**오픈소스 과제2 발표내용 AS**

**D팀 (Jenkins)**

**Q1. CI / CD에 대한 개념**

A1. 먼저 CI와 CD에 대해 간단히 설명하면, 개발단계부터 배포까지 모든 단계들을 자동화를 통해 효율적이고 빠르게 배포할 수 있게 만드는 것을 의미합니다.

CI (Continuous Integration)는 단어 그대로 지속적 통합이라는 입니다. 새로운 코드 변경 사항이 정기적으로 빌드 및 테스트 되어 공유 repository에 통합되는 것을 의미합니다.

원래 개발자들은 코드 변경사항을 주기적으로 빈번하게 merge해야합니다. 예를 들어, 여러 명의 개발자가 하나의 소스 파일에 대해 서로 다른 코드를 작성하다가 한참 후에merge하려고 하면 서로 다른 코드를 어떻게 통합해서 적용해 나가야할지 고민하게 됩니다. 이렇게 되면 코드를 작성하는 시간보다 merge 충돌을 해결하는 데에 더 오랜 시간을 쏟는 문제가 발생할 수 있습니다. 이를 해결할 수 있는 것이 CI입니다. CI는 merge된 코드의 변경사항이 자동으로 빌드 되도록 하고, 이를 통해 새로 추가된 코드변경사항 이외에도 빌드가 성공적으로 됐는지 확인합니다. 새로 추가된 변경사항뿐만 아니라 기존시스템의 다른 버그를 초래하지는 않았는지 테스트 됩니다.

개발자들은main repository에 주기적으로 코드의 변경사항에 대한 코드 리뷰를 받고 merge 합니다. 그 이후에는 CI를 통해 자동으로 이 repository가 빌드가 되고, 테스트가 실행됩니다. 그리고 나중에 배포할 때 반영이 됩니다. 이때, 빌드가 실패하거나, 빌드는 성공했지만 test가 실패하면 문제를 일으킨 해당 개발자에게 자동으로 알려주는 기능이 있습니다.

이러한 CI를 사용하면 여러가지 장점이 존재합니다. 주기적으로 merge를 하기에 merge 충돌 피할 수 있습니다. 이를 통해 개발 생산성 향상됩니다. 또한 merge되는 모든 코드들이 자동으로 빌드, 테스트 되기에 코드의 결함이나 문제점을 빠르게 발견할 수 있습니다.

결론적으로, CI를 사용한다면, 버그를 신속하게 찾아 해결하고, 소프트웨어의 품질을 개선하고, 새로운 업데이트의 검증 및 릴리즈의 시간을 단축시킬 수 있다는 장점이 있습니다.

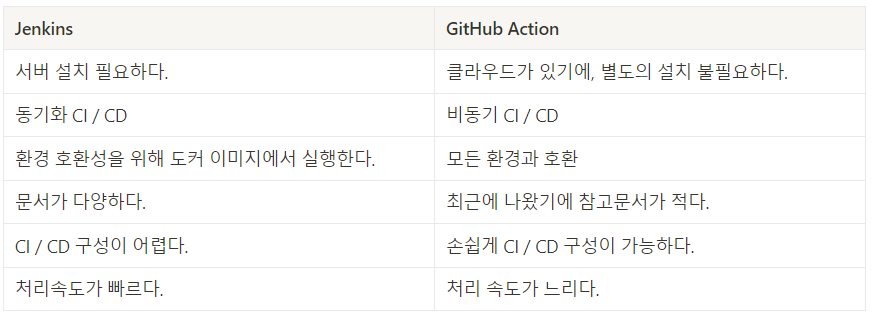
CD (Continuous Delivery / Continuous Distribution)는 지속적 제공, 지속적 배포라는 의미의 단어입니다. 하지만, Continuous Delivery와 Continuous Deployment의 미세한 차이를 예시를 통해 설명하겠습니다. 예를 들어, CI를 통해 주기적으로 merge된 코드의 변경사항들이 자동으로 빌드가 되고 테스트가 되었다면 배포하는 단계에서 배포 준비과정을 거칩니다.

