# 실습 4-2/4-3

# -3팀

## 실습 4-2

### 1 Jenkins

-커밋에 따른 자동 빌드 제공

: 소스코드가 형상관리도구에 커밋되었을 때마다 CI(통합 빌드의 과정을 아주 짧은 주기로 수행함으로써 통합에서 발생하는 오류와 수정 시간을 줄이기 위한 기법)도구, 즉 젠킨스의 CI 파이프라인을 통해 감지하고 자동으로 빌드를 수행할 수 있다.

-코드 품질 검사

: 개인이 확인하지 못한 코드 표준 준수 여부의 검사나 정적 분석을 통한 코드 품질 검사를 빌드 내부에서 수행해준다.

-자동 테스팅 코드 기동

: 빌드 과정에 테스트를 포함하며 주기적인 빌드 과정에 테스트를 포함시켜서 빌드와 더불어 테스팅을 자동으로 수행함으로써 소프트웨어의 품질 향상에 기여한다.

-환경 제공

: 여러 개의 프로젝트 또는 파일들의 빌드를 편리하고 쉽게 할 수 있는 환경 제공한다.

-정기적 빌드 수행

: 시간이 오래 걸리는 대규모 빌드나 개발자들의 커밋에 대한 스케줄을 조정하기 위해 일정 시각을 정해두고 정기적으로 빌드를 수행할 수 있다.

-일관성 유지

: CI도구는 형상관리도구로부터 프로젝트 소스의 코드를 받아 빌드를 수행함으로써 소스 코드의 일관성을 유지할 수 있다.

### 2 Github

-깃허브 레포지토리

: 기본적으로 레포지토리(저장소, Repository) 기능을 제공한다. 호스팅 되는 깃 원격 저장소를 통해 로컬 개발 환경과 온라인에서 안전하게 접근할 수 있다. 커밋을 통해 제출한 원격 저장소에

있는 작업물을 다른 팀원이 내려 받아 작업하고 업로드하여 갱신할 수 있으므로 원격 협업이 가능해진다.

-깃허브 액션

: 프로젝트 저장소에서 소프트웨어 개발 워크플로우를 자동화, 사용자 지정 및 실행한다. CI/CD를 포함해 원하는 작업 수행을 위해 작업을 검색, 생성 및 공유하고 사용자 정의된 워크플로우에서 작업을 결합 가능하다.

-깃허브 패키지: 팀과 비공개로 패키지 공유, 오픈 소스 커뮤니티와 공개적인 패키지 공유할 수 있고 공통적으로 사용되는 모듈을 프로젝트 간에 공유 가능하다. 타인의 저장소나 패키지를 포크하여 자신의 코드 베이스를 덧붙여 개발할 수 있으며, 해당 저장소에 기여할 수도 있다.

-이슈 트래커

: 깃허브는 이슈 트래커 기능을 제공한다. 현재 해야 할 일과 진행 중인 일, 끝난 일 등을 구분하여 관리할 수 있으므로 태스크 관리에 효율적이다.

-마켓플레이스

: 마켓플레이스를 제공한다. 기본적인 깃허브 기능과 연동하여 사용 가능한 서드 파티 앱이나 액션을 자유롭게 구매하여 추가할 수 있도록 되어있다.

-깃허브 페이지

: 정적 웹페이지 호스팅 서비스를 제공한다. html, CSS, JavaScript로 구성된 파일을 구동하고 배포할 수 있다. Jekyll 등을 지원한다.

-깃허브 코파일럿

: OpenAI의 GPT-3을 이용하여 깃허브의 레포지토리를 학습시킨 뒤 개발자가 작성한 주석, 코드, 함수명 등을 분석해 의미를 파악하고 해당 기능을 구현하기 위한 코드를 자동 완성하여 단순 작업을 자동화하는 서비스인 copilot을 제공한다. VS Code의 익스텐션 형태로 제공하여 설치 및 사용이 쉽다.

