

# DevOps : Jenkins & Docker



# 1 DevOps 의 이해



## IT 트렌드의 변화

- 소프트웨어는 비즈니스 지원이 아닌 비즈니스의 중심
- 고객과 사용자는 더 빠른 대응, 더 나은 품질과 안정성을 요구
- 기존의 역할과 책임, 개발 방식, 아키텍처, 인프라의 한계

### IT 기술 트렌드의 변화



# 일반적인 소프트웨어 개발과 유지보수



명확한 역할과 책임 / 폐쇄적 협업

개발  
단계



개발 후반부 통합 빌드 및 배포 / 테스트 횟수만큼 배포

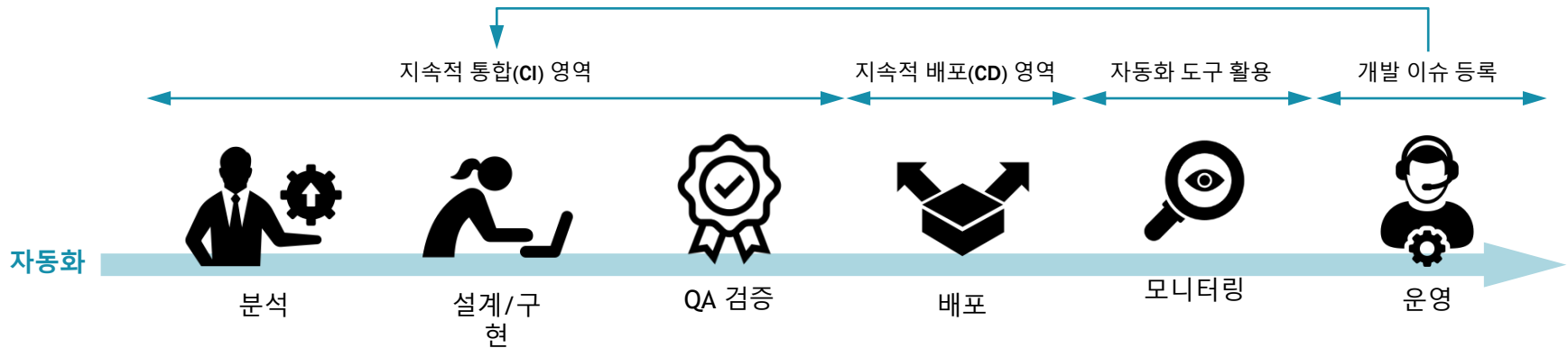
유지보수  
단계



1달에 1~2회 정기 배포 / 문제 발생 시 긴급 배포

느린 피드백 및 반영

# DevOps 소프트웨어 개발과 유지보수



물리적 One-Team은 아닐 수 있으나, **밀접한 협력 문화**

개발  
단계



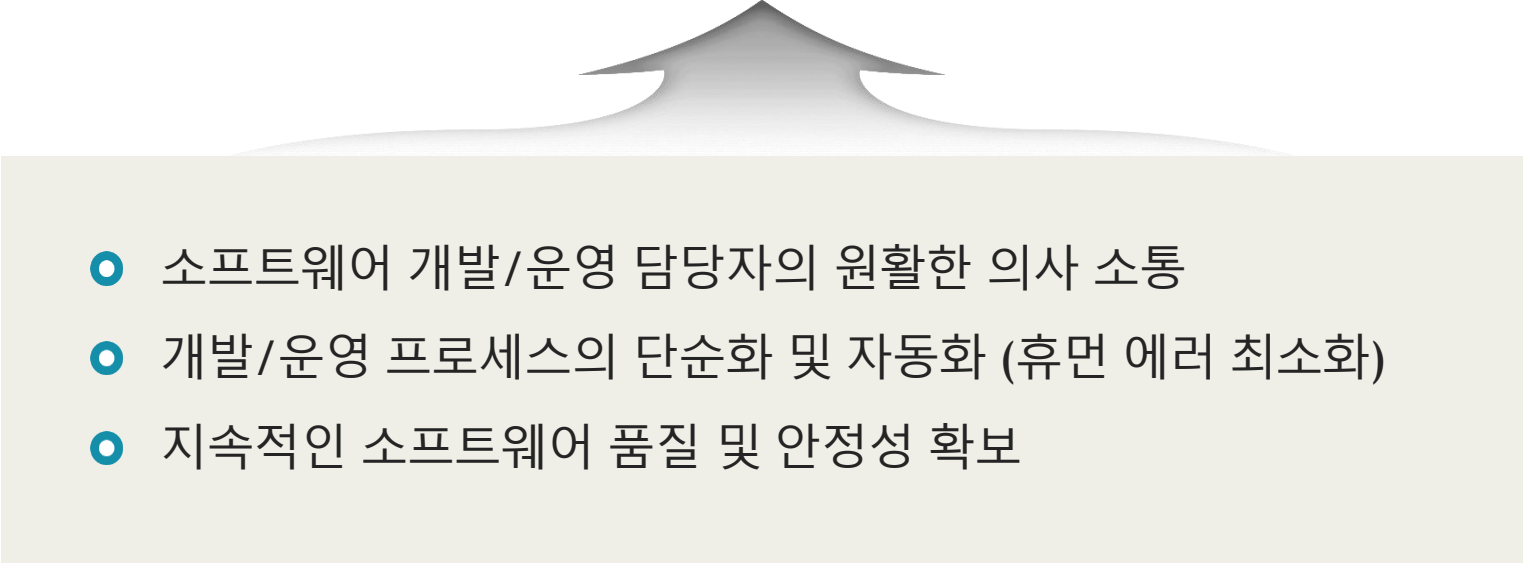
Iteration 및 Sprint에 따른 짧은 주기의 개발 및 배포 / 빠른 피드백 반영 / 진화하는 시스템

