1. 안녕하세요 소프트웨어 마에스트로 13기. 중간평가 발표를 맡은 알락꼬리여우원숭이 팀장 류동인이라고 합니다.
2. 먼저 프로젝트에 대한 간단한 개요입니다. 우리 프로젝트는 개발팀의 생산성을 보여주는 여러 지표를 도식화, 개발 팀에서 발생할 수 있는 여러 병목 요인을 파악하여 해결할 수 있도록 도와주는 모니터링 툴 제작 프로젝트입니다. 프로젝트는 Google DevOps Research & Assessment Team에서 제안한 4가지의 Metric과 생산성 관련 지표들을 시각화해주는 웹 서비스, 그리고 개발 중 발생할 수 있는 여러 상황에 대한 위험 관리 알람으로 이루어집니다.
3. 목차입니다. 팀과 아이템에 대한 간단한 소개 이후 피드백을 통한 고찰을 바탕으로 지금까지 있었던 기획 및 개발의 결과물을 소개할 수 있도록 하겠습니다.
4. 저희 팀 소개와 아이템 소개입니다.
5. 저희는 알락꼬리여우원숭이입니다. 항상 더 나은 방식을 고민하고, 조금만 더 라는 생각 하나만으로 하루하루 개발을 해 나아가고 있습니다.
6. 우리 프로젝트의 기반 개념이 되는 Four Key Metrics에 대해 자세하게 설명 드리겠습니다. DORA Metrics라고도 불리는 이 지표는 Google DevOps Research and Assessment Team의 2017년 Report에서 시작되어 발전했습니다. 그리고 이는 점차 발전하여 현재는 북미 개발 시장에서 New Normal이 되었습니다.
7. 각 지표를 설명드리겠습니다. 첫번째 지표인 Deployment frequency는 개발 팀 내에서 얼마나 Product가 자주 변화하는지에 대한 지표입니다. 이는 DevOps의 핵심 가치인 지속가능한 소규모의 빈번한 개발을 나타냅니다.
   1. DORA 팀의 레포트에 의하면, 소규모 배포는 코드를 더 안전하게 테스트 및 배포할 수 있게 합니다. 또한 지속적이고 빈번한 전달은 배포 및 버전 제어의 올바른 사용과 양의 상관관계를 가집니다.
   2. 결국, 이상적인 상황에서 빈번한 배포는 배포 후 오류를 줄이며, 오류 시 더 빠른 복구와 강한 연관성을 가진다는 결과입니다. DevOps를 사용하는 개발 팀들은 이를 이루기 위해 배포할 Task의 단위 및 기능을 작게 쪼개는 등의 노력을 기울여야 합니다.
8. 두번째는 Lead Time for change입니다. 이 지표는 Production의 변경이 필요할 때, 팀이 Production 환경에서 성공적으로 실행되는 코드를 만들어 내는데까지 걸리는 시간을 의미합니다.
   1. 코드를 작성하는 시간은 물론, 리뷰 및 테스트 시간, 그리고 배포까지 걸리는 시간으로 개발과 관련된 팀 구성원 모두의 시간을 측정하게 됩니다. 문제가 있을 시에는 첫 commit부터 배포까지의 요소들의 효율성을 검사하여 배포까지 걸리는 시간을 최적화 하는 것이 중요합니다.
9. 다음은 change failure rate입니다. 이는 Product의 실패를 유발하는 배포의 비율입니다. 서비스 경험을 저해하는 Release에 포함된 변경사항, 그리고 수정이 필요한 변경 사항을 포함합니다.
   1. 이 지표는 Code Review 및 QA, 그리고 코드 작성 부분에서 얼마나 좋은 퀄리티의 작업을 진행하고 있는지를 나타낸다고 볼 수 있습니다. Production의 안정성을 고려하는 팀에서는 특히 이 부분을 집중해야 할 것입니다.
10. 마지막 Time To Restore Service입니다. Mean Time To Recover이라고도 불리며, 서비스가 장애로부터 복구하는데 평균적으로 걸리는 시간을 나타냅니다.
    1. 이는 개발 팀의 사고 대응 프로세스의 최적화 정도를 알 수 있는 지표로 볼 수 있으며, 이를 낮추기 위해서 많은 노력을 기울여야 할 것입니다.