이렇게 준비된 release에 문제가 없는지 개발자나 검증 팀이 검증하고 최종적으로 사용자에게 배포할 결정한 후에 수동적으로 배포하는 단계를 Continuous Delivery라고 합니다. 비슷하지만, 준비된 release를 누군가가 검증하지 않고 자동으로 배포하는 것을 Continuous Deployment라고 합니다.

이렇게, 최종단계를 수동적으로 release할 수도 있고, 자동적으로 release할 수 있습니다. CI CD라 해서 모든 회사가 동일한 프로세스를 거치는 것은 아니고 팀마다 이렇게 다른 방식으로 적용해서 사용합니다.

정리해보면, CI와 CD는 완벽히 분리된 것이 아니어서 묶어서 설명하곤 합니다. 개발자가 작은 단위로 기능을 나누어 주기적으로 repository에 merge하면 자동으로 빌드 되고 테스트를 통해 release 준비를 합니다. 그렇게 최종 배포를 거치게 되는 것입니다.

**Q2. Jenkins와 GitHub Action의 차이점**



**Q2-1. 동기화 CI. CD / 비동기 CI.CD에 대한 추가적인 설명**

A2-1. 프로그래밍에서의 ‘동기’라는 것은 작업이 순차적으로 진행된다는 것을 의미합니다. 하나의 작업이 실행중이라면 완료될 때까지, 다른 작업이 기다려야합니다. 이러한 동기 방식을 사용하게 되면, 다른 작업들이 지연된다는 문제가 발생합니다. 이러한 방식은 순차적으로 실행 되어야하는 작업과 작업 간의 의존성이 높은 경우에 적합합니다.

Jenkins의 서버에서는 빌드, 테스트, 배포 단계가 순차적으로 동기방식으로 진행됩니다. 예를 들어서, 코드 빌드가 완료된 후에만 테스트가 시작됩니다. 이렇게 직렬방식으로 일정한 순서대로 진행되기 때문에 예측이 쉽고 직관적으로 이해하기 쉽다는 장점이 있습니다.

다음은 ‘비동기’입니다. ‘비동기’의 사전적 의미는 ‘동시에 일어나지 않는다.’ 입니다. 즉, 작업이 독립적으로 실행되는 방식입니다. 여러 작업을 동시에 처리할 수 있으므로 더욱 효율적인 방식입니다. 이러한 비동기 방식은 시간이 오래 걸리는 작업을 병렬로 처리할 때 적합합니다.

GitHub Action의 각 작업은 독립적으로 실행되며, 특정 작업의 완료를 기다리지 않고 다음 작업이 실행될 수 있습니다. 예를 들어, 코드 검증과 유닛 테스트를 동시에 실행할 수 있습니다. 이렇게 여러작업들이 동시에 실행될 수 있게 병렬적으로 처리되므로 빌드, 테스트, 배포 시간을 단축할 수 있습니다.

**Q2-2. Jenkins와 GitHub Action의 처리속도 차이**

A2-2. Jenkins와 GitHub Action 처리속도의 주요한 차이점은 실행 환경이 다르다는 것입니다. Jenkins는 로컬 서버에서 실행되므로, 사용자는 하드웨어 및 네트워크를 환경을 제어할 수 있습니다. 로컬 환경에서 실행되면 네트워크 지연이 줄어들고, 하드웨어 자원을 최적화하여 사용할 수 있습니다.

하지만, GitHub Action은 GitHub의 클라우드 인프라에서 실행됩니다. 즉, GitHub가 제공하는 가상환경에서 실행됩니다. 이는, 사용자는 하드웨어 자원에 직접 접근할 수 없고 네트워크 지연 및 자원 할당에 제한이 있을 수 있음을 의미합니다. 또한, 클라우드 기반 실행 환경에서는 여러 사용자가 동일한 인프라를 공유하므로, 특정 시간대에 리소스가 부족할 수 있고, 이는 빌드 및 배포 속도에 영향을 미칠 수 있습니다.

이러한 이유로 GitHub Action의 처리속도에 비해 Jenkins는 처리속도는 빠른 것입니다.