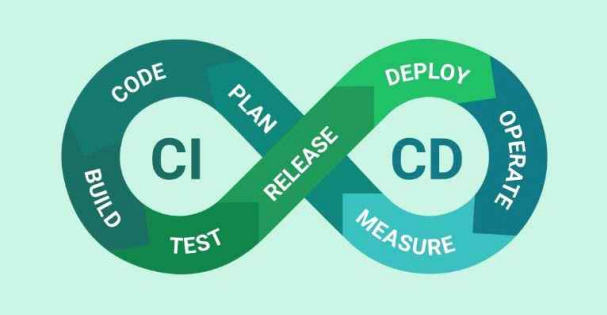
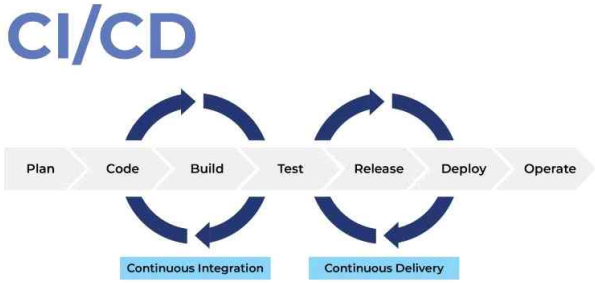
## ■ AWS DevOps를 위한 개발 Toolkits