    2. 하지만 MTTR에만 집중하게 된다면, 전반적인 코드 품질을 낮춰 CFR을 높일 수 있다는 맹점이 있습니다.
11. 아마 여기 계시는 심사위원분들께서는 이러한 Four key Metrics를 들어본 적이 없을 가능성이 높을 것이라고 생각합니다. 그리고 저희 또한 이 점을 이상하게 생각했습니다.
    1. IT 기술 트랜드 분석 보고서 Technology radar는 추천 트랜드로 Four Key Metrics를 제시하고 있습니다. 그리고 이를 전문으로 서비스하는 기업 또한 많이 나온 상태이며, 이를 사용하는 기업은 Meta와 Aspire와 같은 거대기업부터 다양합니다. 하지만, 우리는 왜 이러한 변화에 대해 알지 못하였을까요?
12. 우리는 한국 개발 조직과 북미 개발 조직의 다름에 주목했습니다.
    1. 한국의 개발 조직은 아직 Code Review가 타인에 대한 직접적인 평가로 인식되어 적극적으로 활용되지 않는 기업들이 많고, 생산성 측정이 Feedback이 아닌 Blame의 역할을 하게 될 위험성 또한 많은 조직 문화를 가지고 있습니다. 그리고 이것을 원하는 개발자는 아마 없을 것입니다.
    2. 또한, 개발 환경에서 주로 사용되는 응용 프로그램들이 북미의 상품들이 연동 주체로써 내세우고 있는 서비스들과 거리가 있음을 발견하였습니다.
13. 두번째 원인으로는 홍보 및 절대적 인지도 부족에서 찾았습니다. Four Key Metrics는 2017년 처음으로 소개되었으며, 매년 후속 연구를 진행 한 끝에 2022년이 되어서야 Main 기술 트랜드로 자리 잡게 되었습니다. 하지만 아직 한국에서는 적극적으로 소개된 적이 없어 낮은 인지도를 보일 수밖에 없습니다.
14. 세번째 원인은 비싼 도입 비용이라 생각했습니다. 해외의 Four Key Metrics 솔루션은 평균적으로 팀 소속 개발자 1인당 월 $20~$40달러나 되는 높은 비용을 요구합니다.
    1. 이는 아직 Four Key Metrics에 대한 인지가 부족한 한국에서 투자하기에는 부담이 될 수밖에 없다는 판단을 내렸습니다.
15. 마지막 원인은 너무 많은 초기 설정과 정보의 객관성에 대한 의문에서 찾았습니다. 기존 서비스들의 경우 모든 지표를 얻어내기 위해서 정해진 형상관리 툴부터 시작해 이슈 관리 툴까지 모두 연결해야 합니다.
    1. 만약 하나라도 연결되지 않는다면 특정 지표를 보여주지 않는 등, 특정 서비스의 집합을 강요하는 모양세를 띄고 있습니다. 그리고, 이러한 채널들에서 얻은 데이터가 과연 정확하게 이어질까? 에 대한 의문 또한 생겼고, Demo를 통해서 확인해본 결과 아니라는 결과가 나왔습니다.
16. 이러한 문제들을 파악한 후 저희는 다음과 같은 해결책을 도출했습니다. 첫번째, 최소한의 이벤트 발생 요소인 형상관리 툴과의 연동만으로도 사용이 가능케 할 것입니다.
    1. 물론 더 정확한 지표를 얻고 싶다면 추가적인 연동을 진행, 다양한 종류의 이벤트를 우리에게 전송하면 될 것입니다.
    2. 또한, 우리는 해외가 아닌 한국에서 주로 쓰이는 툴들을 위주로 서비스를 전개할 것이며 연동에 필요한 작업들을 최소화할 것입니다.
17. 우리는 팀 내의 각 개발자들에 대한 Blame 요소를 줄이기 위해 팀 내에서 보여지는 이벤트는 팀의 이벤트로 처리할 예정입니다. 각 개발자에 대한 blame은 없습니다. 각 개인의 성과 혹은 작업 내용 등은 제외될 예정입니다.
18. 마지막으로 저희는 저희 서비스와 DORA Metrics의 필요성을 알리기 위해 기술 블로그 운영을 통해 Dora Metrics 관련 정보를 올리면서 저희 서비스 사용법을 알릴 예정에 있습니다. 또한, 컨퍼런스 발표에 적극적으로 참여함으로써 Four Key Metrics를 사용한 개발팀 생산성 관리 문화가 널리 퍼질 수 있도록 할 예정입니다.
