2024 학년도 1학기 중간과제물(온라인 제출용)

|  |  |  |  |
| --- | --- | --- | --- |
|  | **교과목명** | : | 소프트웨어공학 |
|  | **학번** | : | 202034-153746 |
|  | **성명** | : | 이동열 |
|  | **연락처** | : | 010-5264-5565 |

EMB000023580f65

1.

DevOps는 개발(Development)과 운영(Operations)의 합성어로 개발과 운영을 통합하는 방법론, 문화, 도구 등을 모두 일컫는 개념이다. DevOps의 탄생 배경을 살펴보면, 전통적인 IT 조직구조는 개발조직과 운영조직이 명확하게 분리되어 있어 서로의 목적이 상충하는 것에서부터 시작한다. 개발조직은 요구사항을 받아 최대한 빠르게 어플리케이션을 릴리즈하는 것을 목표로 하는 반면에 운영팀은 최대한 시스템을 안정적으로 유지하는 것을 목표로 하기 때문에 릴리즈 검토에 최대한 많은 시간을 사용하면서 배포에 보수적인 입장을 취하게 된다. 이렇다보니 개발 및 운영조직은 잦은 갈등을 겪게 되고 이는 결국 생선성, 안정성 저하를 가져오게 되며, 이를 해결하기 위해 생겨난 개념이 DevOps이다.

현대에는 스마트폰 및 PC를 통한 다양한 IT 서비스들이 탄생하면서 기업이 살아남기 위한 필수적인 조건 중 하나로 변화에 대한 수용력을 꼽고 있다. DevOps는 개발부터 배포까지의 프로세스를 자동화하고 테스트를 통해 리스크를 배포에 대한 리스크를 최소화해 개발조직이 개발에만 집중할 수 있도록 하고 변화 수용력을 키우는데 중요한 역할을 하고 있다.

DevOps의 도입하기 위한 핵심요소로 크게 도구, 소프트웨어 아키텍처, 개발 방법론, 조직 문화를 꼽을 수 있다. 개발 방법론에 대한 부분부터 보면 기존 프로그램 개발 방식은 조직이 변화에 대한 수용력이 크지 않았기 때문에 대규모 업데이트를 계획된 날짜에 배포하는 워터풀방식의 프로세스가 주를 이뤘다. 이런 워터풀 방식의 개발 방법론으로는 다른 도구를 도입하거나 아키텍처를 변경하더라도 충분한 효과를 보기 힘들기 때문에 애자일과 같이 민첩성을 강점으로 하는 개발 방법론을 채택하는 것이 필수요소가 되었다. 애자일은 빠르게 변화하는 IT 업계에서 고객의 요구사항을 신속하게 반영하기 위해 민첩성에 중점을 둔 개발 방법론이다. 이 개발 방법론은 작은 단위의 프로젝트를 반복적으로 처리해 릴리즈 주기를 짧게 가져가며 고객의 요구사항이나 피드백을 더 빠르게 반영할 수 있기 때문에 변화 수용력을 높여주는 역할을 한다.

문화에 대한 부분을 보면 DevOps는 개발팀과 운영팀을 통합하는 과정인 만큼 개발 문화에 대한 부분도 많은 변화가 필요해진다. 기존 개발팀은 팀의 전략에 따라 DevOps 도구를 사용해 개발, 테스트, 배포, 운영 등 더 넓은 범위의 업무 맡게 될 수도 있고 운영팀은 아키텍처 설계, 운영 정책, 프로세스 수립, 사이트 신뢰성 엔지니어링 등의 업무를 맡게 될 수도 있다. 이렇게 변화하는 조직에서 DevOps를 성공적으로 도입하기 위해서는 각 팀간의 커뮤니케이션이나 역할 분담, 적극적인 피드백, 유연한 의사결정 등 문화적인 부분의 개선이 뒷받침 되어야 한다.

DevOps 도입시 기획부터 개발, 배포, 운영, 피드백까지 프로젝트 라이프사이클 내에서 다양한 도구들을 활용할 수 있다.

프로젝트 초기에는 부서간 커뮤니케이션, 정보 공유, 프로젝트 및 테스크 관리 등이 중요하다. 때문에 프로젝트 관리위한 Jira 같은 도구나 정보 공유, 커뮤니케이션을 위한 위키 등의 도구를 활용할 수 있다.

