CI/CD

목차

CI/CD 개념

CI(지속적 통합)

- 개발자가 코드 변경사항을 자주 병합하는 방식
- 자동화된 빌드와 테스트로 통합 문제 조기 발견
- 코드 품질 향상 및 버그 조기 식별
- 팀 협업 효율성 증대

CD(지속적 배포/전달)

- 지속적 배포: 자동으로 프로덕션 환경까지 배포
- 지속적 전달: 프로덕션 배포 전 수동 승인 단계 포함
- 배포 과정 자동화로 인적 오류 감소

CI/CD 파이프라인

- 코드 커밋부터 배포까지 자동화된 일련의 프로세스
- 주요 단계: 코드 통합, 빌드, 테스트, 배포
- 각 단계별 자동화된 피드백 제공
- 도구 연동을 통한 원활한 워크플로우 구성

CI/CD 도입 이점

- 개발 주기 단축 및 생산성 향상
- 소프트웨어 품질 및 안정성 개선
- 배포 위험 감소 및 롤백 용이성
- 팀 협업 강화 및 개발 문화 개선

CI/CD 중요성

빠른 개발 주기와 안정적인 배포를 통해 비즈니스 민첩성을 높이고, 고객에게 지속적인 가치를 제공할 수 있게 한다.	
소프트웨어 개발 생명주기에서의 역할	- 개발, 테스트, 배포 단계를 자동화하여 전체 생명주기 가속화 - 각 단계 간 원활한 전환으로 병목현상 제거 - 지속적인 피드백을 통한 반복적 개선 가능
개발 생산성 향상	- 수동 작업 최소화로 개발자가 핵심 업무에 집중 가능 - 반복 작업 자동화로 인적 오류 감소 - 빠른 릴리스 주기로 시장 대응력 향상
품질 보증 및 안정성	- 자동화된 테스트로 버그 조기 발견 및 수정 - 일관된 배포 프로세스로 환경 간 차이 최소화 - 문제 발생 시 신속한 롤백 메커니즘 제공

Jenkins 소개

기본 특징 및 아키텍처

- Java 기반의 오픈소스 자동화 서버
- 마스터-슬레이브 아키텍처로 분산 빌드 지원
- 웹 인터페이스를 통한 직관적인 관리

플러그인 생태계

- 1,500개 이상의 플러그인으로 확장성 제공
- 다양한 빌드 도구, 언어, 플랫폼 지원
- 커뮤니티 기반 플러그인 개발 및 유지보수
- 필요에 따른 커스텀 플러그인 개발 가능

파이프라인 구성 방법

- Jenkinsfile을 통한 코드형 파이프라인
- 선언적/스크립트형 파이프라인 두 가지 문법 지원
- 멀티브랜치 파이프라인으로 브랜치별 빌드 자동화
- 병렬 실행 및 조건부 단계 구성 가능

대규모 프로젝트 활용

- 마이크로서비스 아키텍처에서 다중 파이프라인 관리
- 공유 라이브러리를 통한 코드 재사용성 향상
- 복잡한 워크플로우 오케스트레이션
- 엔터프라이즈 환경에서의 확장성 및 안정성 제공