### ▪ DevOps를 위한 CI/CD 방법론

□ CI/CD



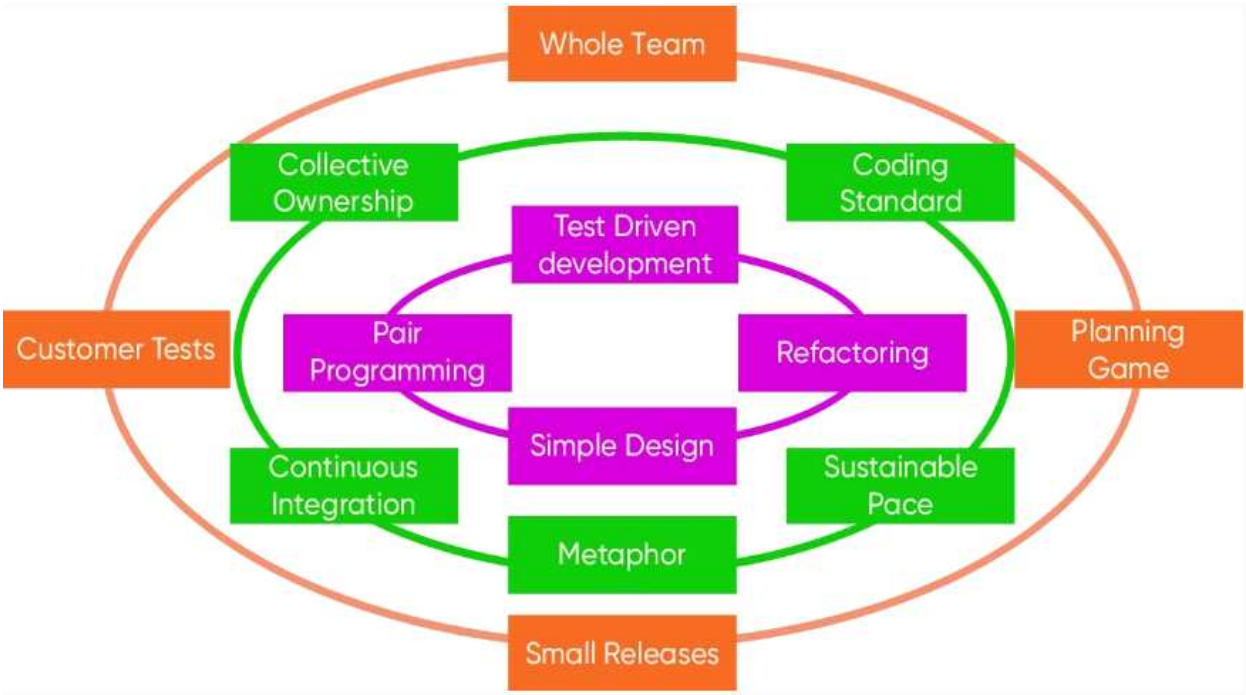

- CI/CD란, Continuous Integration/Continuous Delivery의 약자로 애플리케이션 개발에 필요한 여러 단계에 대한 자동화를 통해 애플리케이션을 보다 빠르고 짧은 주기로 고객에게 제공하는 방법을 말한다.
- CI/CD의 기본적인 개념은 지속적인 통합(Continuous Integration), 지속적인 서비스 제공(Continuous Delivery) 및 지속적인 배포(Continuous Deployment)를 통해 새로운 코드의 통합, 테스트, 릴리스, 배포 등의 애플리케이션 라이프사이클 전체에 대한 자동화 과정을 모니터링 가능하도록 하는 것을 말한다.

- 'CI'는 Continuous Integration의 약자로 지속적인 통합을 의미한다. 개발자를 위한 자동화 프로세스를 통해 새로운 코드 개발과, 코드의 변경 사항이 정기적으로 빌드 및 테스트되고 공유 리포지토리에 병합되어 여러 명의 개발자가 동시에 애플리케이션 개발과 관련된 코드를 작업할 경우에도 서로 충돌없이 원하는 개발 작업을 수행하고 문제를 해결할 수 있다.

- 'CD'는 지속적인 전달(Continuous Delivery) 및 지속적인 배포(Continuous Deployment)의 의미를 가지고 있는데, 두 가지 용어를 혼용하여 사용한다. 개발자가 지속적인 서비스 전달과 배포를 통해 최소한의 노력으로 새로운 코드에 대한 배포를 자동화할 수 있으며, 이로써 신속한 애플리케이션 제공의 속도 저하를 유발하는 수동 프로세스로 인한 운영팀의 프로세스에 대한 과부화를 해결할 수 있도록 도움을 주는 역할을 한다.

□ **CI(Continuous Integration) - 지속적인 통합의 개요**

- CI(Continuous Integration)의 기본적인 컨셉은 1991년 Grady Booch를 통해 처음 소개되었지만, 1999년 켄트 백(Kent Beck)이 창시한 익스트림 프로그래밍(XP, eXtream Programming)의 12가지 핵심 프랙티스(Core Practice)에서 이론과 내용이 소개 되었다. (Agile 프로세스의 대표적 개발 기법)



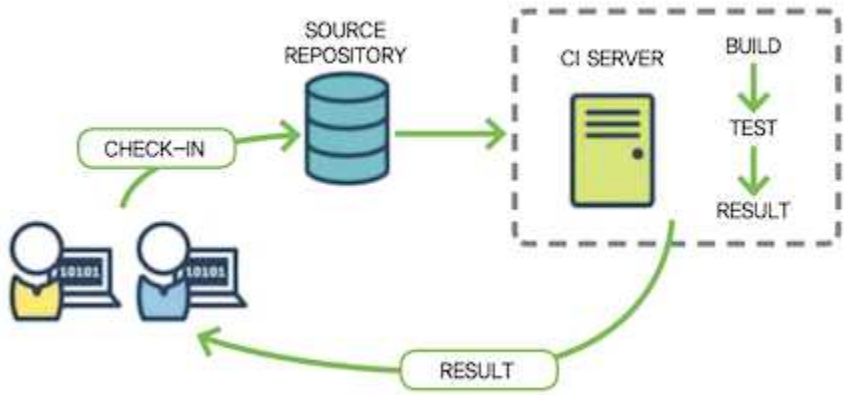

[ eXtheme Programming 12 Practices ]