유지보수  
단계



하루에 여러 번 배포(자동화) / 빠른 피드백 반영

# 품질과 안정성이 확보된 소프트웨어를 더 빠르게 제공

- 
- 소프트웨어 개발/운영 담당자의 원활한 의사 소통
  - 개발/운영 프로세스의 단순화 및 자동화 (휴먼 에러 최소화)
  - 지속적인 소프트웨어 품질 및 안정성 확보

**자동화**된 지속적 통합(CI) / 지속적 배포(CD) 환경 구축



모든 SW 개발에 적용 가능한 정의

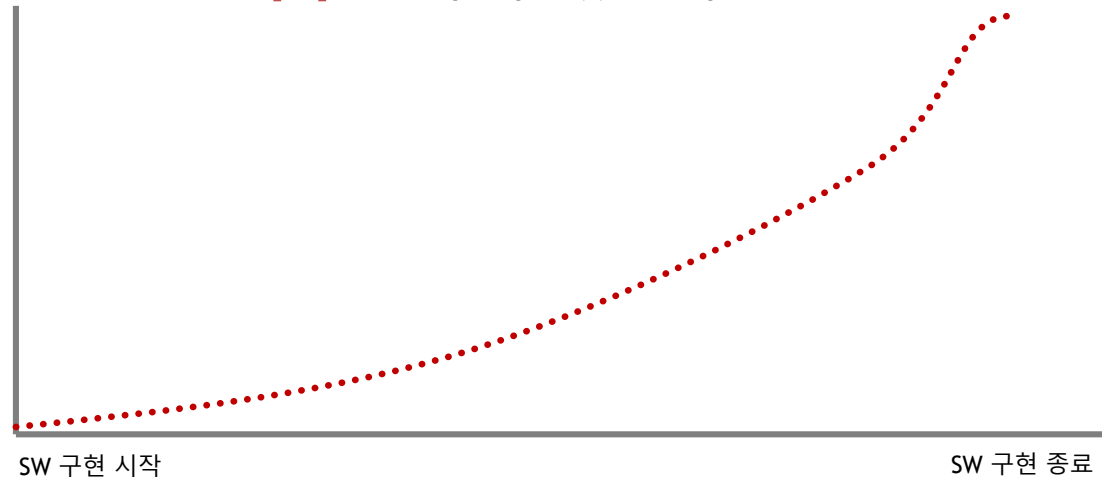


## “소프트웨어 구현 후반부에”

### 소스코드/소프트웨어 문제들

- 빌드 오류 수
- 정적분석 위반사항 수
- 보안 룰셋 위반사항 수
- 테스트 커버리지 불만족
- 함수, 모듈 크기 위반율
- 복잡도 위반율

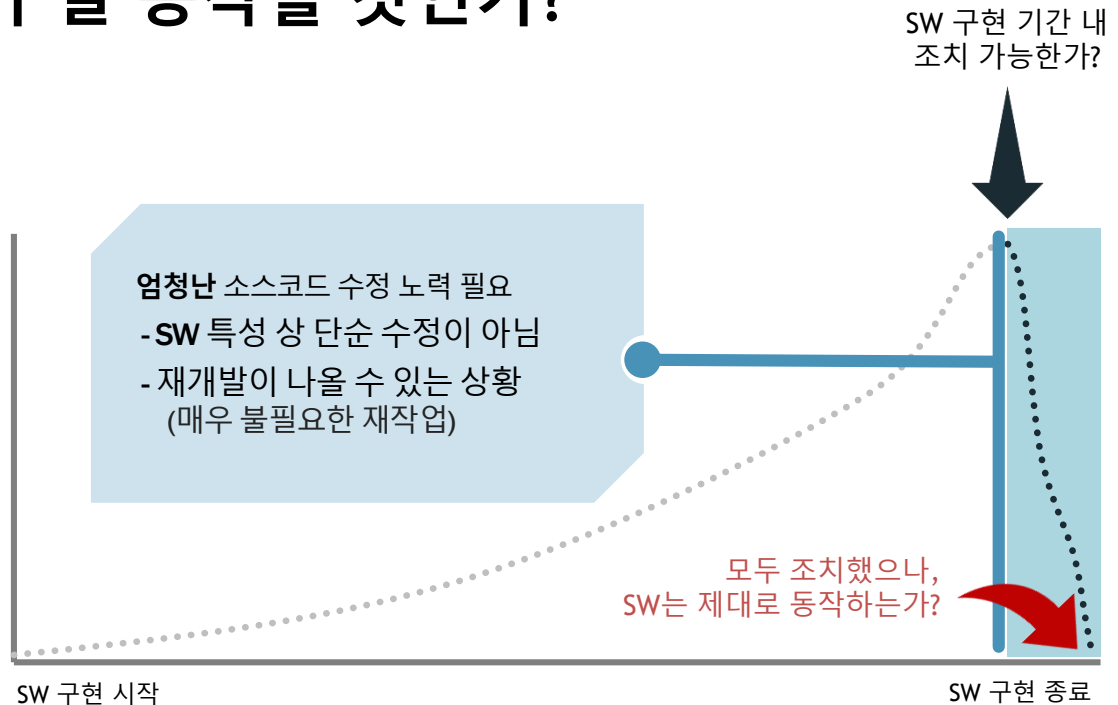
“소스코드가 품질 요건을  
**너무** 많이 위반했는데요...”