19. 기획 심의에서 나왔던 주요 피드백에 대한 저희의 답변입니다.
20. 다른 서비스를 벤치마킹 하여 프로젝트를 업그레이드 해달라는 조언을 해주셨는데, 그에 맞춰 저희도 컨셉을 잡으며 좀 더 명확해지는 결과를 얻었습니다.
    1. 다른 조언으로는 개선 방안을 도출하는 방식을 원하셨지만 다음과 같은 이유로 불가능하다는 판단을 하였습니다.
    2. 저희 프로젝트는 객관적인 지표를 보여줌으로써 팀의 상황에 대한 모니터링을 진행하는 프로젝트입니다. 복합적인 요인이 프로젝트 및 팀의 상황에 적용되긴 하겠으나, 이 모두를 아울러 개선 방안을 도출하는 것은 무리일 것 같다는 판단입니다.
21. 그리고 데이터 수집을 위한 전략 및 데이터 소스 시스템간의 연동을 위한 수행 방안의 구체화에 대한 조언을 해 주셨습니다. 이에 대한 것은 뒤에 있을 아키텍처에서 설명 가능할 것 같습니다.
    1. 그리고, 추가적인 개발을 통해 주기적으로 접속해야 하며 비용을 지불할만한 서비스로 만들어야 할 것이라는 조언 또한 있었는데, 이는 이렇게 설명 가능할 것 같습니다.
    2. DevOps를 이용한 개발이 활성화되면서, 우리는 팀의 성능에 눈을 돌리게 되었습니다. 팀의 개발 역량은 유기적으로 변화하고, 이에 영향을 끼치는 요소들 또한 너무 많습니다. 그렇기 때문에 저희 프로젝트는 성능에 대한 평가를 자동화, 객관화 하여 그 다음을 바라볼 수 있게 하는 툴 로서의 기능을 할 것이라는 의견입니다.
22. 마지막으로 비즈니스 모델이 부족하며, 광고를 제공함으로써 수익을 얻는 것은 좋은 시스템이 아니라는 조언에는 깊이 동감하여 구독형 서비스로 방향을 잡았습니다. 추후 비즈니스 모델 부분에서 설명드릴 수 있을 것 같습니다.
23. 아키텍처, 그리고 데이터 파이프라인에 대한 소개입니다.
24. 저희 팀은 사용자의 데이터를 수집하고 분석하는데 있어 Data Warehouse를 이용한 서버리스 아키텍처를 구현하기로 결정하였습니다. 조사 결과 Github를 포함한 모든 Data source에서 api를 통해 데이터를 받아올 수 있었습니다. 그리고 이것을 필요할 때 마다 가져와서 서비싱 하는 방식 또한 유의미할 것입니다. 하지만, github의 branch 혹은 tag와 같이 github에서 자원으로 취급하지 않지만 저희가 연산 과정에서 사용하는 정보들은 단순한 통신만으로 해결되지 않는다는 문제점이 있었습니다. 이는 추후에 비용 문제를 발생시킬 수 있다는 문제, 그리고 연산의 한계로 인해 저희가 원천 데이터를 저장하기로 하였습니다.
    1. 원천 데이터를 가져오는 방식은 Event Driven Architecture입니다. Serverless Function으로 데이터를 받고, Batch 및 Queue를 통해 End point의 부하 및 down 가능성을 낮춥니다. 여기서 데이터를 가져오는 것은 Time Triggering된 API 처리 architecture일 수도 있지만 event driven architecture를 사용하는 이유는 실시간성과 객관성에 대한 고민의 결과였습니다.
    2. 하지만 그럼에도 사용자의 개인 데이터를 가져오기 위해서는 API 기반 Data 수집 architecture 및 로직이 필요한데, 이를 먼저 구현하지 않은 이유는 Event Driven Architecture의 구조와 데이터 설계가 API 기반 Architecture보다 복잡하고 제한될 것이라고 생각했기 때문입니다.
25. 전체 파이프라인 개요입니다. CloudEvents의 파생 Product인 BigData 수집기 Events Hub를 적극적으로 활용하는 2-Level Serverless Architecture입니다. 다양한 source, 그리고 동시에 들어올 수 있는 많은 이벤트들을 안정적으로 처리하기 위해 Batch 및 Queue를 적극적으로 사용한 Data Store Layer와 이를 사용하기 위한 Market Layer로 나뉩니다.