| 항목 | 설명 |
|---|---|
| Simple Design | 가장 단순하며 정확히 작동하는 디자인 |
| Pair Programming | 두 명의 프로그래머가 하나의 워크스테이션에서 함께 작업하는 소프트웨어 개발 기술이다. 한 명은 코드를 작성하고 다른 한 명은 코드를 검토한다. 이 과정을 번갈아가며 수행한다. |
| Refactoring | 외부 동작을 변경하지 않고 기존 소스 코드를 재구성(더 쉽게)하는 소프트웨어 설계 프로세스이다. 리팩토링은 소프트웨어의 디자인, 구조 및 구현을 개선하는 동시에 기능을 보존한다. |
| Test Driven development | TDD, 테스트 주도 개발을 말하며 실패한 자동화된 단위 수준 테스트 케이스를 작성한 다음 테스트를 통과할 수 있을 만큼만 코드를 작성한 다음 테스트 코드와 프로덕션 코드를 모두 리팩터링한 다음 다른 새 테스트 케이스로 반복하는 코드를 작성하는 방법이다. |
| Coding Standard | 코딩 표준(규칙). 특정 프로그래밍 언어에 대한 일련의 지침으로 해당 언어로 작성된 프로그램의 각 측면에 대한 프로그래밍 스타일, 관행 및 방법을 권장한다. 일반적으로 파일 구성, 들여쓰기, 주석, 선언, 문, 공백, 명명 규칙, 프로그래밍 방법, 프로그래밍 원칙 등을 다룬다. |
| Sustainable Pace | 직역하면 지속 가능한 속도이다. 소프트웨어 개발에서 지속적인 작업을 위해 일정 작업 속도를 유지한다. 긴 근무시간, 초과 근무, 야근, 주말 근무를 하지 않고 프로젝트를 완수하기 위한 개념이다. |
| Metaphor | 공통적인 이름의 체계를 가지고 공통적인 시스템 서술서를 가짐으로써 개발과 의사소통을 돕는다. 문장 형태이며 고객과 개발자간의 의사 소통 언어라고 볼 수 있다. |
| Continuous Integration | 지속적인 통합으로 개발의 불일치를 최소화한다. |
| Collective OwnerShip | 모든 코드는 개발자들이 공동으로 소유하며 누구든지 수정할 수 있음 |

| | |
|---|---|
| Customer Test | 고객이 기능에 대해 수행하는 테스트 |
| Whole Team | 소프트웨어를 제공하는 조직의 모든 것(서비스를 제공하는 직간접적인 참가자 모두를 말한다.) |
| Planning Game | 개발팀과 이해관계자가 함께하는 기획 회의를 말한다. 고객과 개발팀의 모든 개발자가 참여한다. |
| Small Release | 고객이 원하는 기능 중심으로 짧은 시간 내 릴리즈한다. |

- 일반적인 기업에서 애플리케이션을 개발할 때, 여러 개발들이 동일한 애플리케이션에서 각기 다른 신규 기능을 개발하거나, 변경 사항을 수정 변경을 한다.
- 이때 어려운 점은 동일한 애플리케이션에 대해 특정 기능 및 개별적으로 수정/변경된 내용을 반영하는 경우 다른 개발자가 동시에 적용하는 변경 사항과 충돌할 가능성이 있으므로, 이러한 부분을 최소화 할 수 있도록 조정하는 수적업에 많은 시간과 인력이 소모된다.

- CI(Continuous Integration)를 통해 자동화된 프로세스를 구현한다면, 개발자들이 코드 변경 사항을 메인 리포리터리를 통해 업로드를 수행하면 이를 통해 코드를 병합하는 작업을 더욱 수월하게 자주 수행할 수 있다.
- 개발자가 애플리케이션에 적용한 변경 사항이 병합되면 이러한 변경 사항이 다른 개발자가 수정한 내역이나 프로그램에 영향을 주지 않으며, 애플리케이션에 문제가 발생하지 않도록 병합 작업을 수행한다. 그리고 각기 다른 레벨의 자동화 테스트(일반적으로 단위 테스트 및 통합 테스트) 실행을 통해 변경 사항이 애플리케이션에 제대로 적용되었는지 확인한다.




