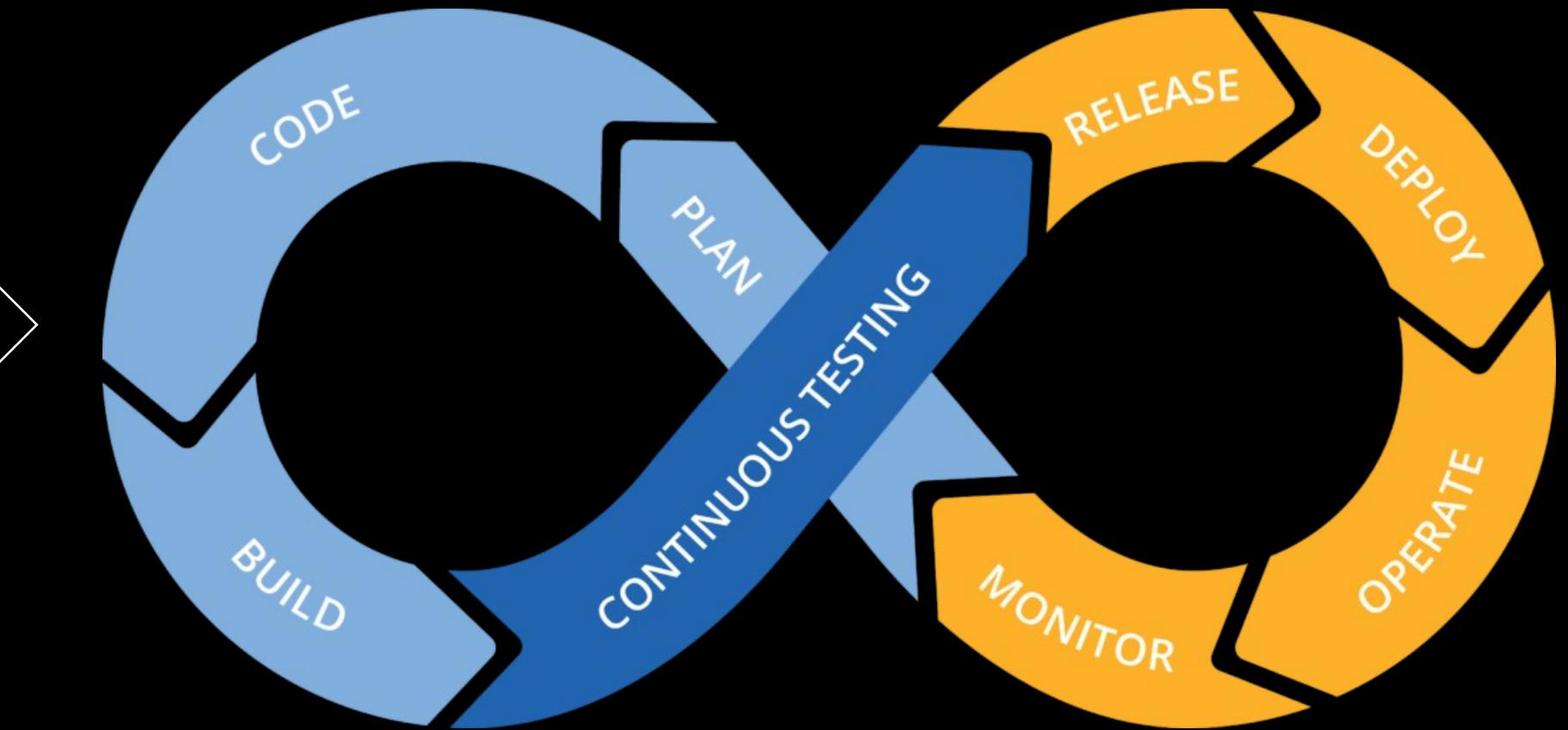
- CI/CD 파이프라인 설계 및 유지
- 배포 자동화 및 환경 설정
- 운영 환경의 안정성 관리