26. 데이터 스토어 레이어입니다. Scale out/in이 지유롭고 사용량에 따라 비용이 청구되는 서버리스 펑션을 통해 데이터를 처리하고, Event hub, Redis를 통해 Batch 및 Deduplication을 실행하여 Bulk 연산으로 Azure MS SQL에 데이터를 저장하게 됩니다.
27. 데이터 마켓 레이어입니다. Time Trigging 된 ETL Function이 Data warehouse에 있는 데이터들을 계산하여 Business Database에 저장하게 됩니다. 그 이후, 사용자는 자신의 데이터가 변화된 모습을 Azure Static Web App 및 Web App을 통하여 확인할 수 있습니다.
28. 개발 중 겪었던 문제들과 해결해 나간 과정입니다.
29. 먼저 저희 팀의 주제는 예상했던 대로 어려운 과제였습니다. 그렇기 때문에 보시는 것과 같은 문제점들이 발생되었습니다. 하지만, 데이터 파이프라인이 정립되고, 프로젝트를 구성하고 있는 소프트웨어들의 윤곽이 잡혀가고 있기 때문에 앞으로는 상황이 나아질 것으로 예상하고 있습니다.
    1. 먼저, 데이터 파이프라인의 각 기능들을 개발하고 합치는 단계에서 Duplication 문제가 발생하였습니다. 예측하지 못했던 것은 아니었지만 당장의 개발에 힘을 쏟느라 애써 외면했던 것을 인정하고 새로운 방법을 찾아 해결했습니다.
    2. 그 다음은 설계상에서 드러났던 C10K Problem이었습니다. 중앙에서 데이터를 모두 수집하여 처리하고, 이를 Public Web에서 서비싱 하는 우리 서비스의 특성 상, C10K까지는 아니더라도 동시에 많은 데이터가 들어올 때, 어떻게 큰 비용을 들이지 않고도 언제나 안전하게 처리할 수 있을지에 대한 고민이 있었습니다. 이에 대한 결론이 앞서 말씀드렸던 아키텍처라고 할 수 있을 것 같습니다. 또한, 이러한 아키텍처를 검증하기 위해 파이썬 기반 부하 테스트 툴인 Locust.io를 이용한 부하 테스트로 검증을 마쳤습니다.
    3. 마지막으로 저희가 다른 서비스들에 비해 어떠한 방식으로 객관성을 확보할 수 있을지에 대한 의문이었습니다.기존 서비스들은 사용자의 행동 추적을 진행함에 있어 메우 제한적인 방식을 통해 진행하고 있습니다. 이러한 방식은 여러가지 의문을 남겼습니다. 이렇게 나이브한 계산 방식이 정말 팀의 행동을 추적했다고 할 수 있을까? 사용자가 직접 계산하기 어려운 점을 이용하여 일종의 기만을 하고 있는 것은 아닌가에 대한 의문 까지 들었습니다. 하지만, 이를 해결함에 있어 많은 문제가 있었고, 이를 개념적, 기술적인 문제로 해결해냈습니다. 이어질 내용에서 설명드릴 수 있을 것 같습니다.
30. 다음은, 현재 상황과 앞으로의 계획, 그리고 소프트웨어 마에스트로 내에서 예상되는 결과물에 대한 설명입니다.
31. 현재는 Dora Metrics를 보여주는 것까지 성공한 상황입니다. 본 과정 시작 후 지금까지는 Data Pipeline에 집중해왔기 때문에 Front End의 완성도는 낮은 상황입니다. 하지만 최고 난도의 과제라고 생각했던 Data Pipeline의 설계가 끝났고, 그것을 이용해 주요 지표들을 추출했으며, 추후에 존재하는 과제들은 이 Data Pipeline의 틀에서 벗어나지 않은 것임을 고려했을 때, 저희가 의도한 바를 대부분 이룰 수 있을 것으로 예상합니다.
32. 이전에 있었던 아키텍처와 저희가 고민한 내용의 결과로 저희는 다른 서비스들과 차이를 두기로 결정했습니다. 다른 상품들의 경우 Lead Time For Change의 추적에 있어 실제 Release에 포함된 내용이 아닌 특정 브랜치에 PR된 내용만을 추적하기에 그 이전의 시간 및 활동들이 뭉뚱그려지는 단점이 있었으나, 우리는 저희만의 아키텍처와 로직으로 이를 극복하였습니다.
