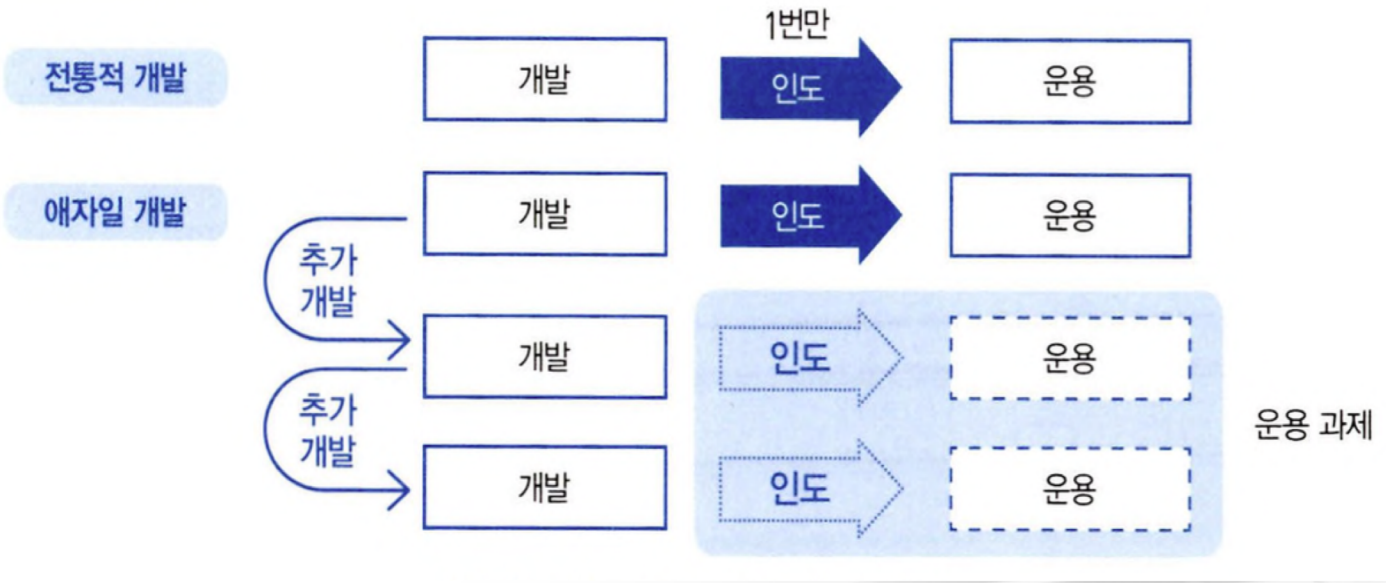
# DevOps

# DevOps

❖ 탄생 배경

✓ 애자일 개발에 의한 계속적인 개발로의 변화

● 전통적 개발 과 애자일 개발

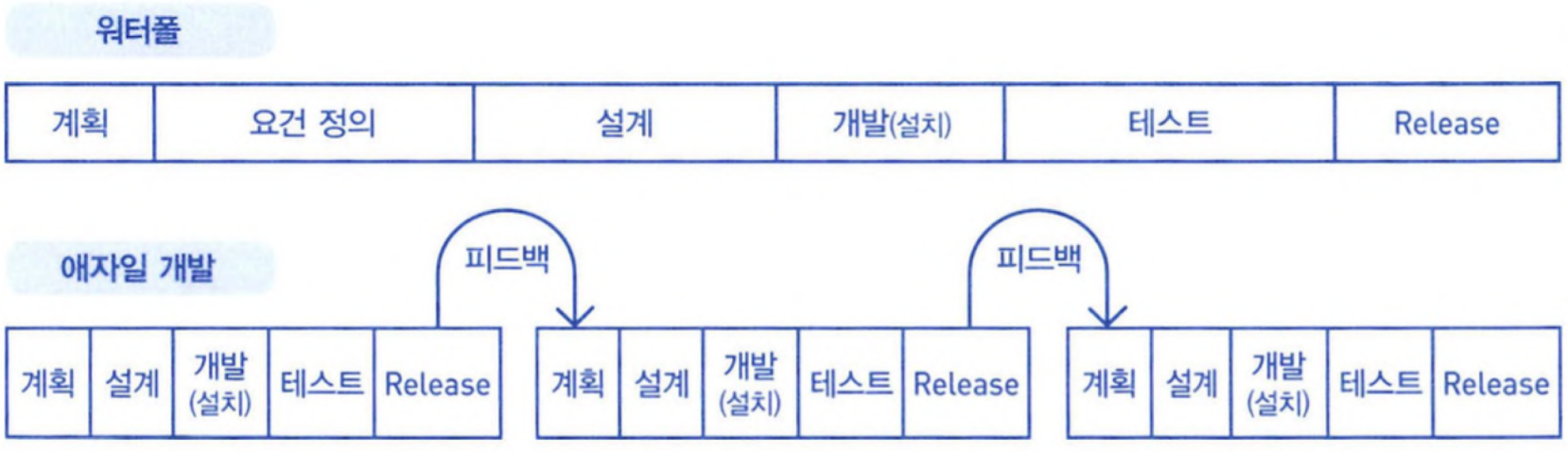


# DevOps

- ❖ 탄생 배경
  - ✓ 애자일 개발에 의한 계속적인 개발로의 변화
    - 현재의 서비스에서는 상당히 짧은 기간에 기능 추가 및 개선의 요구가 발생하고 있는데 워터폴 형태의 개발은 이와 같이 짧은 기간에 점차적으로 요구 사항이 적용되고 상세한 변경이 반복되는 유연성이 필요한 개발에는 대응이 매우 어려움
    - 서비스를 정책적으로 실시하면서 피드백을 받아 개발을 여러 번 반복하는 프로토타입 모델(Prototype Model) 과 소규모 개발을 전제로 필요한 최소한의 요건을 적용한 성과물을 만들어 Release 하고 고객의 피드백을 받아서 계속적인 개선을 반복하는 애자일 개발 기법 등이 탄생하면서 짧은 기간에 개발 사이클(Cycle)을 진행할 수 있는 기반이 만들어 짐
    - 애자일에서는 워터폴 개발과 달리 기능의 추가가 빈번하게 이루어 짐
    - 워터폴 같은 방법에서는 생각할 수도 없는 업데이트 횟수에 대응하고 계속적으로 개선을 반복하기 때문에 계속성 및 효율을 추구하는 개발 방법이나 도구가 생겨나고 지속적 통합(Continuous Integration)이라는 개념이 생겨남
    - 개발자는 개발 범위 효율화 및 자동화를 추진하는 지속적 통합이라는 성과물 생성과 관련된 효율화를 진행했지만 어떻게 하여도 운용 측면 과제가 존재한다는 것을 알게 됨

# DevOps

- ❖ 탄생 배경
  - ✓ 애자일 개발에 의한 계속적인 개발로의 변화