Jenkins 장단점

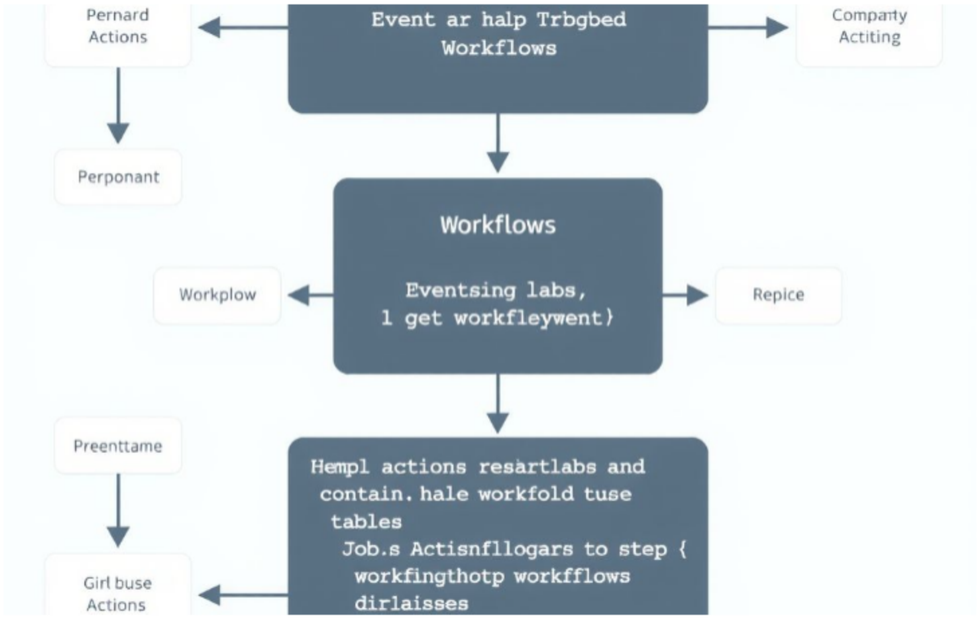
장점

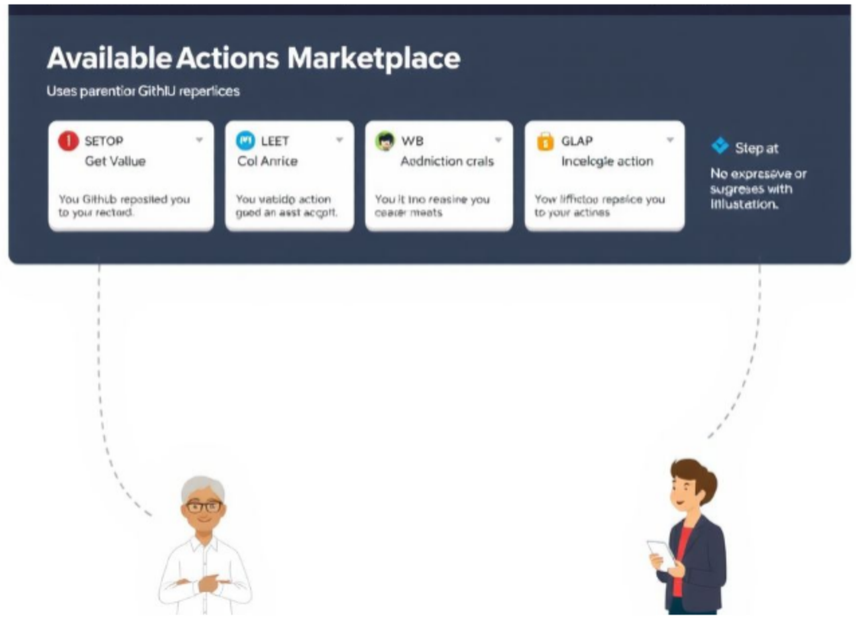
- 풍부한 플러그인 생태계로 거의 모든 도구와 통합 가능
- 높은 수준의 커스터마이징과 유연성 제공
- 성숙한 커뮤니티와 광범위한 문서화 지원
- 복잡한 빌드 및 배포 시나리오 처리 가능
- 분산 빌드 환경 구성으로 확장성 확보
- 다양한 인증 및 권한 관리 기능
- 무료 오픈소스로 라이센스 비용 없음

단점

- 초기 설정 및 유지보수가 복잡하고 시간 소모적
- 웹 인터페이스가 현대적이지 않고 사용자 경험이 부족
- 높은 리소스 소비로 성능 이슈 발생 가능
- 플러그인 간 호환성 문제 발생 가능성
- 추가 서버 할당 필요
- 학습 곡선이 가파름
- 컨테이너 기반 배포에 추가 구성 필요

GitHub Actions 소개





기본 개념 및 워크플로우

- GitHub Actions는 GitHub 저장소에서 직접 CI/CD 워크플로우를 자동화하는 도구
- YAML 파일로 정의되며 이벤트 기반 트리거 방식 사용
- 워크플로우, 작업, 단계로 구성된 계층적 구조
- 다양한 이벤트(push, PR, 일정 등)에 반응하여 실행