    1. 또한, Deployment Frequency의 경우 실제로 Release된 것을 보는 것이 아닌 Tag의 갯수와 생성 시점만을 보는 다른 서비스들과 달리, 저희는 Release가 실제로 진행된 시점을 잡는 방식으로 확실한 Deployment frequency를 확인할 수 있게 되었습니다.
    2. 그리고 Change Failure Rate 및 Time To Restore Service의 경우, 타 서비스들의 경우 Issue의 open/close만으로 측정하고 있으나, 이는 사람의 손에 측정을 맡기기 때문에 정확하지 않으며, 특히 Time To Restore Service의 경우 문제가 발생하였을 때, 관측하는 시간인 MTTD를 제외한 결과물이라고 할 수 있습니다. 저희 또한 현재 이 지표들을 Issue open/close 로 확인하고 있으나, 추후 Health Check 서비스를 포함하여 측정할 예정에 있습니다.
33. 변경되는 전략적인 계획에 대한 이야기입니다. 초기 기획에 있어서, 저희는 모든 Event Source에서 데이터를 받아 처리할 예정이었습니다. 하지만, 치밀한 연산 계획이 없는 상태로 데이터를 받아 처리함에 있어 다양한 Event Source들은 부정확한 연관관계를 가질 수밖에 없습니다. 이를 방지하고 최대한 빨리 MVP를 내기 위해서 저희는 일단 모든 이벤트의 흐름을 볼 수 있는 Github로 Vendor를 제한하기로 했습니다. 다른 툴들과의 연동은 추후 사업화 과정에서 연구를 통해 추가할 예정입니다.
34. 또한, 현재 파악되는 투자 가능 개발 공수는 방학 시즌인 전반기 보다 적을 것으로 예상됩니다. 이에 계획을 일부 수정, 핵심 가치인 DORA Metrics 지표 추출 및 UI/UX 구현 고도화에 대부분의 우선 순위를 높게 배정하여 저희의 핵심 가치를 MVP를 통해 보여주는 것을 후반기 목표로 잡았습니다. 그 이후, 사용자들을 통해 수집한 피드백 반영 알람 서버를 구현하여 사용자 경험을 향상시킬 수 있도록 노력하겠습니다.
35. 예상되는 결과물과 기능은 다음과 같습니다. 사용자가 속한 팀의 DORA Metrics와 관련 지표들 그리고 코드 작성 품질 및 처리 능력을 Dashboard로 출력, 그리고 BIgdiff//Merged without review 등의 위험 알림을 사용자 지정 메신저에 전송하는 기능입니다.
36. 비즈니스 모델 및 정량적 목표입니다.
37. 현재 최소한의 데이터 파이프라인을 유지하고, 서비싱을 진행함에 있어 필요한 고정 비용은 하루 평균 15,000원입니다. 고정 비용을 구성하는 인프라의 scale을 넘어서는 팀 하나가 하루 500개의 Event를 발생시켰을 때, 데이터 파이프라인에서 추가되는 비용은 약 2000원으로 관측됩니다. 적은 비용으로 유지할 수 있는 인프라의 장점을 최대한 활용, 보시는 것과 같이 서비스 이용 비용을 책정함으로써, 보시는 바와 같이 훌륭한 가격 경쟁력을 가질 수 있을 것으로 보입니다.
38. 정량적 목표입니다. 국내 SW 관련 기업의 수는 2019년 기준 25,188개사입니다. 우리의 전략은 낮은 도입 비용과 쉬운 적용 방식, 그리고 초기 시장 진출에 있습니다. 시장 진입 후, 북미와 한국 개발 시장 간의 아이템 시차를 고려하여 7년 후에는 총 잠재 고객 중 10% 정도를 우리의 사용자로 끌어들일 수 있을 것이라는 예측입니다. 그렇게 시장 진입 7년 안에 순 수익 72억을 발생시키는 것이 우리의 목표입니다.
39. Ppt 발표는 여기 까지로 하고, 알락꼬리여우원숭이팀의 Rainmaker 시연 영상을 보여드리도록 하겠습니다. 발표 들어주셔서 감사합니다.