속도 중심

QA와 DevOps 파이프라인의 관계

QA의 DevOps 파이프라인 참여

- 자동화된 테스트 도입으로 품질 보증 강화
- 릴리스 단계에서 결함 발생률 감소



릴리스 속도 향상

품질 문제 조기 발견

**Selenium**

웹 애플리케이션 테스트

**JUnit5**

단위 테스트

**Postman**

API 테스트

DevOps 파이프라인에서 QA 도구의 통합

- Jenkins, GitLab CI/CD와 QA 도구의 연계
- 빌드 단계 이후 테스트 자동 실행



QA의 역할 변화

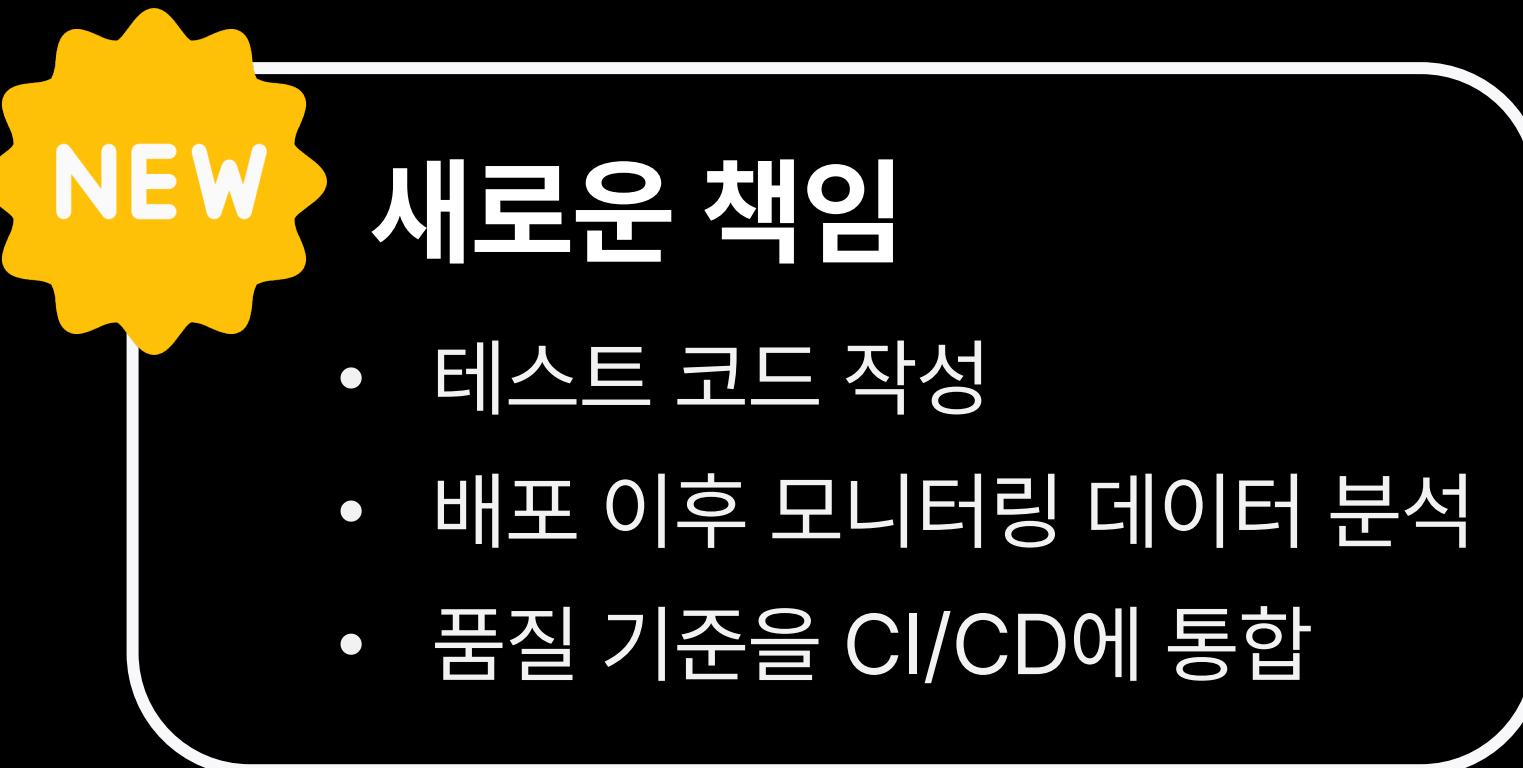
기존 QA

수동 테스트 중심



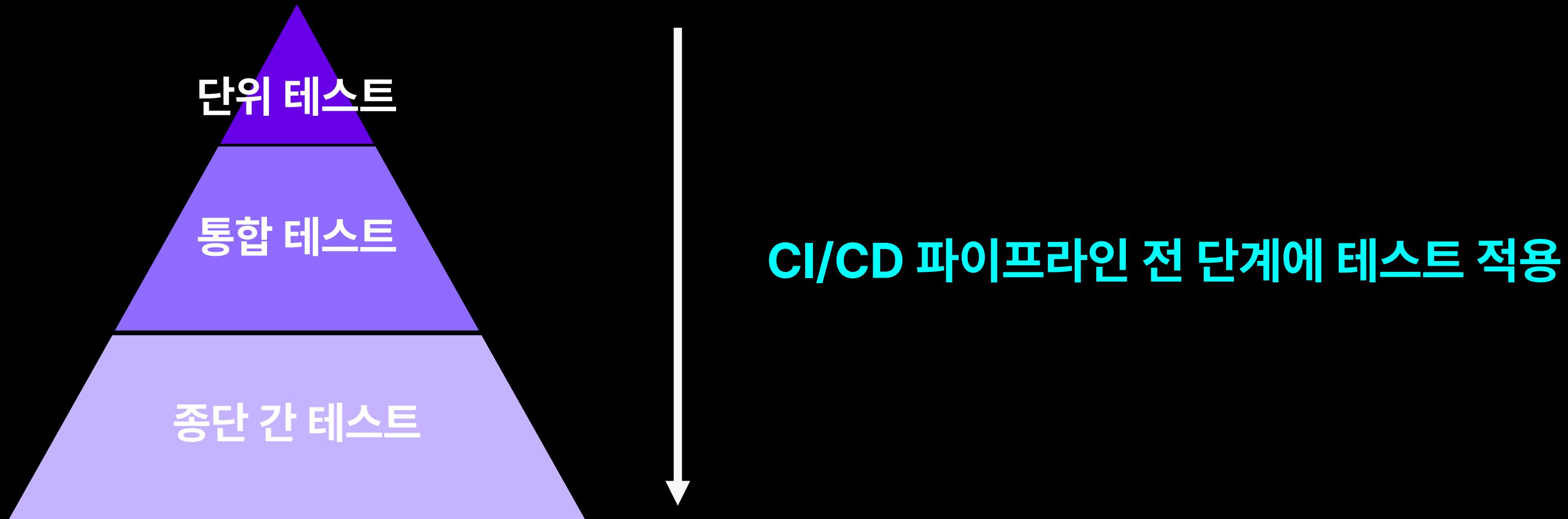
DevOps의 QA

자동화와 지속적인 검증 중심





QA의 테스트 확대



배포 후 시스템 로그와 사용자 피드백 분석

서비스 가용성과 성능 검증



QA와 DevOps 협업의 도전 과제



속도와 품질 간 충돌

DevOps의 속도 중심 문화와 QA의 품질 기준 간의 충돌

자동화 부족

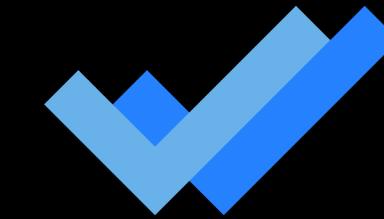
QA 자동화의 미흡으로 인한 배포 지연

팀 간 의사소통 부족

QA와 DevOps 팀의 목표 불일치



QA와 DevOps 협업의 해결 방안



속도와 품질의 균형 유지

QA의 자동화 테스트 도입으로 배포 속도 향상
CI/CD 파이프라인 내 품질 기준 설정



팀 간 협업 강화

QA와 DevOps 팀 간 정기적인 회의와 목표 공유
DevOps 주기 초반부터 QA를 참여시켜 품질 문제 예방



도구 활용

Jira와 Confluence로 QA와 DevOps 팀 간의 소통 강화
CI/CD 도구에서 QA의 테스트 리포트 공유



협업을 성공적으로 이끄는 전략



QA의 초기 단계 참여

DevOps의 계획 단계부터 QA를 포함



지속적인 피드백 루프

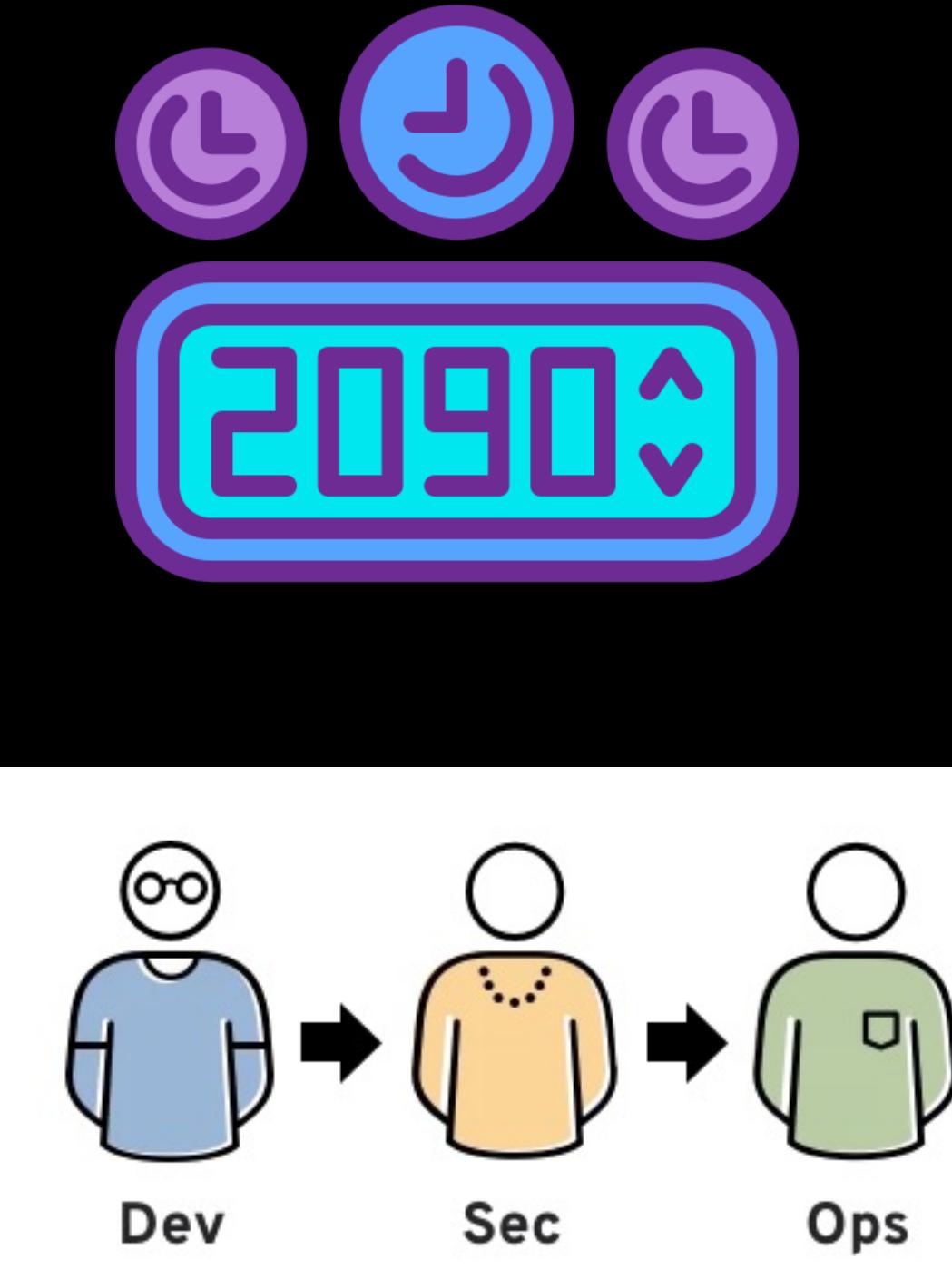
배포 후 QA가 로그와 데이터를 분석하여 개선점 제공



품질 우선 문화 조성

QA와 DevOps 팀이 공통된 목표를 공유





- AI 및 머신러닝 도구를 활용한 테스트 자동화 강화
- QA와 DevOps의 역할 융합으로 팀 간 경계 감소
- DevSecOps 확산으로 보안 중심 QA 역할 강화



DevSecOps(Development + Security + Operations)

개발, 보안, 운영을 나타내는 용어이며 전체 IT 라이프사이클에 걸쳐 보안을 공동의 가치로 통합하는 문화, 자동화 및 플랫폼 설계에 대한 접근 방식



도구는 어떤 일을 할까요?



Jenkins: 오픈소스 CI/CD 도구의 선두주자



GitLab CI/CD: Git 저장소와 통합된 강력한 기능 제공



CircleCI: 클라우드 기반의 간단하고 빠른 설정

CI/CD 도구

빌드, 테스트, 배포를 자동화하는 도구

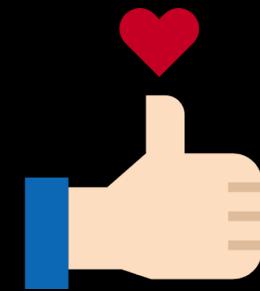
- 수작업 반복 작업 제거
- 팀 협업을 개선하고 배포 시간을 단축



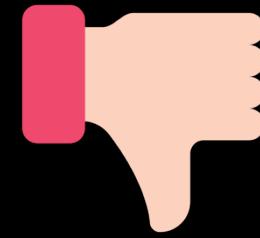
Jenkins, GitLab, CircleCI



Jenkins



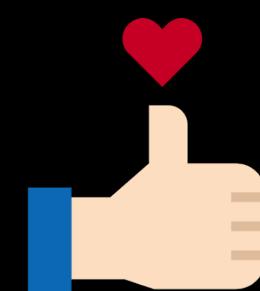
플러그인 지원으로
모든 환경에 적합



설정이 복잡하고
초기 학습 곡선이
높음



GitLab CI/CD



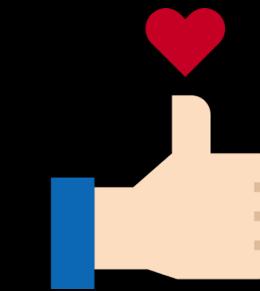
Git 저장소와
자연스럽게 통합



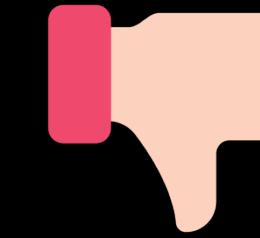
복잡한 워크플로우
구현에 제약



CircleCI



빠르고 간단한 설정,
클라우드 기반



특정 기능에서 비용
발생



우리 팀에 맞는 도구 찾기

도구 선택 기준



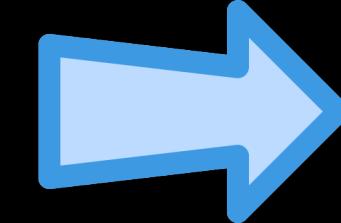
프로젝트 규모

작은 팀 → CircleCI / 큰 팀 → Jenkins



예산

오픈소스 도구는 무료(Jenkins)



“팀의 요구사항과 기술 스택에 따라
도구를 신중히 선택”