# SW 구현 종료까지 조치 가능한가? 조치가 끝나면 SW가 잘 동작할 것인가?

### 소스코드/소프트웨어 문제들

- 빌드 오류 수
- 정적분석 위반사항 수
- 보안 룰셋 위반사항 수
- 테스트 커버리지 불만족
- 함수, 모듈 크기 위반율
- 복잡도 위반율

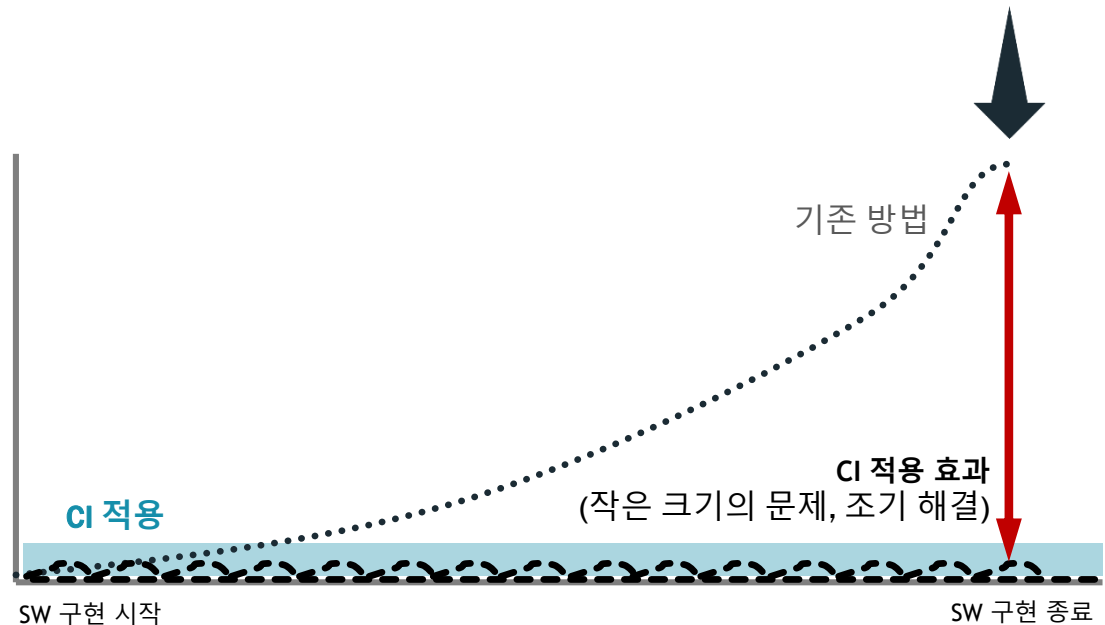


## 가능한 방법은?

SW 구현 시작부터 지속적 통합(CI)을 적용,  
도구가 **알아서** 검사/보고하여 품질과 안정성 조기 확보

### 소스코드/소프트웨어 문제들

- 빌드 오류 수
- 정적분석 위반사항 수
- 보안 룰셋 위반사항 수
- 테스트 커버리지 불만족
- 함수, 모듈 크기 위반율
- 복잡도 위반율

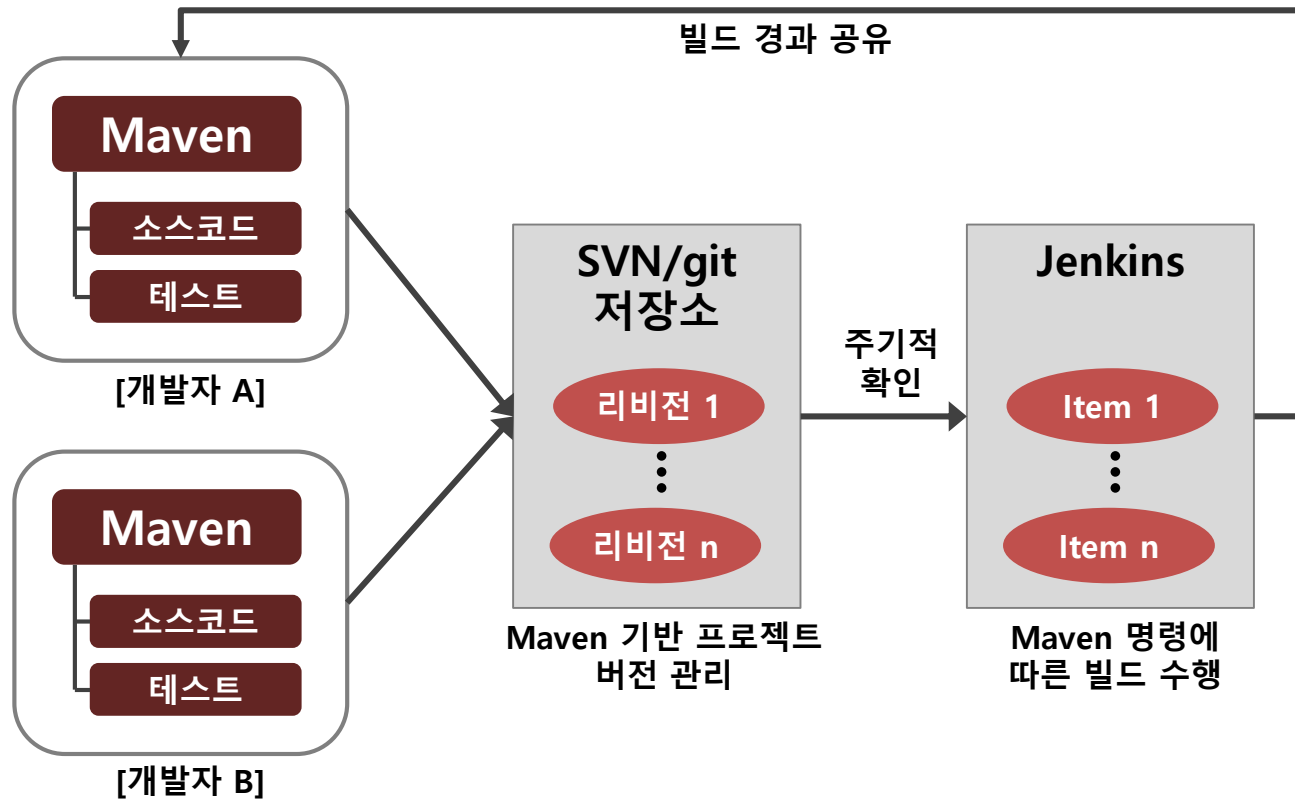


## 지속적 통합(CI)과 지속적 배포(CD)의 장점

---

- 소스코드/소프트웨어 빌드 및 품질 문제점을 빠르게 식별하고 조치
- 빌드, 품질 검사, 문제점 보고, 배포 자동화로 휴먼 에러 및 투입 자원 감소
- 빌드 환경의 형상을 소프트웨어 폐기까지 유지
- 오픈소스를 기반으로 구축 가능 및 다양한 구축 사례 공유

## 일반적인 오픈소스 기반 ALM 구성 (Java 기반)



## 2    지속적 통합 - Jenkins



## 지속 통합(CI: Continuous Integration)

---

- ✓ 팀 구성원들이 자신이 한 일을 자주 통합하는 소프트웨어 개발 실천 방법
- ✓ 각 통합은 자동화된 빌드를 검증하여 최대한 빨리 통합 오류를 탐지
- ✓ 하루에 여러 번 통합 빌드를 수행하는 것