- 다시 말해, 클래스와 기능에서부터 전체 애플리케이션을 구성하는 서로 다른 모듈에 이르기까지 모든 것에 대한 테스트를 수행한다.
- 자동화된 테스트에서 기존 코드와 신규 코드 간의 충돌이 발경되면 CI를 통해 이러한 버그를 더욱 빠르게 확인할 수 있고, 이렇게 확인된 소스를 수정하여 원활하게 배포할 수 있다.

□ **CI(Continuous Integration)의 구성 요소와 기대 효과**

- CI 프로세스를 구현하기 위해 필수적으로 리포지토리 관리, 빌드 자동화 툴, 셀프 테스트, 반영(Commit)을 통한 빌드 툴이 필요하다.

| 구분 | 내용 |
|---|---|
| 리포지토리 관리 | · 소스코드 및 마스터 리포지토리 관리를 위한 버전 관리 솔류션 필요<br>· 다양한 파일의 버전 관리를 통해 소스코드의 개발 내역 관리<br>· 체크아웃 기능을 통해 빌드 가능한 코드 확인 가능<br>· 현재 사용 중인 소스의 버전 관리 및 모든 변경 사항을 리포지토리에 저장<br>· 빌드 서버를 추가로 배치하여 원하는 시점에 코드 변경의 모니터링 수행 |
| 빌드 자동화 | · 한 번의 빌드 실행으로 전체 시스템에 대한 빌드 작업 수행<br>· 팀은 빌드 스크립트 또는 표준화된 빌드 도구를 활용하여 빌드 수행<br>· 빌드 자동화에는 소스코드 통합, 코드 컴파일, 유닛 및 통합 테스트, 프로덕션과 같은 환경 배포 포함<br>· 버전 제어 시스템에서도 이전 빌드의 백업본 유지<br>· 빌드 중 오류 발생 시 이전 버전으로 복구 수행 및 소스코드 비교를 통해 원인 분석 작업 수행 |
| 셀프 테스트<br>(Self-Test) | · 오류/버그의 신속하고 효과적인 빌드를 위해 테스트 자동화 포함 필요<br>· 자동화된 테스트 시스템을 통해 실패한 테스트 정보 확인 및 대응 수행<br>· 자체 테스트 실패 시 전체 테스트도 실패로 처리 |
| 반영(Commit)을 통한 빌드 | · 소스 반영(Commit)의 정기적인 수행 시 개발자 간 이슈의 최소화 가능<br>· 1일 1회 이상의 소스 변경 반영(Commit) 권장<br>· 테스트 상황과 애플리케이션의 상태 확인 가능<br>· 팀은 빈번한 반영(Commit)을 통해 더 많은 테스트된 빌드 취득<br>· 메인 저장소의 소스 상태 확인 및 유지 체크 가능 |

- CI(Continuous Integration)를 통해 다음과 같은 효과를 기대할 수 있다.

| 구분 | 내용 |
|---|---|
| 개발자<br>생산성 향상 | · 개발자의 수동 작업에 대한 부담 감소<br>· 고객에게 제공되는 오류 및 버그 수를 줄이는데 도움이 되는 도구 활용 가능<br>· 팀의 생산성 향상 기대 |
| 버그를 더 빠르게 발견하고 해결 | · 테스트를 보다 빈번하게 수행할 수 있음<br>· 버그가 더 큰 문제로 발전하기 전 조기 해결 가능 |
| 업데이트를 빠르게 제공 | · 팀이 좀더 빠르고 빈번하게 고객에게 업데이트 제공 가능 |