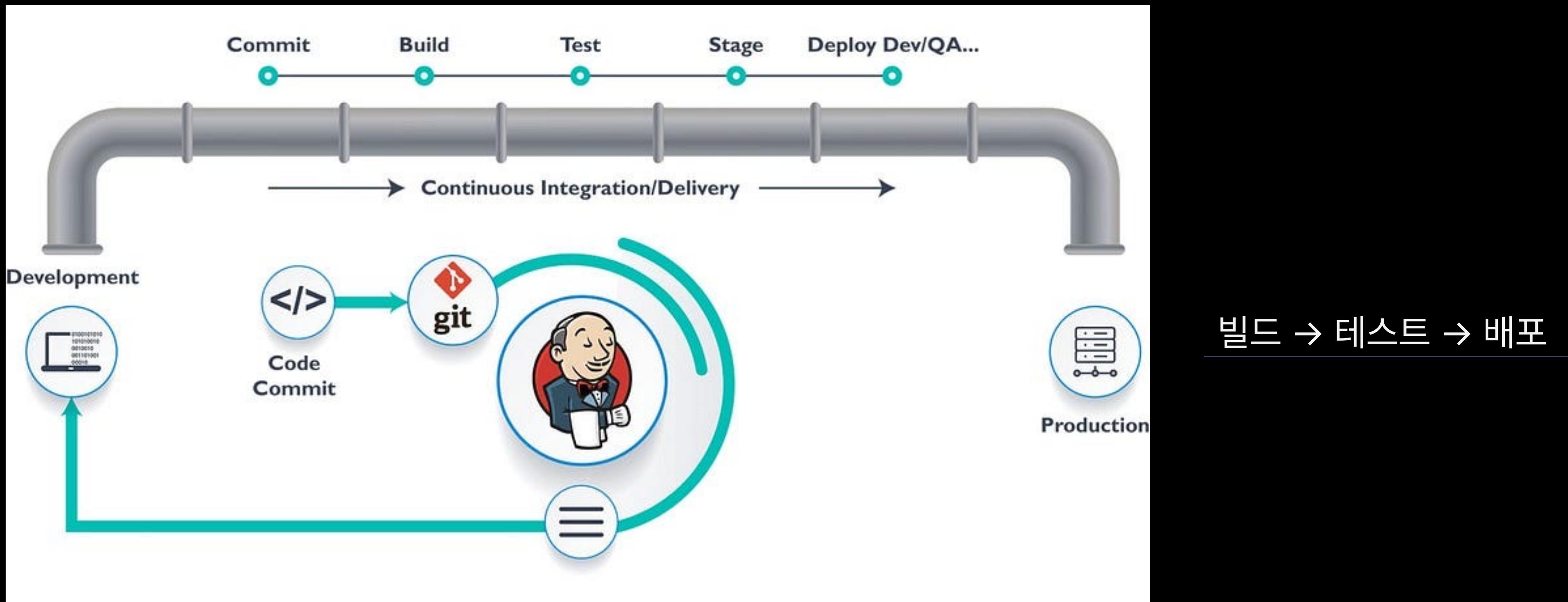


환경

클라우드 기반 → CircleCI / 온프레미스 → Jenkins

**온프레미스 : IT 인프라(서버, 네트워크, 스토리지 등)를 자체적으로 구축하여 운영하는 방식

Jenkins와 함께 파이프라인 만들기



Jenkins

가장 널리 사용되는 오픈소스 CI/CD 도구

무한한 커스터마이징 가능

Jenkins 파이프라인의 핵심 이해하기



```
1 pipeline {  
2     agent {  
3         'the agent'  
4     }  
5  
6     environment {  
7         MY_VAR = 'this value'  
8         OTHER_VAR = "${INITIAL_VAR ?: 'default value'}"  
9     }  
10  
11    stages {  
12        stage('does the thing') {  
13            }  
14        }  
15    }  
16 }
```

Jenkinsfile

파이프라인을 정의하는 코드 기반 파일

스테이지 구성

빌드(Build), 테스트(Test),
배포(Deploy) 단계로 분리



```
stages:
  - build_project
  - detect
  - clean
  - deploy

buid:project:
  stage: build_project
  except:
    - schedules
  script:
    - echo $CI_PROJECT_DIR
    - xcodebuild build -project $CI_PROJECT_DIR/$PROJECT_INIT -scheme $BUNDLE_APPLICATION
  tags:
    - ios_automation

jobDetect:$UIDID_IPHONE_7_BLACK:
  stage: detect
  only:
    - schedules
  script:
    - cd $CI_PROJECT_DIR || ls
    - chmod u+x deviceiOSConnected.sh
    - ./deviceiOSConnected.sh $UIDID_IPHONE_7_BLACK
  tags:
    - ios_automation

jobCleanOldApp:$UIDID_IPHONE_7_BLACK:
  stage: clean
  script:
    - ios-deploy -i $UIDID_IPHONE_7_BLACK -9 --bundle_id $BUNDLE_TEST_APPLICATION
  needs: ["jobDetect:$UIDID_IPHONE_7_BLACK"]
  tags:
    - ios_automation
```

GitLab CI/CD

GitLab 플랫폼 내에서 제공되는 통합 CI/CD 도구



설정 방법

.gitlab-ci.yml 파일 생성.

단계 정의(예: stages: - build - test - deploy)

- 저장소와 밀접하게 통합
- 브랜치 기반 워크플로우 지원

GitLab CI/CD 파이프라인 구성하기

파이프라인 단계 설정

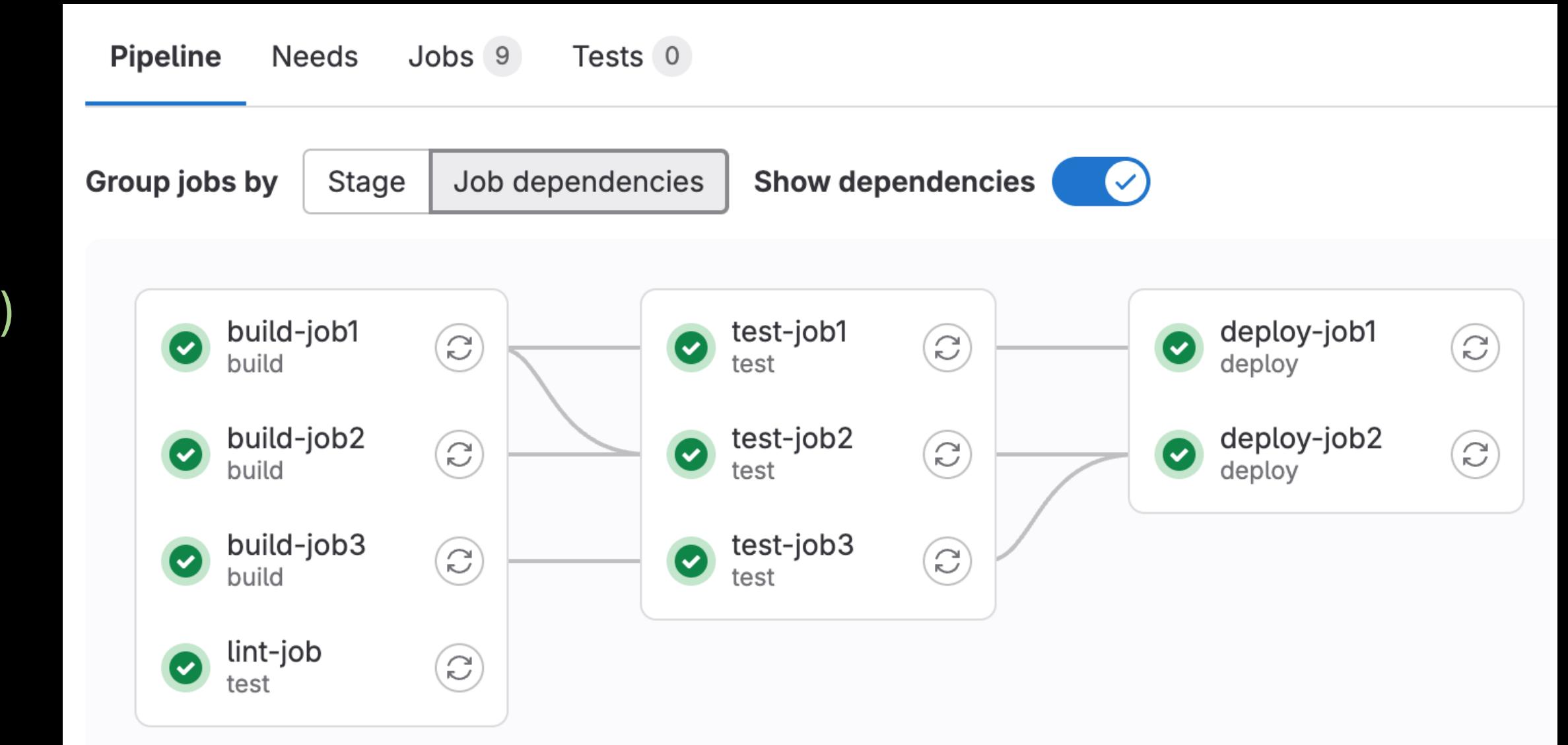
stages로 단계 나누기.

예: 빌드(Build) → 테스트(Test) → 배포(Deploy)

조건부 실행

브랜치 조건에 따라 특정 단계 실행

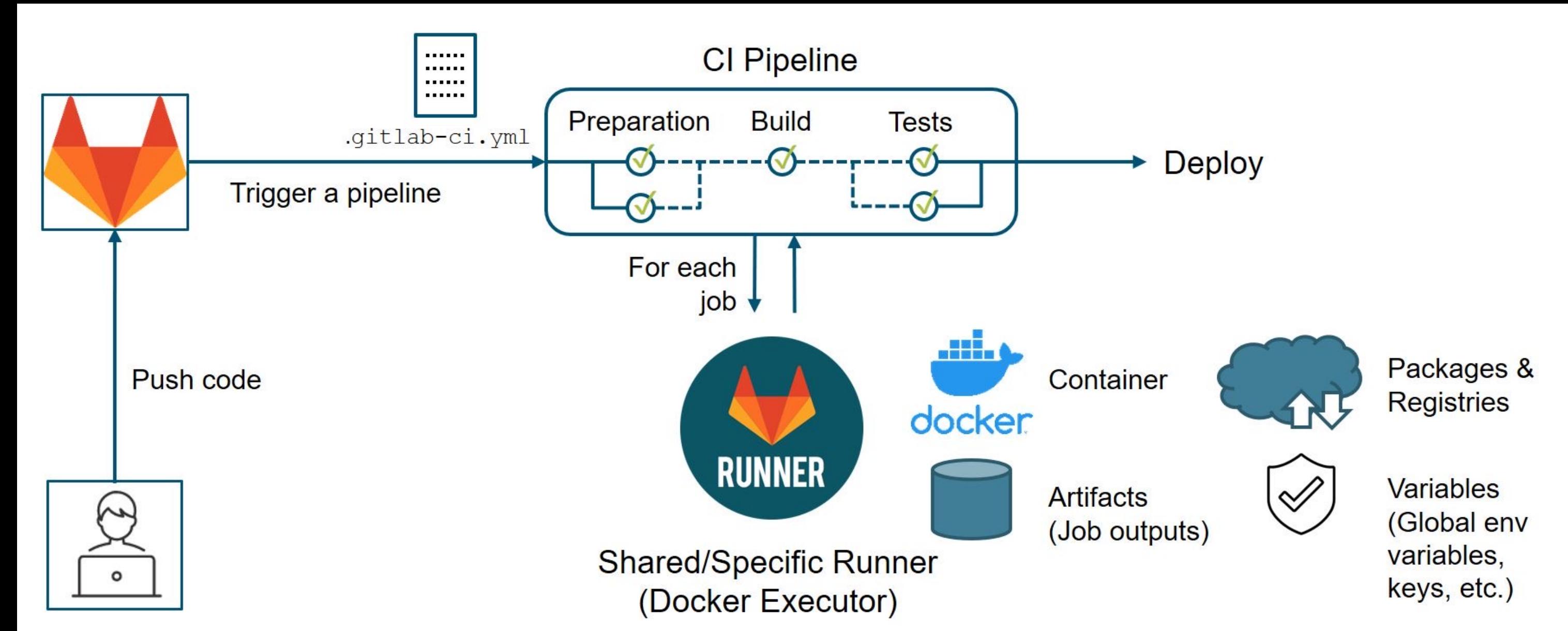
예: only: - main



실시간 모니터링

GitLab CI/CD 대시보드를 통해 파이프라인 상태 확인

GitLab CI/CD의 강력한 기능 활용하기



- **캐싱(Cache)** : 빌드 속도 최적화(cache 설정 사용)
- **병렬 처리** : 여러 작업을 동시에 실행
- **아티팩트(Artifacts)** : 빌드 결과물을 저장 및 공유