- 마틴 파울러

소프트웨어 통합 오류를 개발 초기부터 예방하는 것  
소프트웨어 통합을 위해 현존하는 가장 훌륭한 전략

---

통합 지옥

Integration HELL





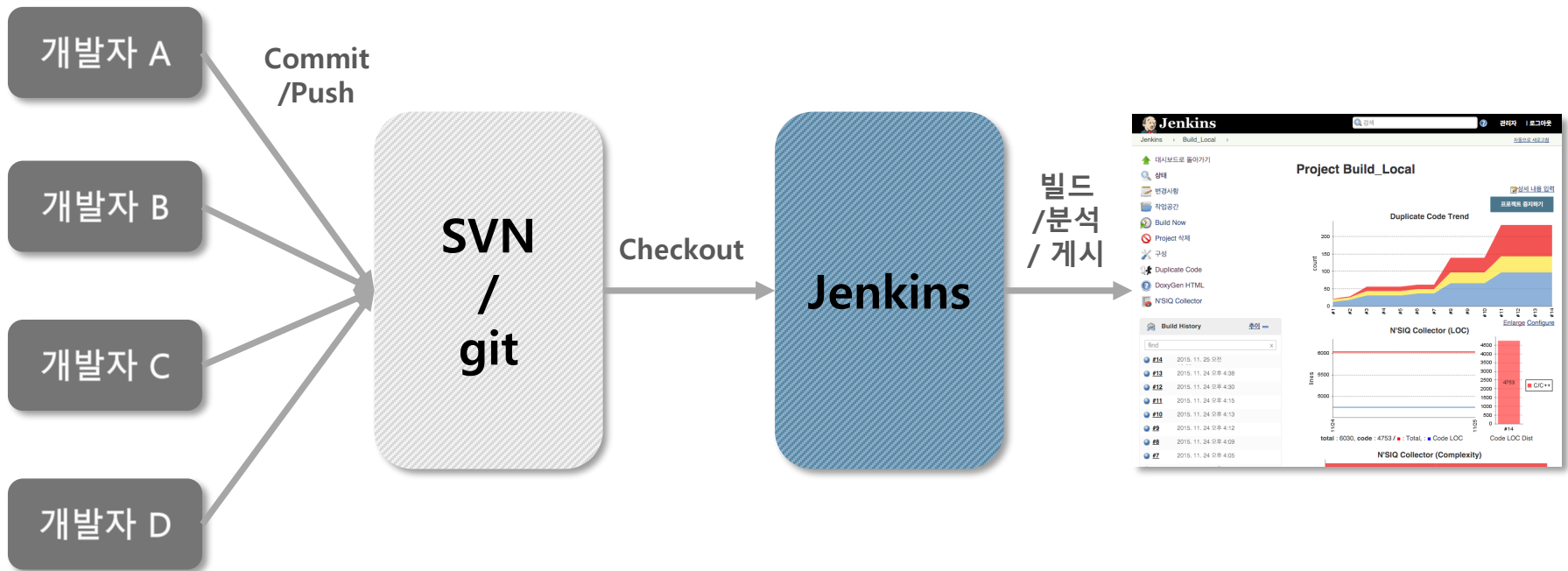
# Jenkins

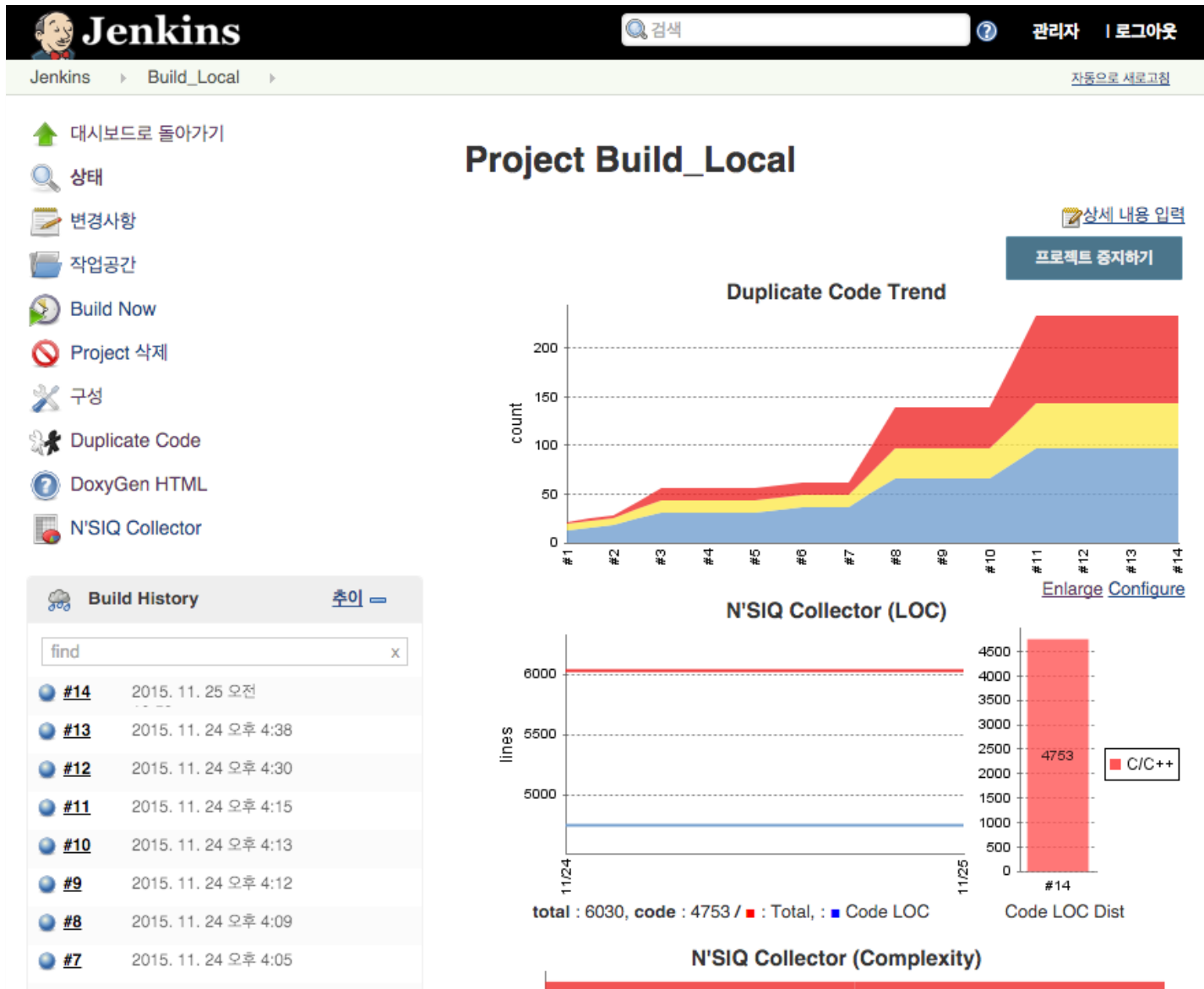
<http://www.jenkins-ci.org>

- ✓ 웹 기반 오픈소스 CI 도구
- ✓ 점유율 약 70%