    - ● 워터폴 과 애자일 개발



# DevOps

- ❖ 탄생 배경
  - ✓ 계속적 개발로 인해 나타나기 시작한 운용 과제
    - 개발자는 운용 측면의 과제를 해소하고 싶어 했으며 특히 본인이 대응할 수 없었던 인프라 관련 설정 및 구축을 보다 효율적으로 하려면 어떻게 해야 좋을지에 대한 해결 방법을 생각하게 되었는데 인프라 구축 및 설정 변경을 자동화하는 도구는 오래 전부터 존재하고 있었고 애자일 개발의 등장과 함께 이러한 자동화 도구들이 계속성이나 효율화를 실천하기 위한 도구로써 점차 받아들여지게 됨
    - 개발자의 시점에서 기능 요건 외의 모니터링 그리고 이중화 및 백업 이나 OS 등 비기능 요건을 소홀하게 취급하는 점이 문제
    - 운용 문제에 대해서는 데일리 스크럼(Daily Scrum)으로 불리는 일일 회의를 통해 진행 방향에 대한 해결책을 도출하고 팀 멤버의 관심 여부에 따라 우선 순위를 매기거나 태스크(task)에 페어(Pair)로 대응하고 information radiation(정보 공유) 라는 이른바 정보가 자연스럽게 전해지는 상태를 형성할 필요가 있음
    - 개발 문제에 대한 해결책은 인프라팀을 포함시킨 애자일 개발을 이용하고 인프라 요건을 기존보다 더욱 이른 단계에서 가시화하고 과제에 대해서는 과제의 주인을 명확히 하는 것을 제안
    - Web 개발에서 발생하는 개발과 운용 사이의 혼란을 Wall of Confusion(혼란의 벽) 이라고 명명하여 언급하고 있는데 이 벽을 넘어서 개발과 운용이 서로 간의 영역으로 들어가 애자일 개발처럼 인프라가 운용되는 애자일 인프라를 실현하는 수단으로써 Infrastructure is code(인프라는 코드) 라는 주제로 해결책을 제시