CircleCI와 함께하는 클라우드 기반 CI/CD

```
.circleci > ! config.yml
11    machine: true
12    steps:
13        - checkout
14        # only run this if we are in a Pull Request
15        # get-pr-info fails otherwise...
16        - run:
17            name: 'Check for PR'
18            command: |
19                if [ "$CIRCLE_PULL_REQUEST" = "" ]; then
20                    circleci-agent step halt
21                fi
22        - ghpr/get-pr-info
23
24    workflows:
25        setup:
26            jobs:
27                - setup:
28                    context: Deployment
29                - continuation/continue:
30                    configuration_path: ".circleci/test.yml"
31                    parameters: |
32                        { "targetBranch": "$GITHUB_PR_BASE_BRANCH", "buildNumber": "$CIRCLE_BUILD_NUM" }
33                    requires:
34                        - setup      You, 5 days ago • Always continue, even if NOT in PR
```

CircleCI

클라우드 기반 CI/CD 도구로 빠르고 간단한 설정이 강점



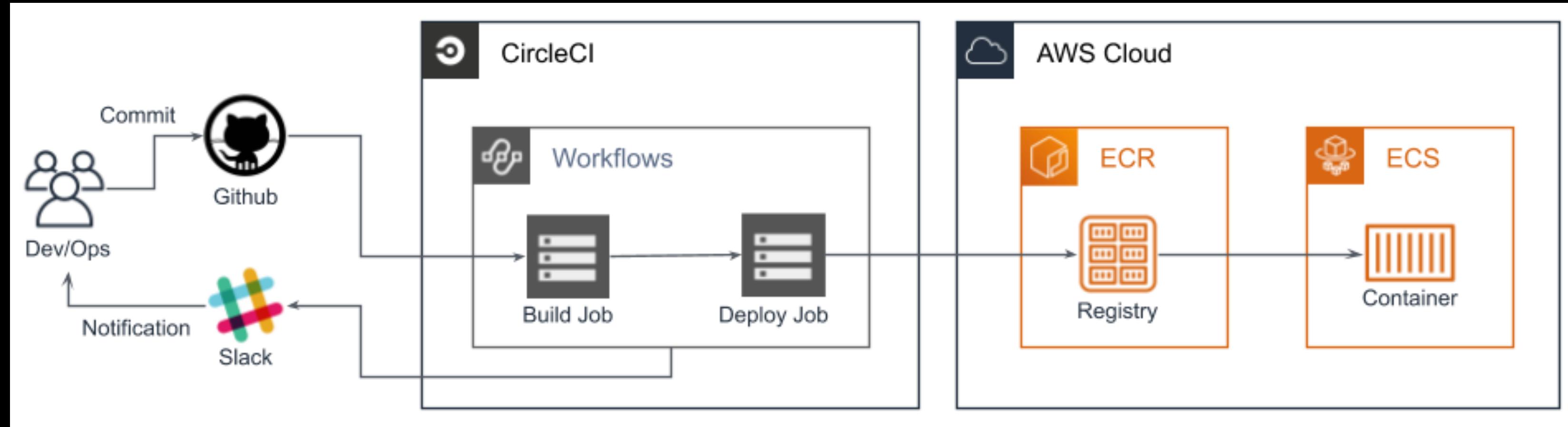
설정 방법

.circleci/config.yml 파일 생성
워크플로우와 작업(Job)을 정의

- 클라우드 환경에서 무설치로 사용 가능
- 자동화된 병렬 실행 지원



CircleCI로 파이프라인 구축하기



워크플로우 정의

workflows를 사용해 빌드 → 테스트 → 배포 흐름 구성

작업(Job) 추가

각 단계의 세부 작업 정의

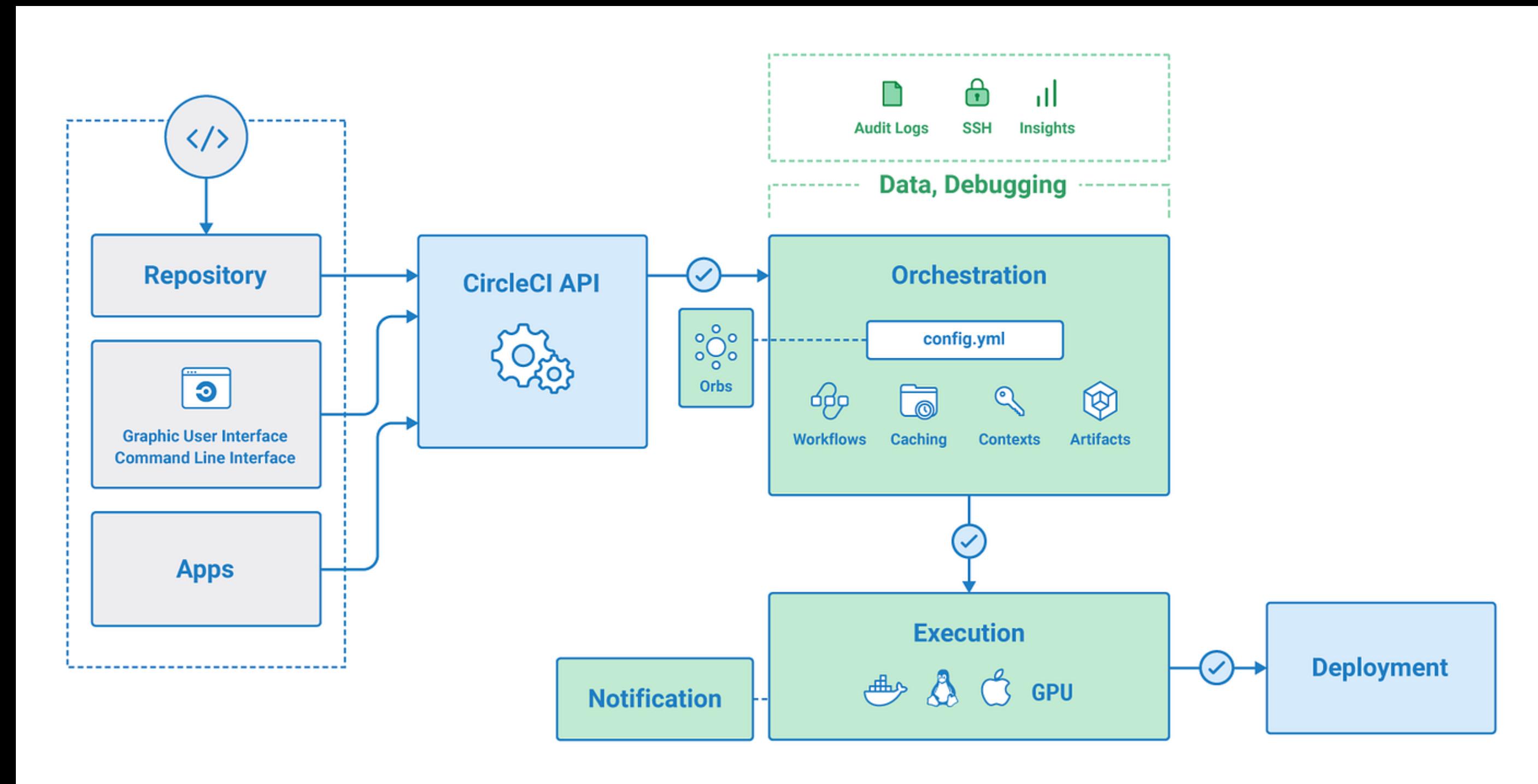
예: build: steps: - run: npm install

병렬 실행

테스트를 병렬로 실행해 속도 향상



CircleCI의 고급 설정 활용



- **환경 변수** : 민감 정보를 안전하게 저장 및 사용
- **캐싱(Cache)** : 빌드 속도 최적화
- **컨테이너화** : Docker 이미지를 활용한 환경 구축



우리 팀에 맞는 도구 찾기

프로젝트 규모가 큰가?



오픈소스 도구를 선호하는가?

클라우드 기반 환경을 요구하는가?

고급 워크플로우가 필요한가?



도구별 비교: 장점과 단점

도구	장점	단점
Jenkins	강력한 플러그인, 자유로운 커스터마이징	설정이 복잡함
GitLab CI/CD	Git과 자연스러운 통합, 무료 사용 가능	복잡한 워크플로우에서 제한적
CircleCI	빠르고 간단한 클라우드 환경	특정 기능에서 비용 발생

도구는 목적을 이루기 위한 수단일 뿐



“도구는 프로세스의 일부일 뿐”

도구 < 팀의 협업과 전략

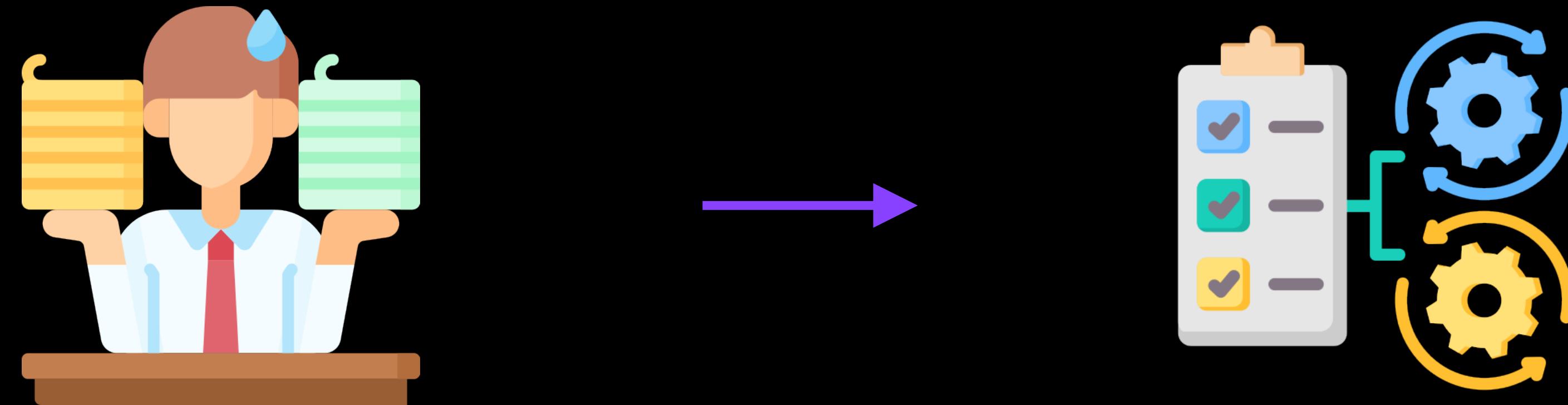


퀴즈 타임



QUIZ 8번부터 14번을 풀어봐요!