GitHub 통합 및 마켓플레이스

- GitHub 저장소와 원활하게 통합되어 별도 서버 불필요
- 액션 마켓플레이스를 통해 사전 구성된 작업 활용 가능
- 커뮤니티에서 제공하는 다양한 액션으로 빠른 구현
- 저장소 시크릿을 통한 안전한 인증 정보 관리

GitHub Actions 장단점

장점

- 별도 인프라 구축 불필요
- 간편한 YAML 기반 설정으로 빠른 시작 가능
- 무료 티어 제공으로 소규모 프로젝트에 경제적
- 마켓플레이스를 통한 다양한 액션 즉시 활용
- 직관적인 UI로 실행 상태 모니터링 용이
- 자동 트리거 및 이벤트 기반 실행 지원
- 멀티 플랫폼(Linux, Windows 등) 지원

단점

- 워크플로우 구현 시 YAML 파일의 복잡성
- 대규모 CI/CD 파이프라인에서 유연성 제한적
- 실행 시간 제한으로 장시간 작업에 부적합
- 고급 커스터마이징 옵션이 제한적
- 자체 호스팅 러너 설정 및 관리가 복잡
- 외부 시스템과의 통합이 Jenkins보다 제한적
- 기업 환경에서 거버넌스 및 규정 준수 기능 부족

Jenkins vs GitHub Actions 비교

성능 및 확장성	사용 편의성	커스터마이징	비용 및 리소스
- Jenkins: 대규모 빌드와 복잡한 파이프라인에 최적화 - 분산 빌드 지원으로 높은 확장성 - 다양한 워크로드 처리 가능 - GitHub Actions: 중소규모 프로젝트에 적합 - 실행 시간과 리소스에 제한 존재	- GitHub Actions: 직관적인 UI와 간편한 설정 - 빠른 학습 곡선으로 초보자 친화적 - Jenkins: 초기 설정과 학습이 복잡함 - 대시보드 사용이 덜 직관적 - 관리 인터페이스가 다소 구식	- Jenkins: 거의 무제한적인 커스터마이징 - 풍부한 플러그인 생태계(1500+) - 복잡한 로직 구현 가능 - GitHub Actions: 제한된 커스터마이징 - YAML 기반 구성의 한계 존재	- Jenkins: 자체 서버 필요로 초기 비용 발생 - 유지보수 인력 필요 - GitHub Actions: 소규모 프로젝트 무료 - 대규모 사용 시 비용 증가 - 인프라 관리 부담 없음

Jenkins는 복잡하고 대규모 환경에 적합하며 무제한 커스터마이징이 가능한 반면,
GitHub Actions는 설정이 간편하고 GitHub와 통합이 원활하여 소규모 프로젝트에 이상적.
프로젝트 규모, 복잡성, 팀의 기술적 역량에 따라 적절한 도구를 선택.