# DevOps

- ❖ 탄생 배경
  - ✓ 계속적 개발로 인해 나타나기 시작한 운용 과제
    - 소프트웨어뿐만이 아니라 컨피규레이션(configuration) 관리를 도입해서 인프라를 관리하는 것 그리고 원스톱 디플로이(One-stop deploy), 모니터링, 인프라의 지속적 통합 등의 방법도 제시
    - 정보 공유의 기본적인 개념으로 개발과 운용은 같은 장소에서 동일한 것을 본다(Dev and Ops see the same thing, in the same place)는 것을 실현하기 위한 수단으로 코드화된 구성 정보를 동일한 소프트웨어 관리 시스템의 Repository에 적재하여 가시화하는 것도 제시하고 전체 인원이 어떤 버전이 사용되는지 파악하기 위한 Branch 운용에서는 항상 메인 Branch에서 Release 하는 방법을 제시
    - 계속적인 개발로 인해 나타난 운용 과제로부터 DevOps라는 사고방식이 형성되고 운용 과제에 대한 해결책으로써 개발(Dev)과 운용(Ops)이 정보를 공유하는 가시화 과정을 통해 인프라의 코드화가 가속화되어 Infrastructure as Code가 형성되어 감

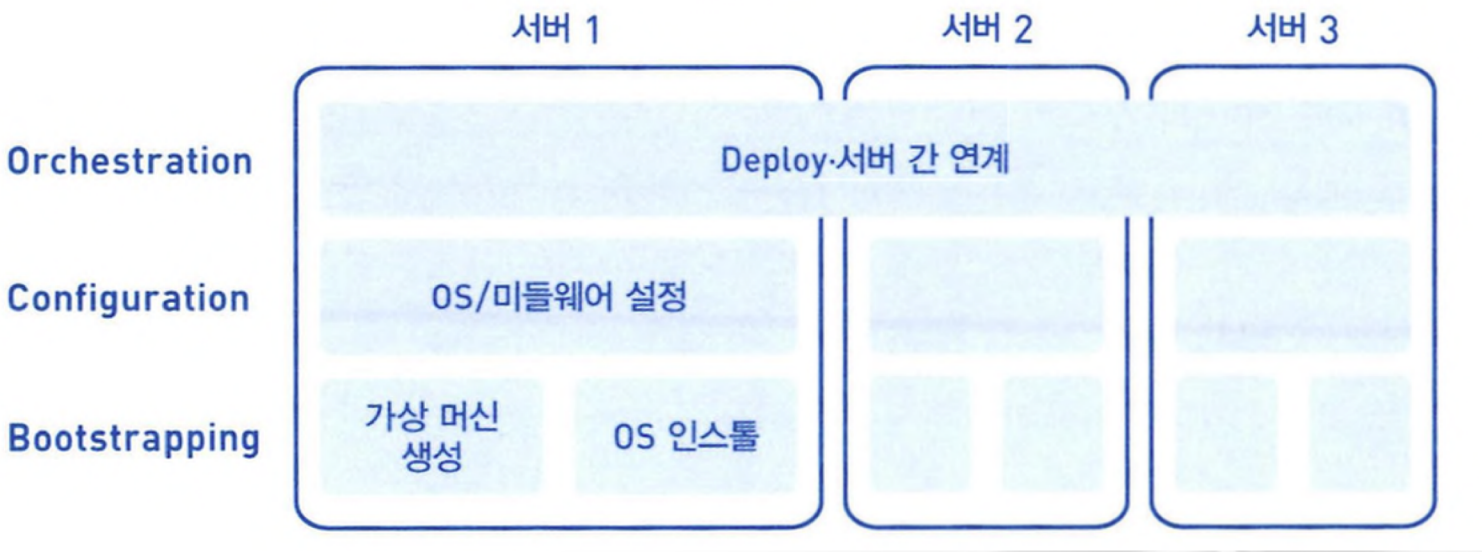
# DevOps

- ❖ 탄생 배경
  - ✓ 인프라 구성 관리 도구
    - 운용 과제 해결을 위한 수단인 DevOps의 인프라 코드화는 서버와 스토리지 그리고 네트워크 등 인프라에 대한 설정을 수행하는 인프라 구성 관리(provisoning) 도구에 의해서 발전
    - 프로비저닝이라고 한마디로 정의하고 있지만 도구로 인프라 설정이 가능한 범위는 다양
    - Velocity 2010에서 리 톰슨(Lee Thompson)은 Provisioning Toolchain 프레젠테이션에서 서버의 프로비저닝을 3개 영역으로 분류

| 프로비저닝 영역 | 설명 |
|---|---|
| Orchestration | Deploy 나 노드 간 연계 등 복수의 서버에 대한 설정 ㅏ나 관리를 담당 |
| Configuration | OS, Middleware의 설정을 담당 |
| Bootstrapping | 가상 서버 생성이나 OS 설치를 담당 |