# 실습 4-3

## 1 마이크로서비스를 정의한다.

마이크로서비스는 소프트웨어 아키텍처 디자인 패턴 중 하나로, 하나의 큰 애플리케이션을 작고 독립적인 서비스로 나누는 방식을 말한다. 각 개별 모듈은 개별적인 작업을 담당하며 간단하고 보편적으로 액세스 할 수 있는 API를 통해 다른 모듈과 통신할 수 있다. 독립성, 분산 아키텍처, 확장성, 개별 관리 등을 주요 기능으로 가지며 이런 기능을 활용하여 대규모 시스템을 모듈화하고, 빠른 개발을 통해 여러 요구사항에 빠르게 대응할 수 있는 장점을 지니고 있다.

## 2 마이크로서비스를 이용한 서비스 개발 및 배포의 장점은 무엇인가?

**사례 1) 쿠팡**

: 창립 초기 하나의 단일 아키텍처로 구성되었고 저렴한 비용과 고객 피드백에 빠르게 적응할 수 있다는 장점을 가졌으나 고객 수와 엔지니어 수의 증가로 확장성, 관심 분리 문제로 인해 성장 속도를 높이는 데에 한계를 느끼게 되었다. 또한 한 구성요소의 오류가 다른 구성요소로 이어져 전체 시스템이 중단되는 경우가 증가했다.

따라서 모놀리식 아키텍처에서 마이크로서비스 아키텍처로의 전환을 지원하도록 설계된 로드맵인 비타민 프로젝트를 시작, 구성 관리 데이터베이스를 구축하고 하나의 서비스 오류가 다른 서비스로 연쇄적 발생하는 것을 방지하기 위해 Valve라는 내부 회로 시스템 개발 등을 통해 성공적으로 마이크로서비스 아키텍처로 전환하였다.

이 예시에서 쿠팡이 마이크로서비스를 이용함으로써 얻게 된 장점으로는 서비스가 독립적으로 분리되어 특정 기능의 장애 발생에 대해 연쇄적인 장애 발생을 막고(전체 서비스 확장) 독립적 배포가 가능하기 때문에 특정 서비스에 대한 확장성이 유리하다는 점이다.

**사례 2) 스포티파이**

: 스포티파이는 온라인 음악 시장 속에서 경쟁하기에는 몸집이 작은 편이다. 이런 시장에서 경쟁우위를 점하기 위해서는 경쟁사들보다 빠른 속도를 가져야 한다고 판단한 스포티파이는 서비스 제공 속도를 최적화하는 방법을 택했다. 개발자의 신속한 행동을 가능하게 하기 위해 마이크로서비스 아키텍처를 도입하여 더 작고 상호 연결된 구성 요소로 분해함으로써 소프트웨어의 새 버전을 배포할 때 다른 팀과 조율할 필요가 사라졌다. 이를 통해 음악 라이브러리 업데이트나 사용자 경험 개선과 같은 변경 사항을 효과적으로 구현하고 서비스를 안정적으로 유지할 수 있다.

이 예시에서 스포티파이가 마이크로서비스를 이용함으로써 얻게 된 장점으로는 서비스가 독립적으로 분리되어 소규모의 작은 배포, 서비스별 개발 및 배포가 가능하고 배포 시에 전체 서비스 중단이 없고 속도가 빠르다는 점과 소프트웨어를 수시로 업데이트할 수 있어 고객의 요구를 반영하는데 용이하다는 점이다.

이렇듯 마이크로서비스는 각 서비스가 독립적으로 분리됨으로써 얻을 수 있는 장점이 많다. 서비스 연쇄적 장애 방지, 독립적 개발 및 배포를 이용한 개별 서비스 단위의 수시 업데이트, 속도 향상, 확장성 상승 등이 그 예이다. 따라서, 많은 트래픽을 처리할 수 있고, 속도가 빠르고 오류 없는 서비스가 가능하다는 이러한 마이크로서비스의 장점 때문에 많은 기업들이 모놀리식 아키텍처에서 마이크로서비스 아키텍처로 넘어가는 추세이다.