개발단에서는 지속적 통합 및 전달을 위해 Jenkins, Github Actions와 같은 지속적 통합/전달 도구를 활용할 수 있다. 이런 도구들은 형상 관리 툴과 연계하여 이벤트 기반(또는 수동)으로 소프트웨어의 통합을 자동화하고 통합 과정을 가시화해주며 통합된 소프트웨어를 배포 전략에 맞춰 안전하게 배포해주는 역할을 때문에 개발단에서는 핵심적인 도구로 꼽힌 다. 이 외에도 지속적 통합 시 소프트웨어의 신뢰성이나 코드 품질을 유지할 수 있도록 정적 코드 분석 도구를 활용하거나 배포시 모니터링 툴을 결합해 배포중이나 배포후에 소프트웨어가 문제가 없는지 확인할 수 있다.

운영팀에서는 소프트웨어의 신뢰성을 위해 Observability를 확보해야 한다. 개발된 소프트웨어 특성에 맞게 지표를 설정하고 SLO와 SLA를 관리해야하거나 에러예산을 계산해야하기 때문에 각종 로깅, 모니터링, APM 등의 도구를 활용할 수 있다.

인프라 레벨에서도 DevOps 도구를 활용할 수 있다. 컨테이너를 오케스트레이션 도구인 쿠버네티스나 인프라 코드로 관리할 수 있는 Terraform, 서버 구성 관리 도구인 Ansible 등이 있다.

DevOps는 아키텍처와 깊게 연관되어 있다. 전통작인 모놀리식 아키텍처는 서비스 규모와 코드베이스 규모가 비례하기 때문에 컴파일, 빌드, 테스트 속도가 느려지고 에러가 발생하기 쉬워 DevOps를 도입하더라도 요구사항에 빠르게 대처할 수 없어진다. 때문에 서비스 규모가 어느 정도 커지면 어플리케이션 수준의 아키텍처를 재설계하거나 인프라 수준에서 아키텍처를 재설계하는 경우가 일반적이다.

서비스 규모가 커지면서 채택되는 아키텍처 중 가장 DevOps에 적합한 아키텍처가 있는데 바로 마이크로 서비스 아키텍처이다. 마이크로 서비스 아키텍처는 서비스를 동작 가능한 가장 작은 단위로 쪼개 구축하는 방식이다. 이렇게 서비스를 직은 단위로 쪼개놓으면 각 서비스 규모가 작아져 요구사항이나 장애에 신속하게 대응이 가능하며 이는 DevOps 핵심인 변화 수용력을 최대화시켜준다. 하지만 마이크로 서비스 아키텍처는 서비스간 팀이 다른 경우가 많고 통합 테스트 수행이 어려우며 장애 지점을 찾기 어려워 높은 수준의 DevOps가 필수적으로 요구되는 아키텍처이다.

이렇게 보면 DevOps가 무조건 적으로 좋아보일 수 있지만 분명하게 장단점이 존재한다. DevOps의 가장 큰 장점은 요구사항에 빠르게 대응하면서 확장성, 안정성, 보안 향상, 비용 최적화를 달성할 수 있다는 점입니다. 하지만 이 정도 수준의 DevOps를 달성하려면 조직 전체에서 DevOps 필요성에 대해 공감해야하고 적극적인 지원을 해줘야한다는 단점도 있습니다. 만약 조직 차원의 지원이 없다면 무리한 DevOps 도입이 조직에 더 큰 혼란을 가져올 수도 있습니다.

[참고 문헌]

* [What is DevOps Culture? – Atlassian](https://www.atlassian.com/devops/what-is-devops/devops-culture)

2.

(1)

임계 경로 프로젝트 중 가장 긴 시간이 걸리는 작업 경로를 말한다. CPM에서 전체 경로를 구하면 다음과 같다.

* A > B > D > G > I = 13
* A > B > D > H > I = 12
* A > B > E > H > I = 14
* A > C > E > H > I = 15
* A > C > F > I = 12

이 중 가장 긴 시간이 걸리는 작업 경로는 총 15인 A > C > E > H > I로 이 경로가 임계경로가 된다.

(2)

작업 F가 가장 빠르게 시작하려면 A > C > F 순서로 대기 시간없이 수행하면 된다. 따라서 가장 빠르게 시작할 수 있는 시간은 A(2) > C(4)를 합한 시간인 6이 된다.

임계 시간을 초과하지 않는 선에서 가장 늦게 시작하려면 I 작업이 시작되는 시간에 맞춰 F 작업이 종료되면 된다. I의 시작 시간은 A(2) > C(4) > E(6) > H(2)로 해서 총 14이며 14에 맞춰 F 작업을 완료하려면 9에 F작업을 시작하면 된다.

* 가장 빨리 시작할 수 있는 시간 : 6
* 가장 늦게 시작할 수 있는 시간 : 9