# DevOps

- ❖ 탄생 배경
  - ✓ 인프라 구성 관리 도구
    - ● 프로비저닝 영역의 구성



# DevOps

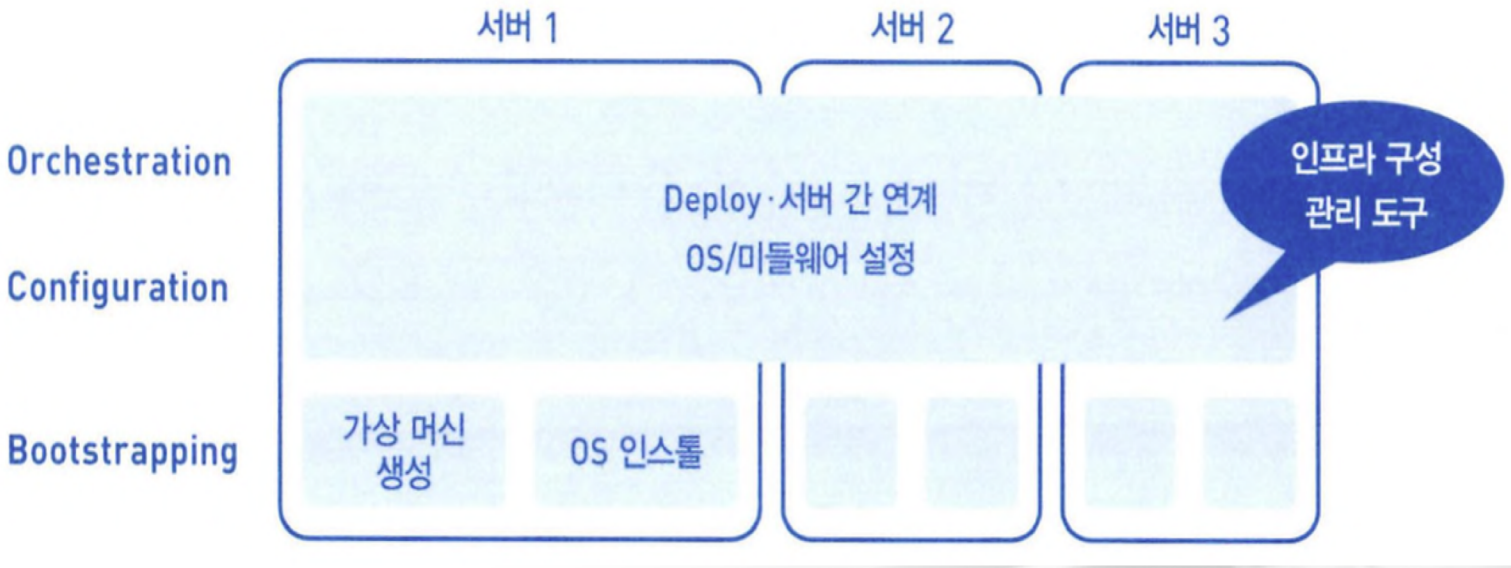
- ❖ 탄생 배경
  - ✓ 인프라 구성 관리 도구
    - ● 프로비저닝 영역의 구성
      - ◆ Bootstrapping 영역
        - ▪ 가상 머신이나 OS의 설치 나 Setup을 의미
        - ▪ IaaS(Infrastructure as a Service)는 인프라 환경을 제공하는 클라우드 서비스에 가상머신의 구성과 컨테이너 생성에 해당
        - ▪ 이용자의 눈높이에서 보자면 인프라 운용 담당자가 하드웨어에 기본(base)이 되는 OS을 설치하는 경우에 이용되는 도구
        - ▪ 가상 머신이나 Docker 같은 컨테이너가 당연한 것이 된 현재는 가상 머신의 템플릿이나 서버 이미지를 만들고 한 번 만들어 놓은 이미지를 바탕으로 설정에 대한 변경 작업을 실시하는 경우가 많아졌기 때문에 인프라 담당자의 담당 영역도 하드웨어보다는 소프트웨어에 보다 가까워지고 있는 등 부트 스트래핑을 이용한 작업의 수행은 많이 없어짐
        - ▪ 가상 머신의 생성으로부터 킥스타트(kickstart)와 연동한 게스트 OS 설치 그리고 템플릿화까지를 일관적으로 실행하는 도구도 등장하고 있음
        - ▪ 해시코프(HashiCorp)사의 Packer2는 OS 설치 및 설정 변경 과 템플릿화까지 한 번에 실행