- ✓ 1,300 개의 플러그인

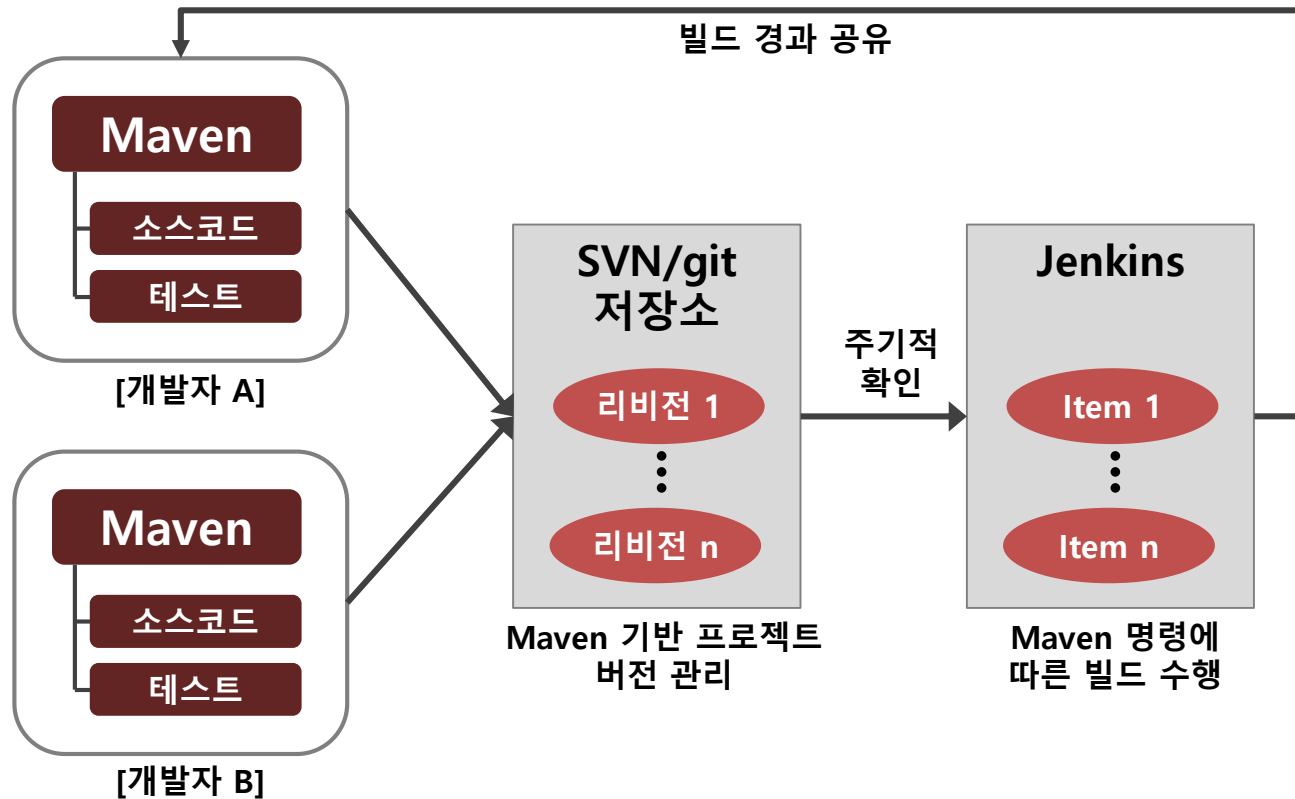
## 지속적 통합 - 동작 방식

- ① 버전관리 도구에서 **최신 리비전을 체크아웃** 받아
- ② 주어진 **명령대로 빌드**하여
- ③ 결과를 **게시/전달**함





## 일반적인 오픈소스 기반 ALM 구성 (Java 기반)



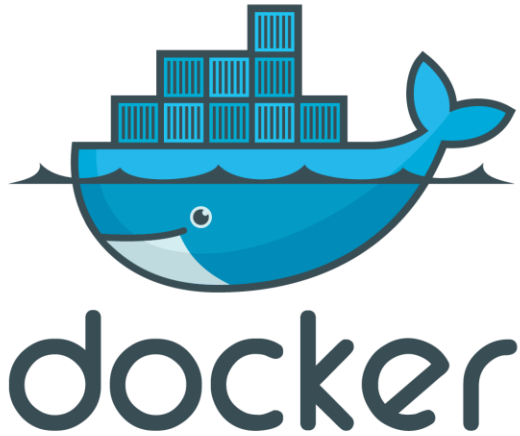