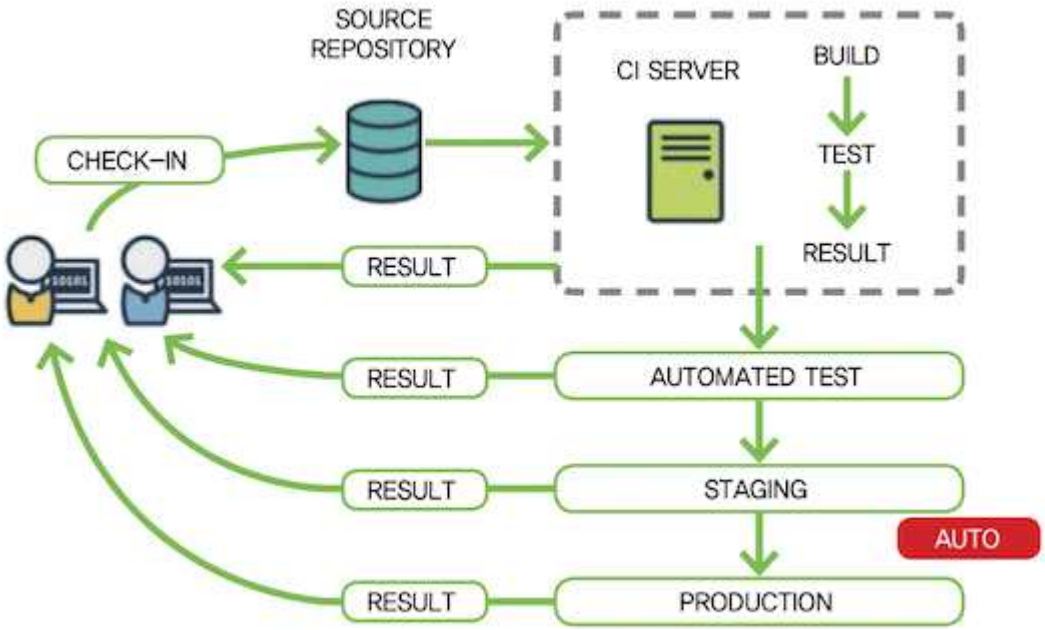
□ **CD(Continuous Delivery/Continuous Deployment) - 지속적인 전달 및 배포**

- CI(Continuous Integration)를 통한 빌드의 자동화 및 유닛의 통합 테스트 수행 이후 이어지는 CD(Continuous Delivery) 프로세스는 유효한 소스코드를 리포리토리에 자동으로 전달한다.
- 따라서 효과적인 CD의 프로세스를 실현하기 위해서는 개발 파이프라인에 CI가 먼저 구축되어야 한다.
- 이러한 지속적 제공의 목표는 운영 환경으로 배포할 준비가 되어 있는 코드베이스를 확보하는 것에 있다.

- CD는 소스코드의 변경 사항을 병합하고, 운영 환경에 배포 가능한 적합한 빌드 제공에 필요한 테스트 자동화, 소스 배포의 자동화가 포함된다.

- 이러한 CD 프로세스를 수행하면 운영팀이 보다 빠르고 손쉽게 애플리케이션을 운영 환경으로 배포할 수 있다.

- 지속적 배포인 Continuous Deployment는 CI/CD 파이프라인의 마지막 단계로, 운영 환경으로 배포 준비가 완료된 빌드를 리포지토리에 자동으로 릴리스하는 Continuous Delivery의 확장된 형태인 Continuous Deployment는 애플리케이션을 운영 환경으로 릴리스하는 작업을 자동화한다.
- 운영 환경으로 빌드를 배포하기 이전의 파이프라인 단계에는 수작업이 없기 때문에 지속적인 배포는 자동화된 테스트에 많은 부분 의존하게 된다.




- 지속적인 배포를 위해 개발자가 애플리케이션에 변경 사항을 작성한 후 몇 분 이내 애플리케이션을 자동으로 실행게 된다. 이를 통해 사용자의 피드백을 지속적으로 확인하고 통합함으로써 애플리케이션의 배포로 인한 서비스의 안정성 ㅁ저하의 문제를 해결할 수 있다.
- 애플리케이션의 밴경 사항을 한번에 모두 적용하지 않고 작은 조각으로 나누어 적용함으로써 빠르게 변화하는 시장 환경에 효과적으로 대응할 수 있는 비즈니스 경쟁력을 제공한다.