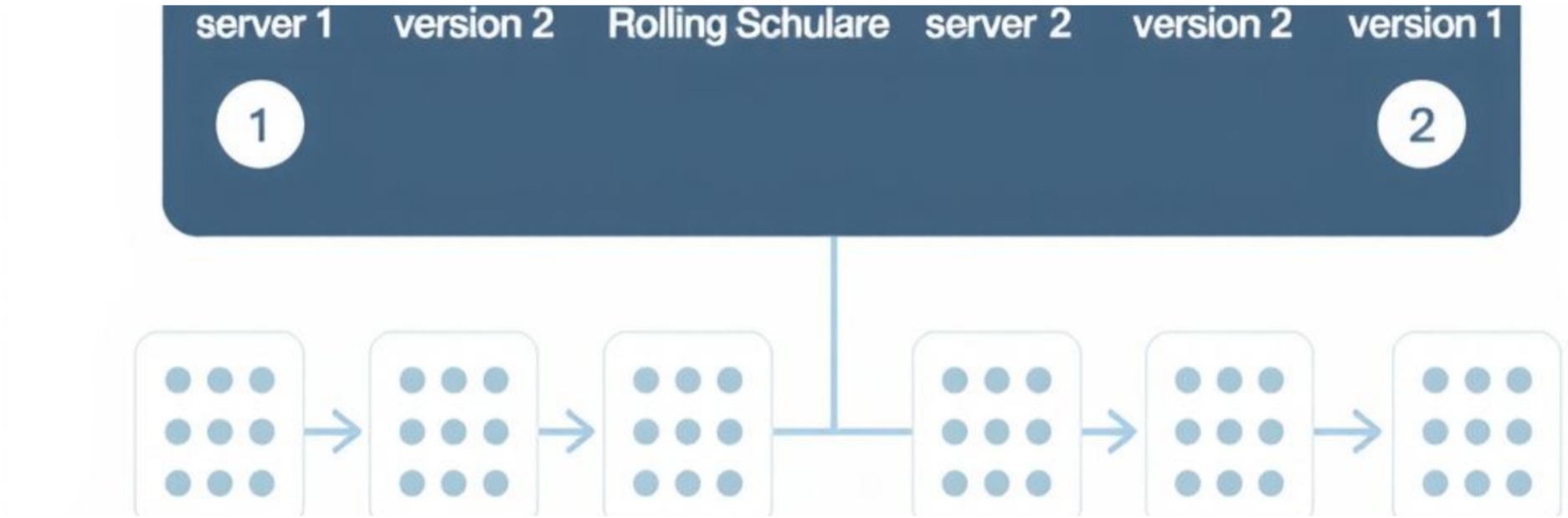
배포 전략

배포 전략 개요

배포 전략은 소프트웨어를 프로덕션 환경에 안전하고 효율적으로 릴리스하기 위한 방법론.
적절한 배포 전략을 선택하면 다운타임 최소화, 위험 감소, 사용자 경험 향상 등의 이점을 얻을 수 있다.

배포 전략의 중요성	- 서비스 중단 최소화로 비즈니스 연속성 보장 - 배포 실패 시 빠른 복구 메커니즘 제공 - 새 기능의 안전한 출시와 검증 가능
배포 전략 선택 시 고려사항	- 애플리케이션 특성 및 아키텍처 - 인프라 환경 및 가용 리소스 - 팀의 기술적 역량과 운영 성숙도
CI/CD와 배포 전략의 연계	- CI/CD 파이프라인에 자동화된 배포 전략 통합 - 테스트 자동화와 모니터링 시스템 연계 - 지속적 피드백을 통한 배포 프로세스 개선