왜 QA 자동화 도구가 필요한가요?



QA 자동화 도구
테스트와 품질 보증 작업을 자동으로 수행하는 소프트웨어

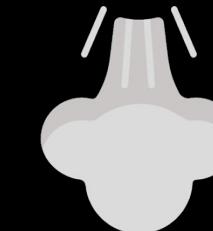
- 수작업 테스트의 한계를 극복하고, 효율성을 향상
- 빠른 피드백 제공으로 릴리스 주기를 단축



자동화 도구로 무엇이 가능할까?

테스트 속도 증가

5배 이상 빠른 테스트 수행



인적 오류 감소

모든 테스트가 동일한 절차로 실행



확장성

대규모 프로젝트에서 안정적 테스트 가능



Selenium: 브라우저를 제어하는 마법사

- WebDriver: 브라우저 제어
- Selenium Grid: 병렬 테스트 실행
- Selenium IDE: 테스트 기록 및 재생

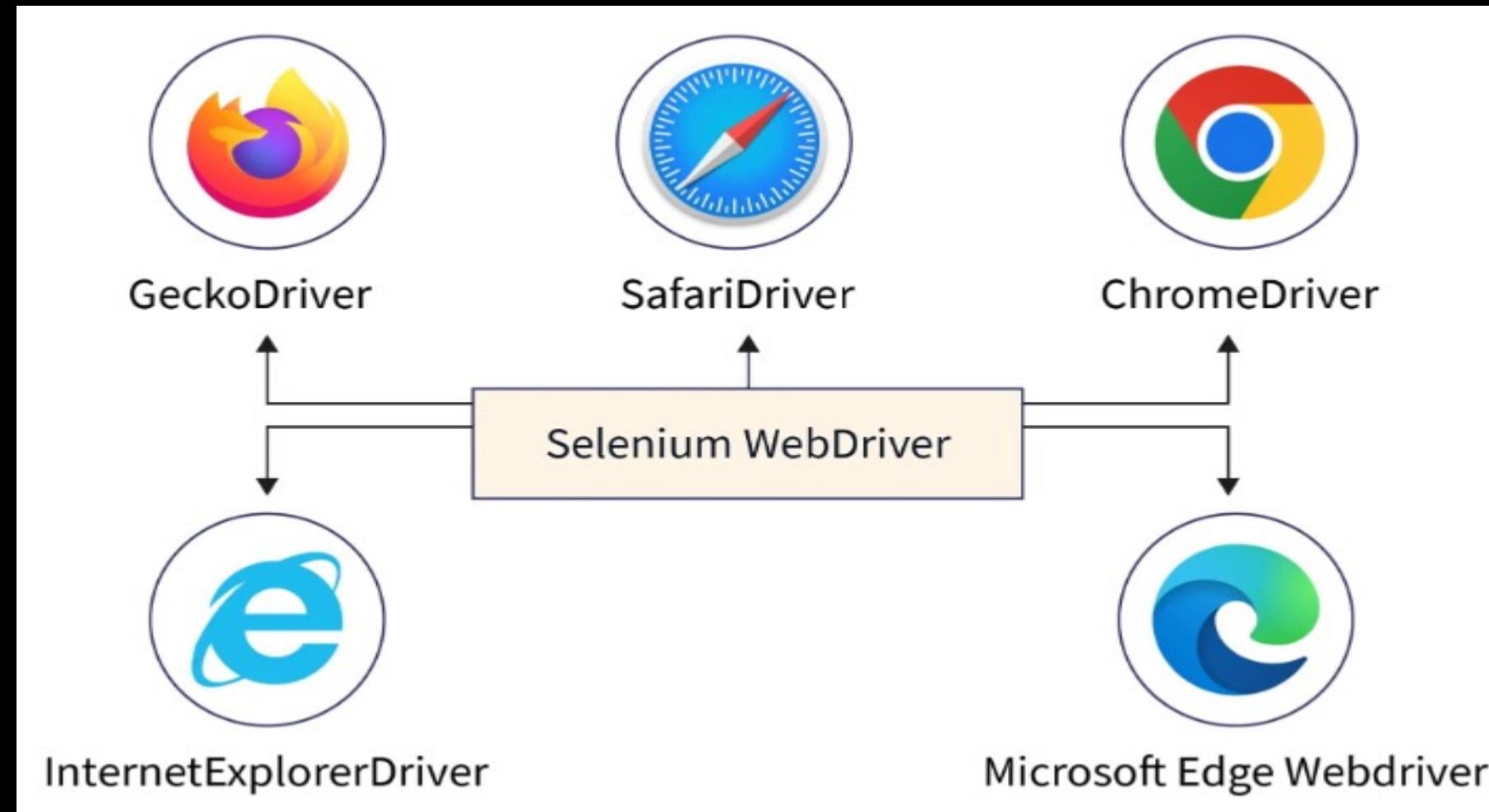
The screenshot shows the Selenium IDE interface running in a browser window. The title bar says "Selenium IDE - SeleniumHQ*". The main area is titled "SeleniumHQ*" and contains a "Tests" dropdown and a search bar. Below that is a table with columns "Command", "Target", and "Value". The table rows represent the following steps:

Command	Target	Value
1. open	/	
2. click at	link=Projects	22,13
3. click at	xpath= //a[contains(text(),'Selenium WebDriver')...]	37,7
4. click at	link=Projects	20,17
5. click at	link=Selenium IDE	88,11
6. click at	link=Projects	48,18
7. click at	xpath= //a[contains(text(),'Selenium Grid')][2]	79,9

Selenium

웹 애플리케이션 테스트를 자동화하기 위한 오픈소스 도구

Selenium WebDriver로 마법 부리기



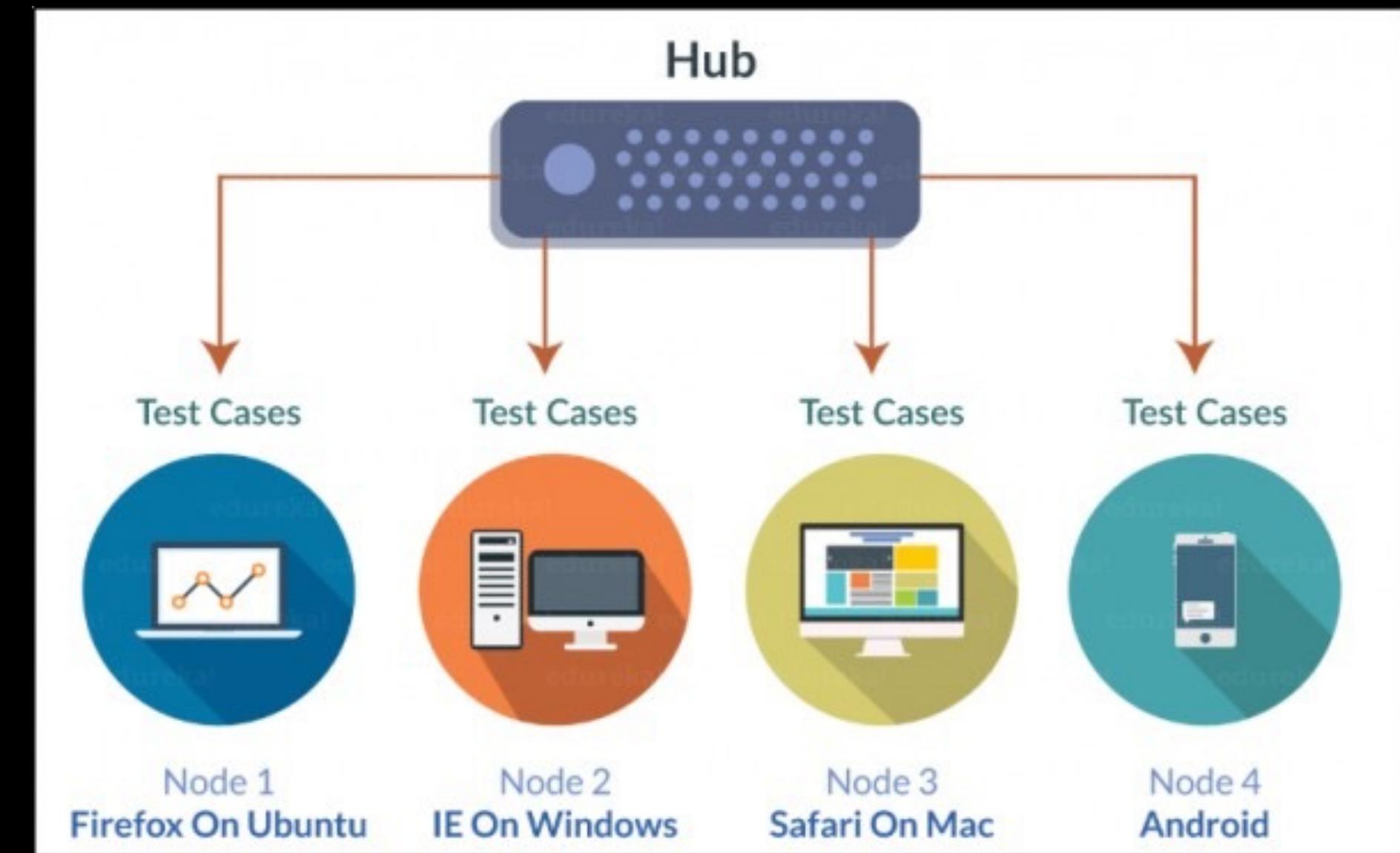
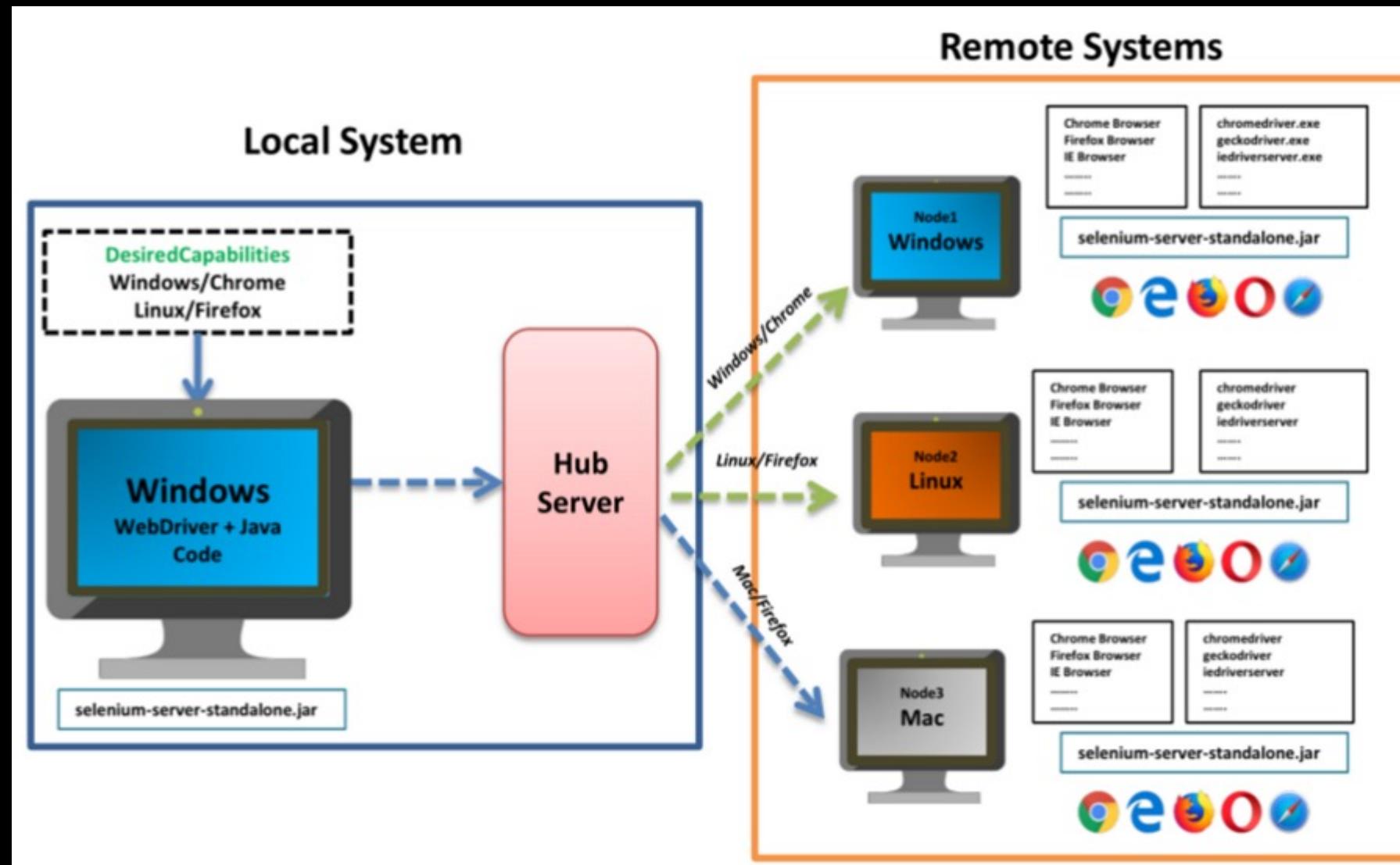
```
from selenium import webdriver  
from selenium.webdriver.common.by import By  
from selenium.webdriver.common.keys import Keys  
import time  
  
# Chrome WebDriver 실행  
driver = webdriver.Chrome()  
  
try:  
    # 특정 웹사이트로 이동  
    driver.get("https://example.com")  
    print("페이지 타이틀:", driver.title)
```