□ **CD(Continuous Delivery/Continuous Deployment)의 프로세스와 기대 효과**

- Continuous Delivery/Continuous Deployment 프로세스는 다음과 같은 단계로 수행될 수 있다.

| 구분 | 구성요소 | 내용 |
|---|---|---|
| Continuous Delivery | 빌드 자동화 | · 운영 환경 배포에 필요한 빌드는 자동화 도구 활용 |
| | Continuous Integration | · 개발자가 공유 리포지토리에 코드 체크인<br>· 빌드 자동화 도구가 체크인을 확인하고, 오류, 버그, 발급자 정보를 확인하며, 팀에게 진행 결과에 대한 Report 수행<br>· 반복적이며, 지속적인 S/W의 Delivery 수행 |
| | 테스트 자동화 | · 새로운 버전의 응용 프로그램 품질과 기능에 대한 점검<br>· 파이프라인을 통한 보안, 성능, 컴플라이언스 검증<br>· 다양한 자동화된 활동 수행 |
| | 배포 자동화 | · 고객에게 새로운 기능을 수분 이내에 신뢰성 있게 제공 |
| Continuous Deployment | 리포지토리 체크인 | · 개발 작업 완료 후 리포지토리에 체크인 수행 |
| | 통합 테스트 | · CI 서버에서 변경 사항을 선택하여 소스 병합<br>· Unit 테스트 결과를 통해 스테이징 환경에 반영 후 QA 테스트 수행 |
| | 운영 환경 배포 | · QA 테스트 통과 시 빌드가 운영 환경으로 전달<br>· CI 서버에서 운영 환경에 빌드 병합 여부 최종 확정 및 배포 |

- Continuous Delivery/Continuous Deployment를 통해 다음과 같은 효과를 기대할 수 있다.

| 구분 | 구성요소 | 내용 |
|---|---|---|
| Continuous Delivery | 배포 위험 최소화 | · 변경 사항을 작은 조각으로 배포함으로써 배포 위험의 최소화<br>· 문제 발생 시 빠르고 손쉽게 복원 가능 |
| | 프로세스 모니터링 | · 진행 절차나 프로세스 모니터링 가능<br>· 운영 환경과 동일 환경에서 전체 프로세스 사전 점검 가능 |
| | 사용자 피드백 | · 작게 구성하여 빠르게 사용자 피드백 확보<br>· 실사용자로부터 애플리케이션의 유용성에 대한 빠른 피드백 |
| Continuous Deployment | 생산성 및 품질향상 | · 제품에 대한 집중도가 높아지며, 업무 효율 및 생성성 향상<br>· 테스트에 집중하고, 반복 작업 자동화를 통한 품질 향상 |
| | 업무 프로세스 개선 | · 통합 파이프라인을 통해 팀과 프로세스 간 통합<br>· 개발, 테스트 및 운영 환경 전반에서 워크플로우 생성<br>· 기존 툴과의 연동을 통해 간단하게 작업 수행 가능 |