롤링
배포 방식



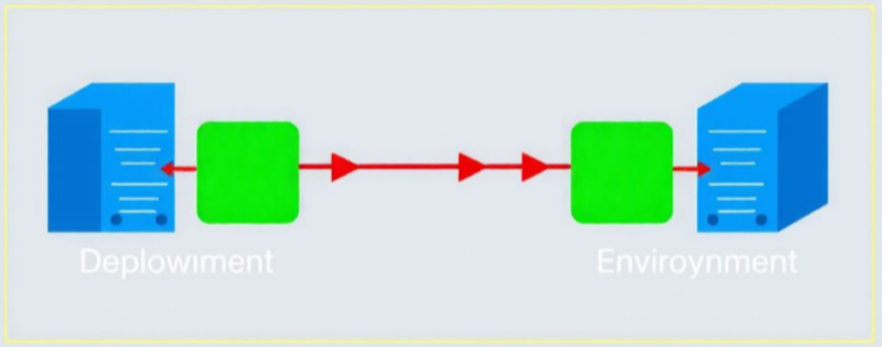
롤링 배포의 개념 및 특징

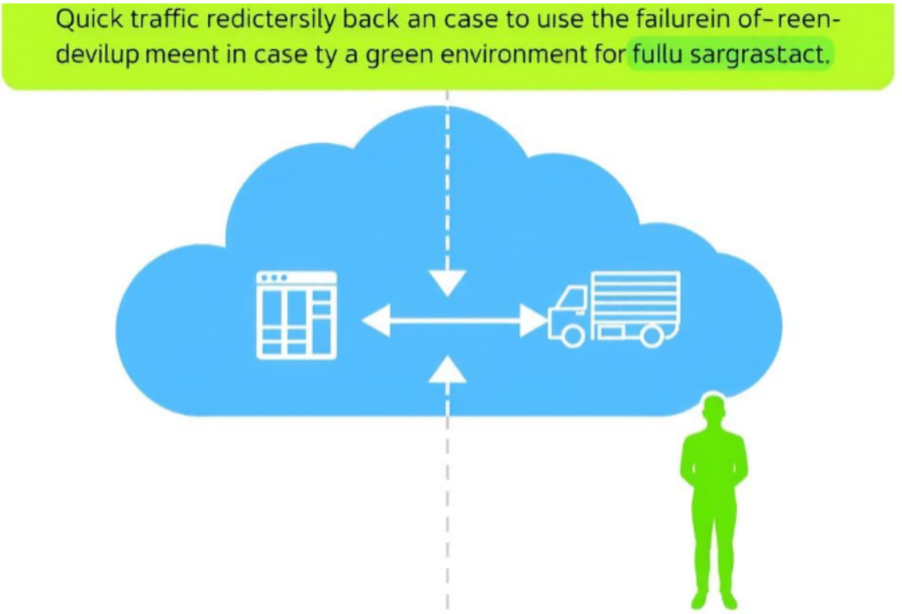
롤링 배포는 애플리케이션 인스턴스를 점진적으로 교체하는 방식.
기존 버전에서 새 버전으로 서버를 하나씩 업데이트하여 전체 시스템을 점진적으로 전환.

롤링 배포의 장단점

장점: 리소스 효율적, 추가 인프라 불필요, 구현 간단
단점: 롤백 과정 복잡, 배포 중 버전 불일치 발생, 전체 배포 시간 길어짐, 부분 장애 발생 가능성

블루/그린 배포 방식





- 두 개의 동일한 프로덕션 환경(블루/그린)을 유지
- 블루 환경에서 현재 버전 실행 중 그린 환경에 새 버전 배포
- 테스트 완료 후 트래픽을 블루에서 그린으로 한 번에 전환
- 문제 발생 시 즉시 이전 환경으로 롤백 가능

장점:
- 즉각적인 롤백 가능으로 위험 최소화
- 무중단 배포로 사용자 경험 향상
- 새 버전의 사전 검증 가능

단점:
- 인프라 자원 두 배 필요로 비용 증가

카나리 배포 방식

카나리 배포는 새 버전을 소수의 사용자에게만 점진적으로 노출시키는 전략. 이 방식은 실제 환경에서 새 버전의 안정성을 검증하면서도 위험을 최소화할 수 있다. 트래픽의 일부만 새 버전으로 라우팅하고 문제 발생 시 즉시 롤백이 가능하다.	
카나리 배포의 개념	- 새 버전을 소규모 사용자 그룹에 먼저 배포하는 방식 - '카나리'라는 이름은 광산의 유독가스 감지용 카나리새에서 유래 - 실제 프로덕션 환경에서 안전하게 테스트 가능
트래픽 분할 및 모니터링	- 트래픽 비율 조정: 5%, 20%, 50% 등 단계적 증가 - 실시간 모니터링: 오류율, 응답시간, 시스템 부하 등 관찰 - 자동화된 롤백 트리거: 임계값 초과 시 자동 롤백 설정
장단점 분석	- 장점: 위험 최소화, 실시간 사용자 피드백 수집 가능 - 장점: 점진적 출시로 시스템 안정성 확보 - 단점: 구현 복잡성 높음, 모니터링 시스템 필수, 관리 오버헤드

배포 전략 비교 분석

안정성	카나리 > 블루/그린 > 롤링 (위험 최소화 측면)
리소스 효율성	롤링 > 카나리 > 블루/그린 (리소스 사용 효율성)
구현 복잡성	롤링 < 카나리 < 블루/그린 (구현 난이도)
롤백 용이성	블루/그린 > 카나리 > 롤링 (롤백 속도 및 용이성)
적합한 상황	상황별 최적 전략 선택이 중요 (규모, 중요도 고려)

결론

CI/CD 도구와 배포 전략은 현대 소프트웨어 개발의 핵심 요소

- 배포 전략은 프로젝트 특성에 맞게 선택

QnA