기능 : 브라우저 열기, 탐색, 버튼 클릭, 데이터 입력

사용자의 행동을 시뮬레이션하여 테스트를 자동화



병렬 테스트로 시간을 절약하자!



Selenium Grid

병렬로 여러 브라우저와 운영 체제에서 테스트 실행

- 테스트 시간 단축
- 다양한 환경에서의 동작 검증



구성 요소

Hub: 테스트 요청 관리

Node: 테스트 실행



JUnit과 TestNG: 테스트의 든든한 동반자



JUnit

Java 기반 단위 테스트 도구로 간단하고
직관적인 테스트 작성 가능

단순성과 신뢰성에 중점



TestNG

JUnit의 확장 버전으로 병렬 실행과 데이터
기반 테스트를 지원

복잡한 테스트 워크플로우를 쉽게 관리



JUnit으로 테스트하는 방법

```
import org.junit.jupiter.api.Test;  
import static org.junit.jupiter.api.Assertions.*;  
  
public class CalculatorTest {  
  
    // Calculator 인스턴스 생성  
    private final Calculator calculator = new Calculator();  
  
    @Test  
    public void testAdd() {  
        // 덧셈 테스트  
        int result = calculator.add(2, 3);  
        assertEquals(5, result, "2 + 3 should equal 5");  
    }  
}
```



기본 사용법

테스트 클래스와 메서드를 정의
@Test 애노테이션을 사용해 단위 테스트 실행

간단한 문법으로 빠르게 테스트 구현 가능



TestNG로 더 많은 걸 해보자!

```
import org.testng.Assert;
import org.testng.annotations.Test;

public class CalculatorTest {

    private final Calculator calculator = new Calculator();

    @Test
    public void testAdd() {
        // 단체 테스트
        int result = calculator.add(2, 3);
        Assert.assertEquals(result, 5, "2 + 3 should equal 5");
    }
}
```



주요 기능

- **병렬 실행**
여러 테스트를 동시에 실행
- **테스트 그룹화**
관련된 테스트를 묶어 실행
- **데이터 기반 테스트**
외부 데이터를 기반으로 테스트 수행

복잡한 테스트 요구 사항을 간단하게 처리



SonarQube

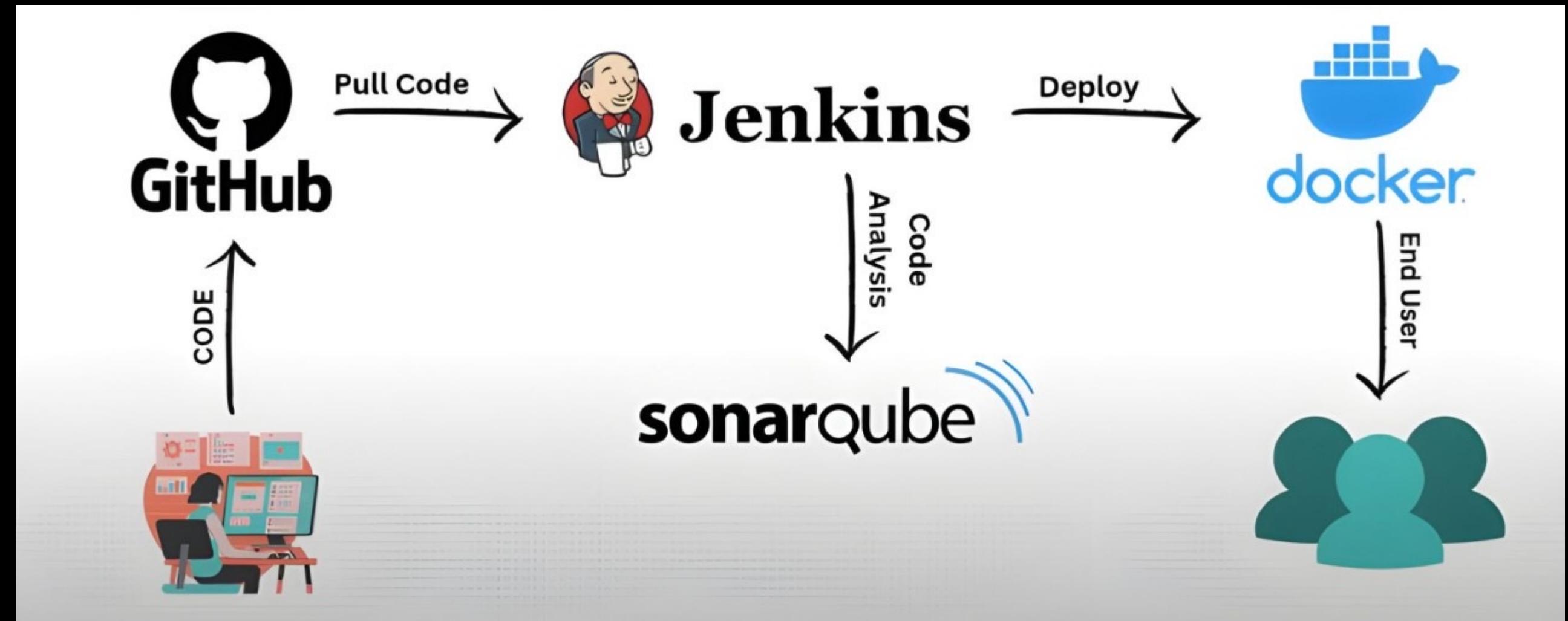
코드 품질을 분석하고, 잠재적인 결함과 보안 취약점을 찾아내는 도구



주요 기능

- 코드 스멜(Code Smell) 탐지
- 보안 취약점 및 버그 탐지
- CI/CD 파이프라인과 통합 가능

SonarQube를 CI/CD와 통합하기



- Jenkins나 GitLab CI/CD에 SonarQube Scanner를 추가
- 빌드 단계에서 코드 품질 분석 자동화



코드 품질을 배포 전에 검증
CI/CD 프로세스의 품질 관리 강화



왜 QA 도구와 CI/CD를 연결해야 할까요?

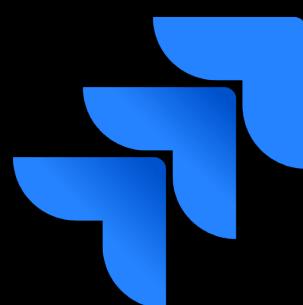
BEFORE



AFTER



- 수동 품질 검증으로 인한 배포 지연
- 품질 문제 발견 시 해결 시간 증가
- 테스트 자동화를 통해 반복 작업 제거
- 배포 전 품질 문제 사전 탐지
- 실시간 피드백을 제공하여 팀 협업