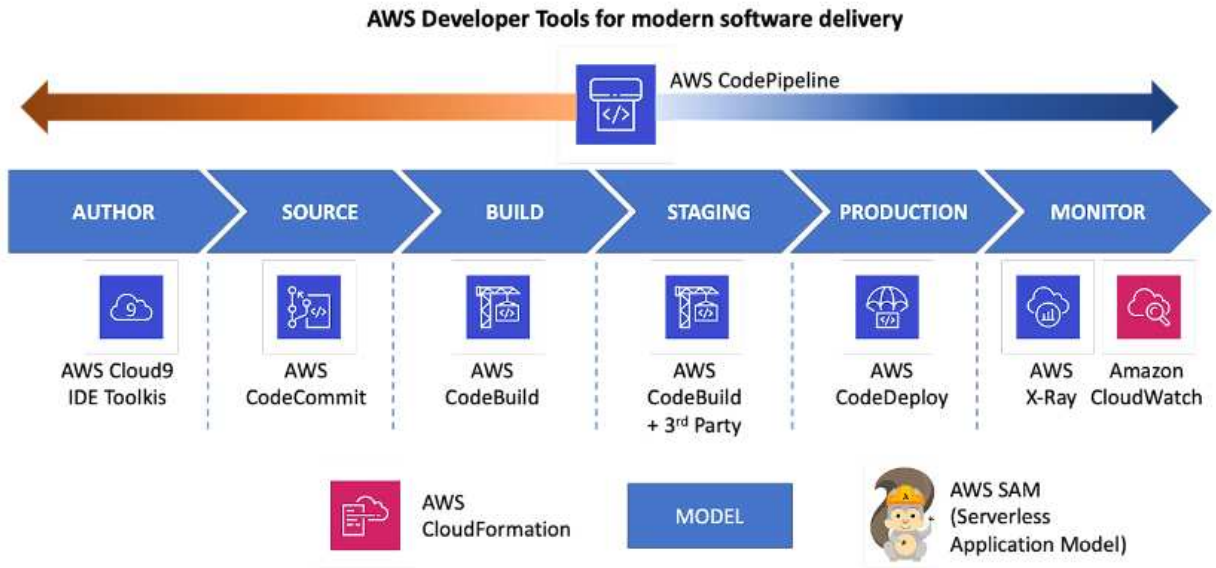
□ CI/CD를 위한 도구들

- DevOps 수행을 위해 다양한 도구를 활용할 수 있다.
- DevOps를 위한 활용 도구는 소스코드 관리, 구성 관리, 시스템 구축 및 테스트, 통합, 애플리케이션 배포, 버전 제어 및 지속적인 모니터링을 위한 도구들이 포함된다.
- Continuous Integration, Continuous Delivery 및 Continuous Deployment에는 서로 다른 도구가 필요하다. 애플리케이션의 종류와 요구사항에 따라 적합한 도구를 선정하여 사용할 수 있다.

| 구분 | 내용 |
|---|---|
| 소스코드 관리 | GIT, Bitbucket, Subversion |
| 빌드 자동화 도구 | Maven, Ant, Gradle |
| 테스트 자동화 도구 | Selenium, JUnit, Cucumber |
| CI 도구 | Jenkins, Bamboo, Hudson |
| 구성 관리 도구 | Puppet, Chef, Ansible |
| 모니터링 도구 | Nagios, Ganglia, Sensu |

▫ 성공적인 DevOps를 위한 AWS의 도구들

- AWS는 성공적인 DevOps 수행을 위해 필요한 매우 강력한 소프트웨어와 인프라 환경을 제공한다.
- 이를 통해 소프트웨어와 제품을 안정적이고 신속하게 구축할 수 있는 일련의 유연한 서비스를 사용하여 이를 제공한다.




- 애플리케이션 코드 구현을 단순화하고, 릴리스 프로세스 및 프로비저닝을 자동화하며, 인프라 관리 및 애플리케이션 모니터링 및 인프라 성능 관리 등의 수작업을 클라우드 서비스와 IaC(Infra as a Code)를 통한 인프라 관리 자동화 도구를 활용하여, 규모에 관계없이 복잡한 환경을 관리하는 데 도움이 된다.

- AWS 개발자 도구는 Cloud9, CodeCommit, CodeBuild, CodeDeploy, CodePipeline, CodeStar 및 AWS X-Ray 와 같은 서비스들을 제공한다.

| AWS 서비스명 | 솔루션 | 내용 |
|---|---|---|
| AWS CodeCommit | 버전 관리 | 소스코드를 개인 Git 저장소에 안전하게 저장 |
| AWS CodeBuild | CI/CD | 연속적인 스케일링으로 코드 빌드 및 테스트에 사용 |
| AWS CodeDeploy | CI/CD | 소스코드의 자동 배포 |
| AWS CodePipeline | CI/CD | 지속적인 통합/지속적인 전달(CI/CD) 서비스 |
| AWS CodeStar | CI/CD | 템플릿 기반의 신속한 애플리케이션 개발, 구축 및 배포 |
| AWS X-Ray | 모니터링 | 애플리케이션의 디버그 및 분석, 모니터링 수행 |
| AWS Command line Interface | 명령 인터페이스 | Command를 기반으로 AWS 리소스 관리 |
| AWS Cloud9 | 개발 툴 | 웹 브라우저를 기반으로 Could IDE를 활용한 개발 및 실행, 디버그를 위한 개발 툴 |