### 3    지속적 통합 - Docker



## Docker 란?

---

- 컨테이너 기반의 오픈소스 가상화 플랫폼

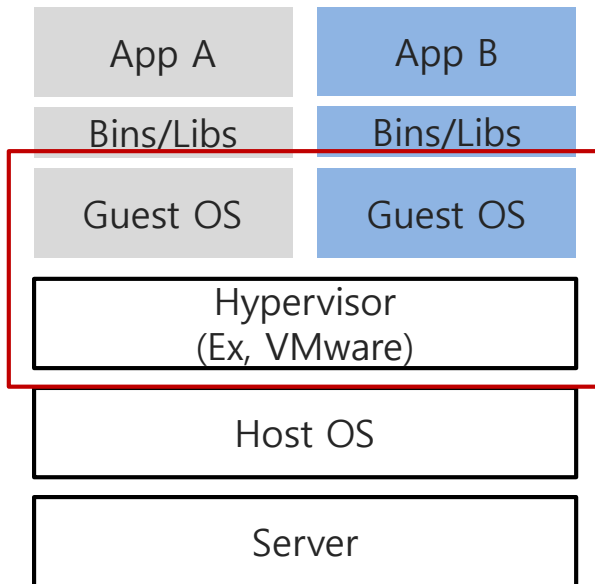
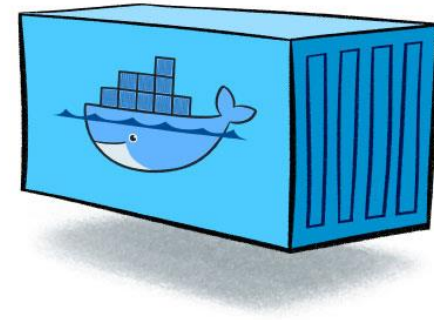


**Develop faster.  
Run anywhere.**

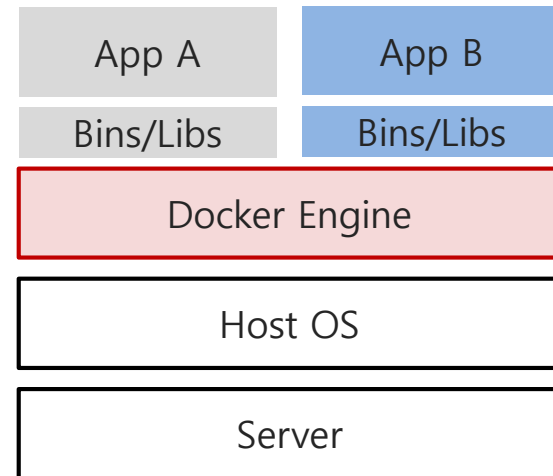
Build with the #1 most-used developer tool