# DevOps

- ❖ 탄생 배경
  - ✓ 인프라 구성 관리 도구
    - ● 프로비저닝 영역의 구성
      - ◆ Configuration 영역
        - ▪ 부트 스트래핑이 끝난 서버에 대한 OS 설정 변경이나 미들웨어의 설정 적용 등을 의미
      - ◆ Orchestration 영역
        - ▪ 개발 결과물을 복수의 서버로 일괄 Deploy 하듯이 여러 대에 동일한 설정을 하는 작업
      - ◆ 최근에는 Puppet이나 Chef, Ansible 등의 구성 관리 도구로 인해 컨피규레이션과 오케스트레이션의 경계선이 사라지고 있는데 이들 구성 관리 도구는 여러 서버를 Web 서버 그룹 또는 DB 서버 그룹처럼 분류하고 각각의 그룹에 대하여 설정을 배포하거나 명령을 실행할 수 있음

# DevOps

- ❖ 탄생 배경
  - ✓ 인프라 구성 관리 도구
    - ● 최근의 프로비저닝 구성



# DevOps

- ❖ 탄생 배경
  - ✓ Infrastructure as Code 와 DevOps
    - Infrastructure as Code는 인프라의 코드화를 지칭하며 인프라 구성 관리 도구에 의한 설정의 코드화 및 구성 정보화 전체를 지칭
    - 중요한 것은 모든 인프라는 이 설정 및 구성 정보에 의해서만 구축 및 설정이 변경된다는 것
    - 동일한 구성 정보를 일원적(unified)으로 볼 수 있고 실제 기기에 구성 정보의 설정이 반영
    - 인프라 구성 관리 도구 중에서도 컨피규레이션 도구는 직접 실제 기기에 설정을 하는 대신 서버나 미들웨어의 구성 정보와 설정 정보를 정의 파일에 기재하고 도구를 이용하여 정의 파일대로 설정을 수행하여 계속적인 개발로 인해 나타난 운용 과제의 해결 방법이 되는 도구
    - 정의 파일에 기술하는 것만으로 인프라의 구성과 설정을 선언으로 기재할 수 있게 되기 때문에 인프라 플랫폼별 차이를 흡수하기 위한 지식은 필요하지 않기 때문에 어플리케이션 엔지니어도 구성 정보를 기술하는 인프라를 구축할 수 있음
    - Infrastructure as Code는 인프라 구성과 설정의 구성 정보화에 의해서 소프트웨어 영역으로 전환되고 인프라의 구성과 설정에 있어서도 소프트웨어 개발 방법을 적용하는 것이 가능하게 함
    - 설정을 변경하기 위해서는 코드와 같은 구성 정보를 기술하고 구성은 일반적인 어플리케이션 개발과 같은 방법으로 테스트 가능한 도구를 사용하여 테스트

# DevOps

- ❖ 탄생 배경
  - ✓ Infrastructure as Code 와 DevOps
    - 완성된 구성 정보는 코드와 마찬가지로 버전(Version)을 관리하는 것과 같이 개발 부문에서 생겨난 애자일 개발과 같은 계속적인 개발 방법을 인프라 구축 및 운용에도 적용하는 것이 가능
    - Infrastructure as Code라는 단어에서도 알 수 있듯이 인프라를 코드로 표현한다고 해도 코드화를 위한 고도의 프로그래밍 지식은 필요없는데 대부분의 컨피규레이션 도구에서는 프로그래밍 언어보다 학습 비용이 낮은 DSL(Domain Specific Languages)로 인프라 구성과 컨피그(Config)를 기술할 수 있게 되어있기 때문에 프로그래밍과 무관한 인프라 엔지니어도 일반적인 미들웨어의 컨피그를 습득하는 정도의 학습 비용으로 기술하는 것이 가능
    - 인프라 엔지니어도 어플리케이션 엔지니어도 인프라 설정 및 운용에 대하여 서로의 방향에서 다가가는 것이 가능하게 됨
    - 개발과 운용이 긴밀하게 연계되어 비즈니스 가치를 높이려는 DevOps에 있어서 어플리케이션 개발 담당자가 인프라 운용에 대해서 소프트웨어의 지식을 이용하여 이해하고 혹은 직접 설정을 변경할 수 있게 되는 Infrastructure as Code는 매우 중요한 사고방식