JIRA

Jira

프로젝트와 이슈를 관리하는 도구로, 팀 협업 강화

Jira와 CI/CD 통합의 장점

코드 변경이 Jira 이슈와 연동되어 추적 가능
빌드/배포 상태를 Jira에서 실시간으로 확인

주요 기능

GitLab이나 Jenkins에서 빌드 성공/실패를 Jira 이슈에 자동 업데이트

Jira와 GitLab/Jenkins 통합하기

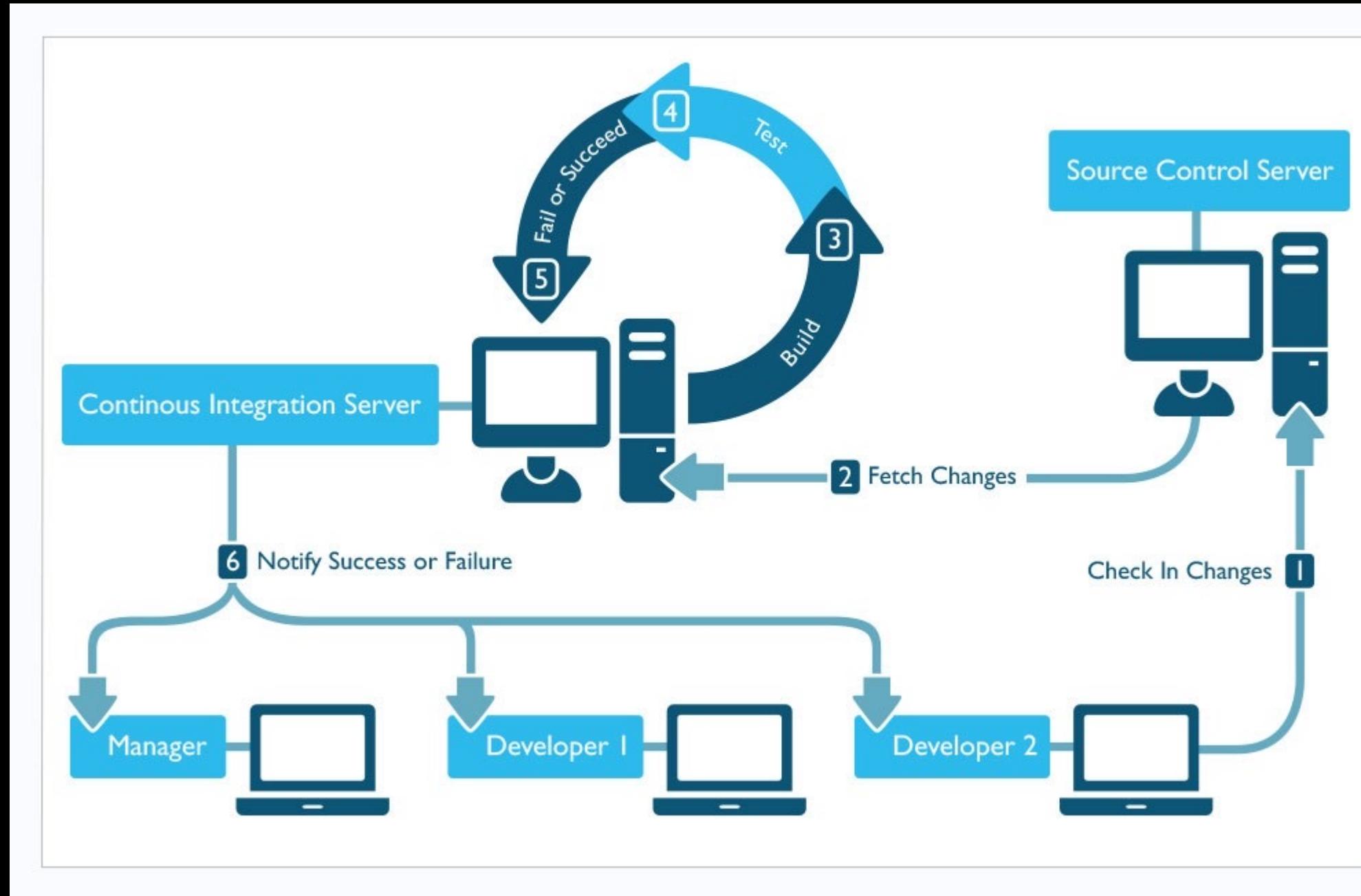
The screenshot shows the 'WebHooks' configuration page in Jira. A new webhook listener is being created with the following details:

- Name:** New WebHook Listener
- Status:** Enabled
- URL:** <http://example.com/rest/webhooks/webhook1>
- Description:** (Empty)
- Events:** Issue related events
- All issues** is selected under Events.
- Syntax help:** A table showing event types for Comment, Worklog, Issue link, and Issue.

Comment	Worklog	Issue link	Issue
<input type="checkbox"/> created	<input type="checkbox"/> created	<input type="checkbox"/> created	<input type="checkbox"/> created
<input type="checkbox"/> updated	<input type="checkbox"/> updated	<input type="checkbox"/> deleted	<input type="checkbox"/> updated
<input type="checkbox"/> deleted	<input type="checkbox"/> deleted		<input type="checkbox"/> deleted
			<input type="checkbox"/> worklog changed

- Jira에서 GitLab/Jenkins의 웹훅(Webhook) 설정
- 코드 커밋 시 Jira 이슈와 자동 연동
- 배포 결과를 Jira 이슈에 기록

CI/CD 파이프라인에서 빛을 발하다



Selenium을 CI/CD에 통합해야 하는 이유

- 배포 전 브라우저 기반 테스트 자동화
- 빠르고 안정적인 릴리스 보장

Jenkins에서 Selenium 실행하기

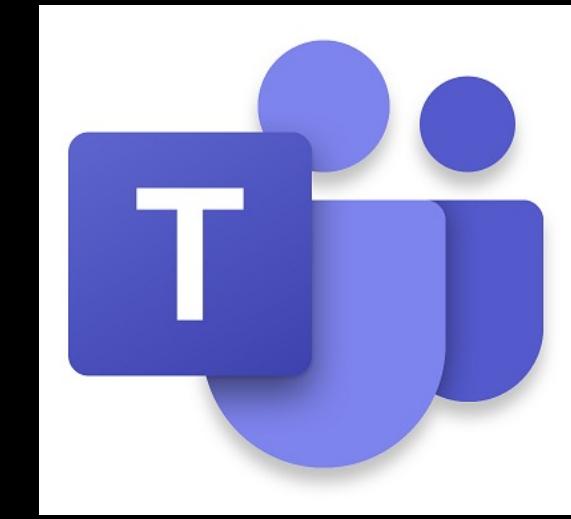


- Jenkins에서 Selenium WebDriver 환경 설정
- 파이프라인에 Selenium 테스트 단계 추가



빌드 후 자동화된 테스트 실행
테스트 실패 시 즉시 빌드 중단

Slack, Teams가 CI/CD에 들어오면



- 팀원 간 실시간 피드백을 강화하여 협업 효율 향상
- 빌드, 테스트, 배포 상태를 즉시 알림으로 전달



주요 기능

- 빌드 성공/실패 알림
- 테스트 결과 보고
- 긴급 상황 시 실시간 알림 제공

Slack과 GitLab CI/CD 통합하기



A screenshot of the GitLab web interface for a project named "Shinhan_Campus". The sidebar on the left shows various project sections like Project overview, Details, Activity, Releases, Repository, Issues, Merge Requests, CI / CD, Security & Compliance, Operations, Packages & Registries, Analytics, Wiki, Snippets, Members, and Settings. A red arrow points to the "Integrations" section in the sidebar, which is currently highlighted. The main content area displays basic project information: "Project overview", "Details", "Activity", "Releases", "Repository", "Issues 0", "Merge Requests 0", "CI / CD", "Security & Compliance", "Operations", "Packages & Registries", "Analytics", "Wiki", "Snippets", "Members", and "Settings".

Slack 알림 설정 방법

GitLab CI/CD 파이프라인에 Slack Webhook을 설정
.gitlab-ci.yml에 알림 단계 추가

주요 알림 내용

- 빌드 성공/실패
- 테스트 결과
- 배포 완료





Teams와 Jenkins 통합하기

The screenshot shows the 'Connectors for "jenkins-notifications" channel in "testing" team' page. On the left, there's a sidebar with 'MANAGE' and 'CATEGORY' sections. Under 'CATEGORY', 'All' is selected. In the main area, 'Connectors for your team' is listed. The 'Jenkins' connector, which has a icon of a person holding a coffee cup, is highlighted with a red box. It is described as 'Continuous Integration and Continuous Delivery'. To its right is a 'Configure' button. Below it are other connectors: 'Forms' (easily create surveys, quizzes, and polls), 'Azure DevOps' (collaborate on and manage software projects online), 'RSS' (get RSS feeds for your group), and 'Incoming Webhook'. Each of these has an 'Add' button to its right.

Teams 알림 설정 방법

Jenkins에서 Teams Webhook 플러그인을 설치
Jenkins 파이프라인에 알림 단계 추가



- 배포 및 테스트 상태를 팀 전체에 실시간으로 공유
- 문제 발생 시 즉각 대응 가능

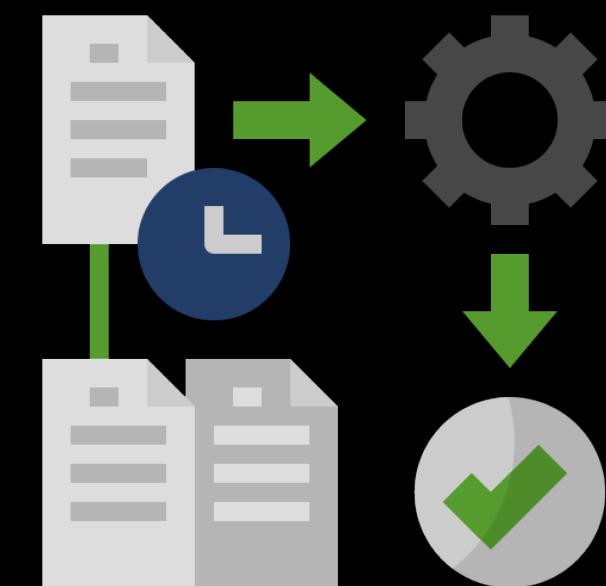
QA 팀, CI/CD로 인해 어떻게 달라지나요?