# 컨테이너 (Container)

- 격리된 공간에서 프로세스가 동작하는 기술



기존 가상화 기술



Docker 사용한 가상화 기술

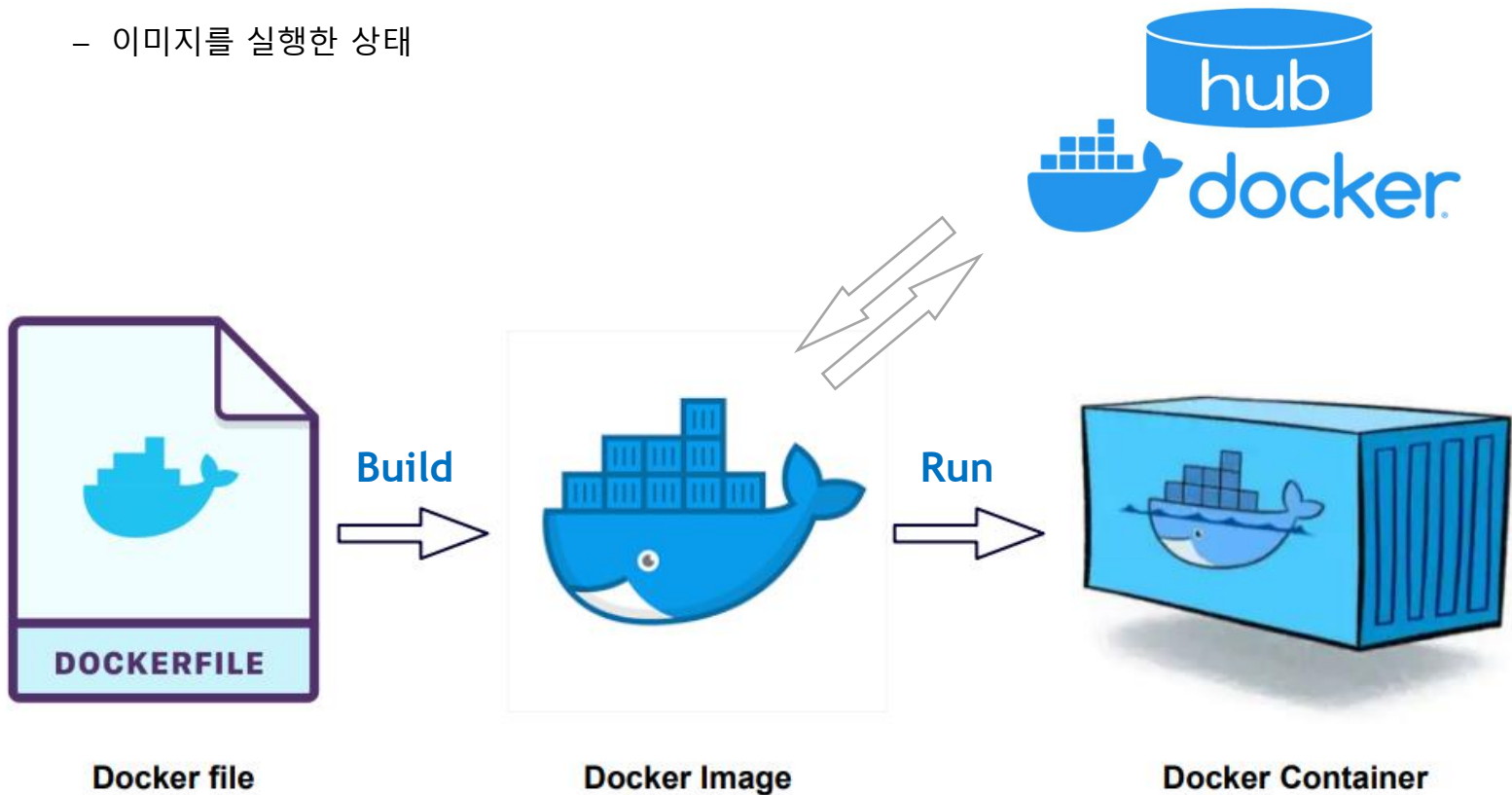
# 도커 엔진의 핵심 기능

## □ 도커 이미지

- 컨테이너 실행에 필요한 파일과 설정 값들을 포함하고 있는 템플릿

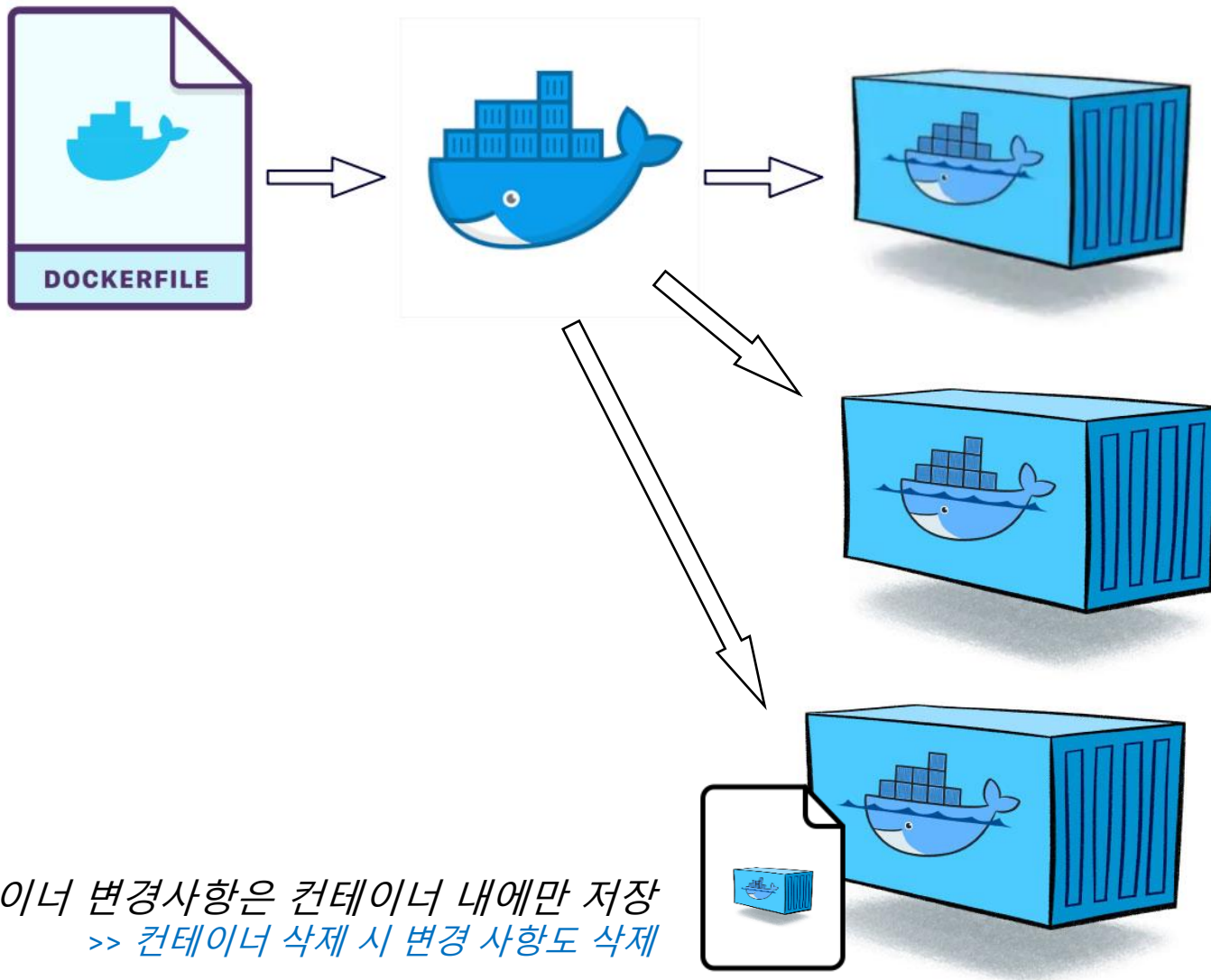
## □ 도커 컨테이너

- 이미지를 실행한 격리된 프로세스 환경
  - 이미지를 실행한 상태





## 도커 엔진의 핵심 기능



컨테이너 변경사항은 컨테이너 내에만 저장  
>> 컨테이너 삭제 시 변경 사항도 삭제

실습



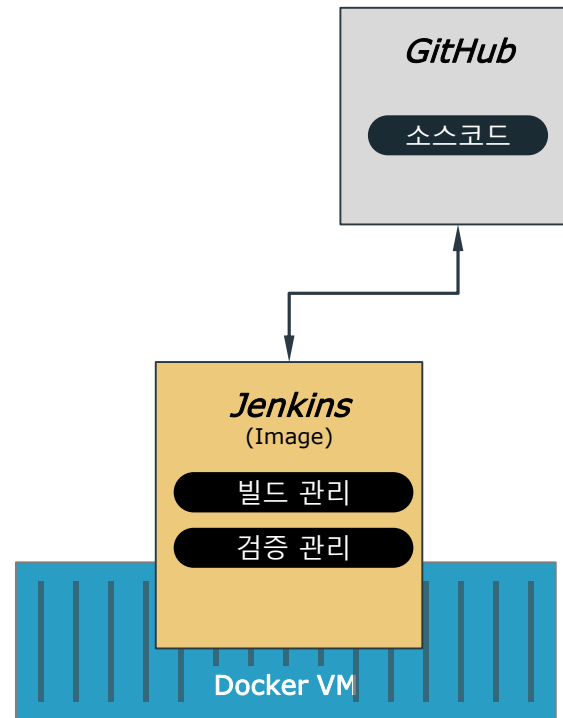
# 다루는 도구

## □ 핵심 도구

- 가상화 : Docker
- 지속적 통합: Jenkins

## □ 지원 도구

- 빌드: Maven
- 소스코드 버전 관리: GitHub 코드



## 실습 순서

---

### ❑ Docker Desktop 설치 (Window)

- Window – WSL2 설치

### ❑ Docker 사용

- 컨테이너 생성
- 이미지 생성 > 컨테이너 생성

### ❑ Docker-compose를 활용한 Jenkins 설치

- docker-compose.yml

### ❑ Jenkins 설정

- Github Token 발급
- Jenkins <> Git 연결
- Maven 설정

### ❑ Jenkins Job 생성

- Job 1 : git clone
- Job 2 : junit4 실행 / 정적 분석 Report