# DevOps

- ❖ 탄생 배경
  - ✓ 컨피규레이션 도구의 역사와 특징
    - 현재의 컨피규레이션 도구의 원형이 된 CFEngine은 1993년 마크 버지스(Mark Burgess)에 의해서 오픈 소스 컨피규레이션 도구로 등장했는데 컨피규레이션 도구의 조상이라고도 불리는데 CFEngine은 C 언어로 기술되어 있어 매우 가볍게 동작하고 시스템 간의 차이를 C 언어 등으로 확장하지 않고 DSL을 이용하여 구성을 기술할 수 있는 점에서 매우 주목을 받음
    - 2005년에는 퍼펫 랩스(Puppet Labs)의 창업자이자 CEO인 루크 캐니스(Luke Kanies)가 Ruby로 개발한 Puppet을 발표했는데 Puppet은 매니페스트(Manifests)로 불리는 구성 정보 파일에 독자적인 DSL로 구성을 선언하는 것이 가능하고 Puppet은 C 언어로 쓰여진 CFEngine과 비교해서 Ruby로 되어 있기 때문에 휴대성(Portability)이 뛰어남
    - 2009년에는 아담 제이콥(Adam Jacob)에 의해 Ruby와 Erlang으로 작성된 Chef가 등장했는데 레시피(recipes)로 불리는 구성 정보 파일에 Ruby 기반의 DSL로 구성을 기술하고 Ruby의 문법 등을 그대로 사용할 수 있는 것이 특징
    - 2012년에는 미첼 드한(Michael DeHaan)이 Python으로 쓰여진 컨피규레이션 도구인 Ansible을 등장시켰는데 설정 및 구성 정보는 YAML3 형식으로 표현되지만 실제 기계에 대한 설정을 수행하는 실행 모듈을 다양한 언어로 기술할 수 있는 것이 특징

# DevOps

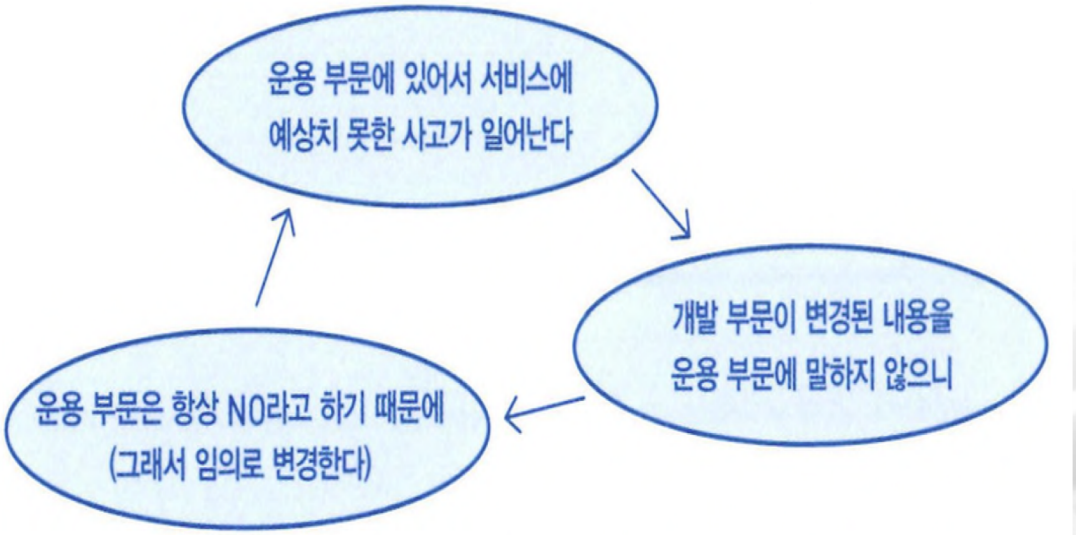
- ❖ 탄생 배경
  - ✓ 컨피규레이션 도구의 역사와 특징
    - Ansible의 장점
      - ◆ 자동화: 자동화에 의한 신속한 설정
      - ◆ 선언적: 구성 정보에 의해 현재 설정 대상의 상태를 명확하게 기재할 수 있고 파악할 수 있음
      - ◆ 추상화: 구성 정보를 설정 대상의 세부적인 환경 차이에 따라서 나누어 기술할 필요가 없는데 가능한 한도에서 코드 실행의 전문성 배제가 가능
      - ◆ 수렴화: 설정 대상이 어떠한 상태이더라도 기대하는 상태로 변경되는 것
      - ◆ 멱등성: 몇 번을 실행해도 같은 결과를 얻을 수 있음