단순 테스트

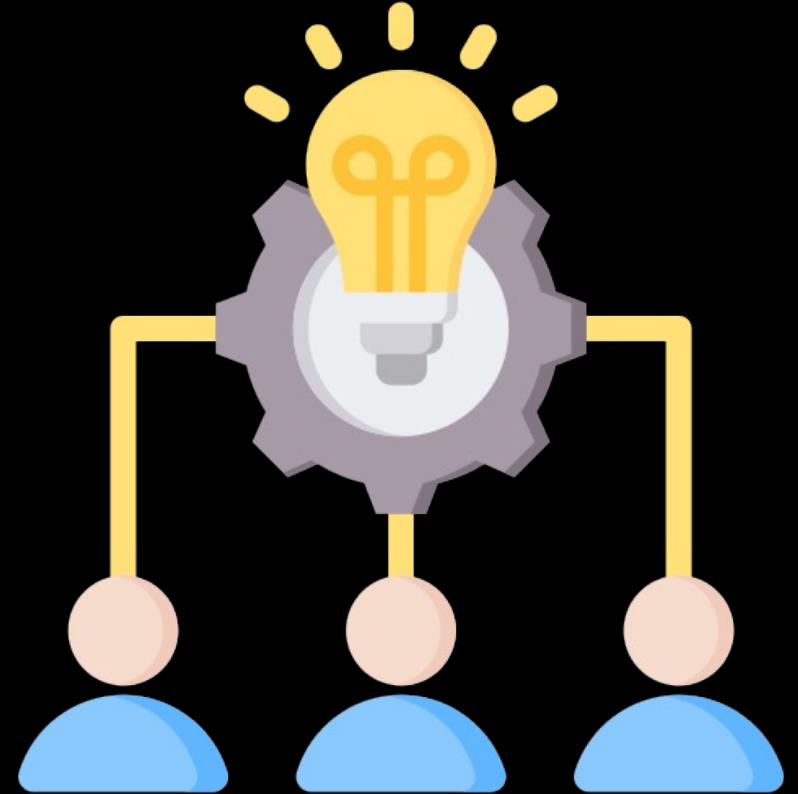


- QA 팀의 기술 역량 강화 필요
- DevOps 및 개발팀과의 협업 확대



자동화 설계와 품질 검증

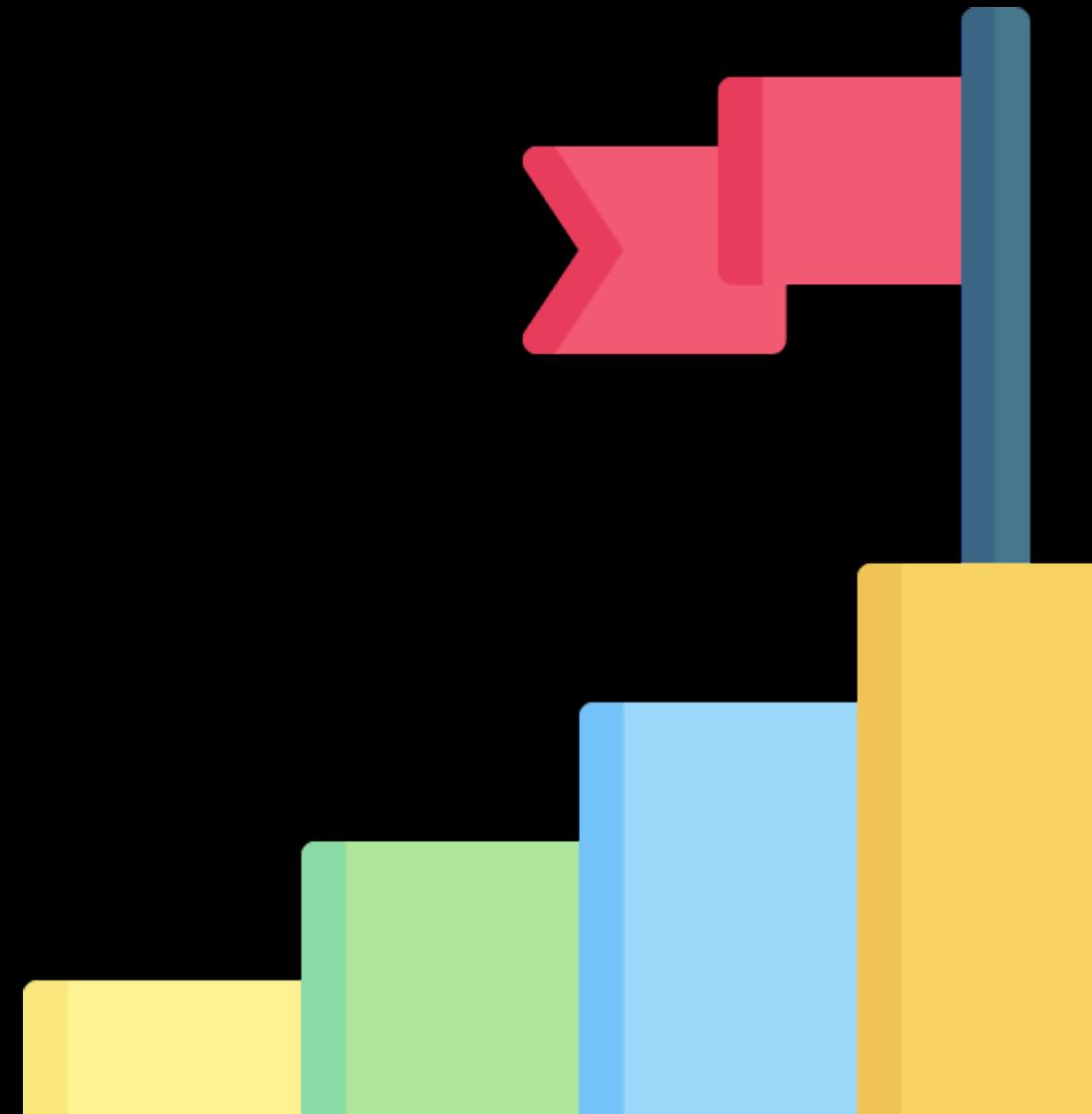
QA 팀이 CI/CD 환경에 적응하려면?



- 자동화 도구 학습(Selenium, JUnit 등)
- CI/CD 파이프라인의 구조와 동작 원리 이해
- DevOps 철학 및 Agile 개발 방식의 수용
- 팀 간 협업 확대



QA 팀의 기술 향상을 위한 로드맵



테스트 자동화

Selenium, TestNG 학습

CI/CD 도구 활용

Jenkins, GitLab CI/CD 이해

코드 품질 분석

SonarQube 및 코드 리뷰 참여

+ 교육과 지원

- 내부/외부 교육 프로그램 도입
- 멘토링 시스템 구축



CI/CD 파이프라인에서 발생하는 문제



문제

테스트 자동화 부족으로 인한 릴리스 지연

파이프라인 단계 간 의존성 과다로 인해 작업 병목 발생

오류 탐지 후 느린 문제 해결



CI/CD 파이프라인 문제 해결법



해결 방안

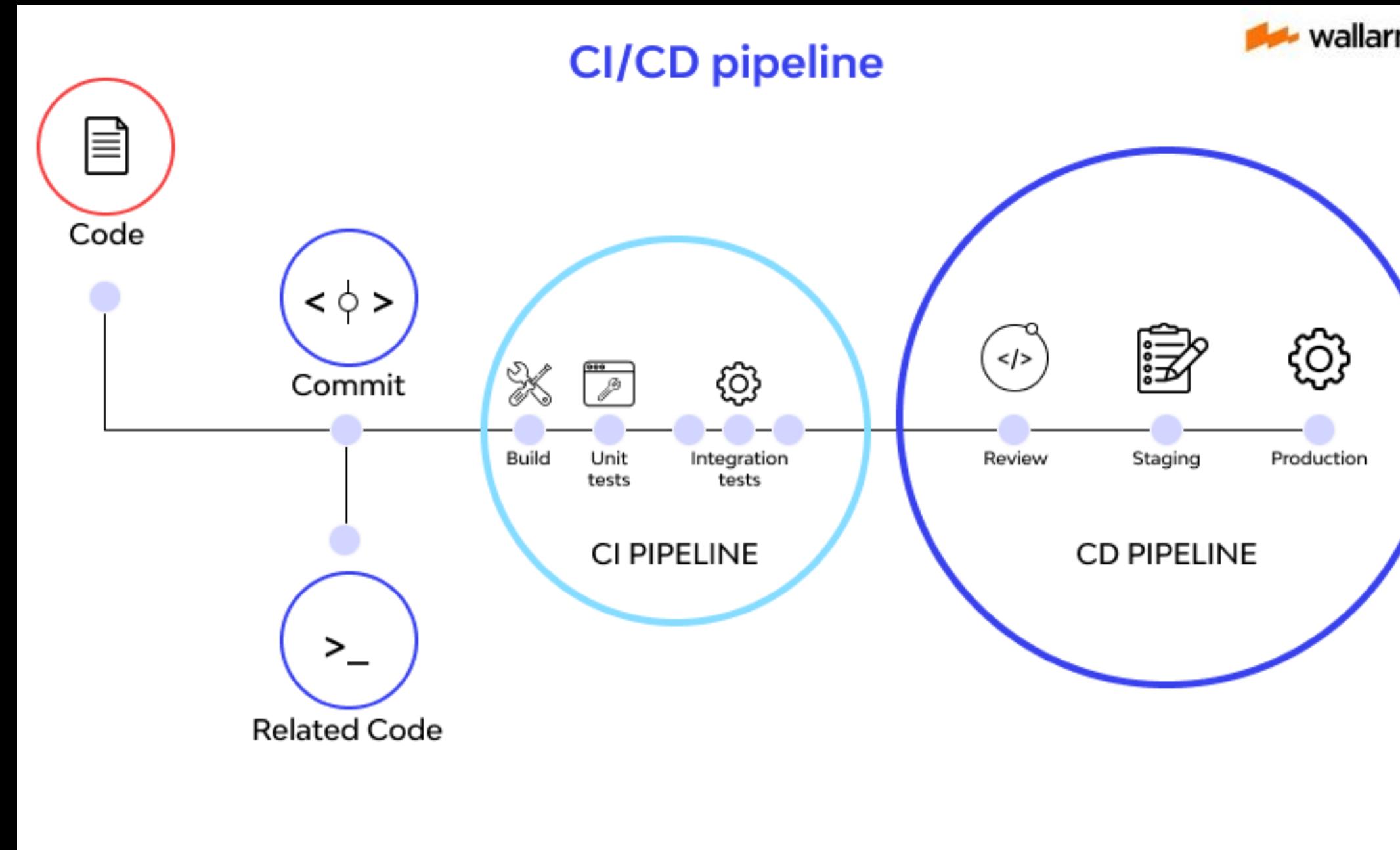
자동화 테스트 도입: 반복 작업 제거

단계별 독립성 강화: 병목 현상 방지

CI/CD 도구의 모니터링 기능 활용



문제를 줄이기 위한 지속적 개선



- 파이프라인 단계의 최적화
- 정기적인 파이프라인 성능 점검
- 오류 로그 분석 및 자동화된 해결책 구현

CI/CD 도구, 모두가 쉽게 쓸 수 있을까?



도구 학습 곡선이 높음

기존 워크플로우와의 충돌

특정 환경에 의존한 설정 문제

도구 활용의 도전, 어떻게 극복할까?



교육과 문서화

- 팀원들에게 초기 교육 제공
- 도구 사용법 및 FAQ를 문서화하여 쉽게 참조 가능



표준화된 설정

- 환경 설정을 통일하여 충돌 방지



점진적 도입

- 기존 워크플로우에 통합할 때 점진적으로 적용



QA와 개발 팀의 협력, 무엇이 문제일까?

QA



개발팀



- QA는 품질을, 개발 팀은 속도를 중시
- 의사소통 부족으로 인한 품질 이슈
- 개발 단계에서 QA의 부족한 참여



협력을 강화하는 해결책

공동 목표 설정

속도와 품질의 균형을 맞춘 목표 설정

초기 단계 협업

QA를 설계 및 요구사항 정의 단계부터 참여



피드백 루프

QA가 개발 팀에 빠르게 피드백 제공

정기적인 커뮤니케이션

데일리 스탠드업 미팅을 통해 상태를 공유



퀴즈 타임



QUIZ 15번부터 23번을 풀어봐요!