# DevOps

- ❖ 탄생 배경
  - ✓ DevOps의 탄생 과 역사
    - ● DevOps의 태동
      - ◆ 2009년 오라일리(O'Reilly) 주최 벨로시티(Velocity) 컨퍼런스에서 Flickr(당시 루 디코프 - 현 미국 Yahoo) 의 엔지니어 2명이 10+ Deploys per Day : Dev and Ops Cooperation at Flickr(하루에 10회를 초과하는 Deploy: Flickr에 있어서 개발과 운용의 협력)이라는 프레젠테이션을 함
      - ◆ 이 프레젠테이션에서는 전통적인 조직에서의 개발 부문과 운용 부문은 대립 관계에 있다고 언급하고 있는데 개발 부문은 사업(Business)을 위해 서비스에 변화를 주려 하지만 운용 부문은 변화를 싫어함
      - ◆ 운용 부문의 미션은 안정적 가동이기 때문에 손을 대고 싶지않아 하고 거기에 음(Minus)의 무한 반복이 생기기 시작하면 골이 더 깊어지게 됨

# DevOps

- ❖ 탄생 배경
  - ✓ DevOps의 탄생 과 역사
    - ● DevOps의 태동
      - ◆ 개발 부문 과 운용 부문의 음 무한 반복



# DevOps

- ❖ 탄생 배경
  - ✓ DevOps의 탄생 과 역사
    - ● DevOps의 태동
      - ◆ 개발 부문 과 운용 부문의 음 무한 반복
        - ▪ 세상의 변화에 대응하기 위해 개발 부문이 비즈니스에 변화를 주고 싶어 하지만 운용 부문은 안정을 위해 사업으로 인한 변화를 가하고 싶지 않아 하는데 양쪽의 목적은 일치하므로 그렇다면 변화에 겁을 먹고 계속 피할 것인가 그렇지 않다면 변화가 요구될 때마다 변화할 것인가에 대한 갈등이 생겨나는데 슬라이드에는 후자를 위한 아래와 같은 내용이 담겨 있음
        - ▪ Lowering risk of change through tools and culture(변화의 위험을 도구와 문화로 낮춤)

# DevOps

- ❖ 탄생 배경
  - ✓ DevOps의 탄생 과 역사
    - ● DevOps의 태동
      - ◆ 개발 부문 과 운용 부문의 음 무한 반복
        - ▪ 변화에 대응하기 위한 도구
          - Automated infrastructure(인프라 자동화)
          - Shared version control(버전 관리 공유)
          - One step build and deploy(원 스텝 빌드와 deploy)
          - Feature Flags(어플리케이션 기능의 유효/무효를 설정 파일로 관리)
          - Shared metrics(metrics 공유)
          - IRC and IM robots(IRC와 인스턴트 메신저 bot)
        - ▪ 변화에 대응하기 위한 문화
          - Respect(존중)
          - Trust(신뢰)
          - Healthy attitude about failure(실패에 대한 긍정적인 자세)
          - Avoiding Blame(비난을 하지 않는 것)

# DevOps

- ❖ 탄생 배경
  - ✓ DevOps의 탄생 과 역사
    - ● DevOps 탄생
      - ◆ 2009년 10월 30일 벨기에에서 열린 DevOpsDays Ghent 2009에서 패트릭 드부와가 처음으로 DevOps라는 말을 언급하며 탄생과 동시에 충격을 줌
      - ◆ 프레젠테이션 중에서 특히 10+ Deploys per Day 또한 IT 업계에 논란을 불러일으킴
      - ◆ 기존의 개발 문화가 남아 있는 기업에서는 불가능하지 않나 또는 가능하다고는 해도 그렇게 Deploy 하는 것이 의미가 있나 라는 소극적인 의견도 있었음
      - ◆ DevOps라는 말을 만들어 낸 패트릭은 초기 블로그에서 개발과 운용이 협력한다니 어처구니없다 라는 비난을 받게 되어 체념했었다고 말하고 있음
      - ◆ 기업 IT 부문의 개발과 운용 사이에 생긴 골은 깊은 것이지만 그러나 IT 업계 뿐만 아니라 세계의 모든 일들이 헤아릴 수 없는 속도로 변화하는 요즘 아무리 개발과 운용 사이의 골이 깊어도 그것을 메워서 조직 전체가 하나로 움직여 비즈니스에 적용할 수 있는 기업이 경쟁 우위에 설 수 있을 것

# DevOps

- DevOps
  - ✓ 목적은 신속하게 비즈니스 요구에 응하는 것
    - DevOps에서는 개발 부문과 운용 부문이 긴밀히 연계하고 다양한 방법과 문화를 도입하여 상품이나 서비스 개선에 걸리는 시간을 단축하며 신속하게 비즈니스 요구에 대응하는 것이 요구됨
    - 반복되는 것에 대한 개선을 통해 새로운 정책을 실시하거나 잘 안 되는 곳을 신속하게 변화시켜 감으로써 비즈니스를 지원
    - 단지 신규 서비스를 창출하거나 신규 기능을 추가하는 것에 머물지 않고 기존의 서비스를 신속하게 개발과 운용 양쪽에서 개선하여 잘 만들어 가는 것이 매우 중요
    - 신속하게 비즈니스 요구에 부응한다는 기본적인 생각을 나타내고 있는 것이 DevOps라는 단어가 생겨난 벨로시티
    - 벨로시티는 단위 시간당 비즈니스에 대한 기여도를 의미하는데 얼마나 빠르게 비즈니스에 기여하는가를 결과로 하여 나타난 산물 중의 하나가 DevOps
    - 개발과 운용이 긴밀하게 연계하고 상호 간의 관점을 가지고 서비스를 개선하기 위해서는 서로의 전문성을 인정하고 어떻게 서로를 더 이해할 수 있을까 생각하는 것이 중점
    - 개발과 운용의 연계를 뒷받침하는 도구의 종류로써 인프라의 자동화나 버전 관리의 공유 등이 있음

# DevOps

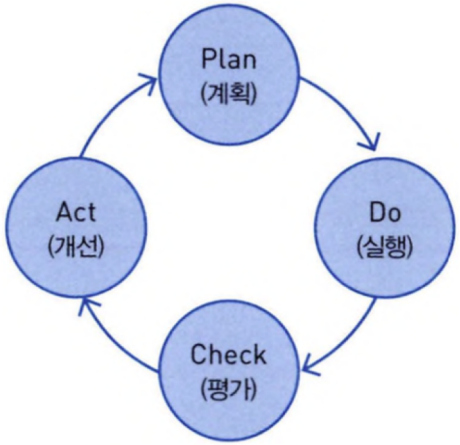
- DevOps
  - ✓ 목적은 신속하게 비즈니스 요구에 응하는 것
    - 개발 과 운용의 연계를 뒷받침하는 도구를 형성하는 요소
      - 추상화: 모든 리소스를 추상화하여 어떠한 플랫폼의 차이도 흡수하고 전문성과 복잡성을 절감하는 것
      - 자동화: 추상화된 리소스 이용의 자동화를 가능하게 하고 전문성이나 개발과 운용을 하는 인적 부담을 줄여 주는 것
      - 공통화: 공통의 버전 관리 시스템 과 커뮤니케이션 도구를 이용하여 정보를 가시화하고 개발과 운용의 긴밀한 관계를 구축하는 것
      - 지속적 통합: 개발과 운용의 개발 및 구축 방법 통일화로 개선 속도를 비약적으로 신장하는 것
      - 모니터링: 리소스 정보를 일원화하여 가시화하고 개발과 운용이 긴밀하게 협력할 수 있는 관계를 구축하는 것

# DevOps

- DevOps
  - ✓ 목적은 신속하게 비즈니스 요구에 응하는 것
    - 개발 과 운용의 연계를 지원하는 문화를 형성하는 요소
      - 목적 의식: 개발 과 운용이 함께 서비스를 키워서 신속하게 비즈니스 요구에 부응하는 공통의 목표를 가지고 연계하기 쉽게 하는 것
      - 공감대: 개발과 운용이 서로의 팀 감정을 없애고 받아들인 관계를 긴밀하게 하는 것
      - 자율적 사고: 개발 부문과 운용 부문이 의뢰를 기다리는 것이 아니라 자율적으로 움직임으로써 공통의 목표 달성에 근접하는 것
    - 개발과 운용의 연계를 뒷받침하는 도구를 형성하는 요소는 주로 전문성과 복잡성을 배제하고 오버헤드를 줄이며 정보를 가시화하는 것을 목표로 함으로써 동일한 정보에 기반하여 신속히 움직일 수 있도록 개발 과 운용의 긴밀한 연계를 지원하고 자동화나 지속적 통합으로 개선에 소요되는 시간을 극적으로 줄여 신속이라는 목적을 달성하도록 연결하는 것을 목표로 함
    - 개발과 운용의 연계를 지지하는 문화를 형성하는 요소는 개발과 운용이 자신들의 미션(mission)을 넘어 공통된 목적을 가지고 서로의 입장을 이해하면서 자율적으로 생각하고 행동하는 비즈니스 요구에 대응한다 라는 것을 목표로 함
    - DevOps에 얽혀 있는 다양한 문화적 사고방식과 도구는 DevOps가 지향하는 개발 부문과 운용 부문이 긴밀히 연계하고 다양한 방법과 문화를 도입하는 상품이나 서비스 개선에 걸리는 시간을 단축하며 신속히 비즈니스 요구에 대응하는 것에 공통점을 갖고 여러 가지 요소들에 의해서 형성되고 있음

# DevOps

- DevOps
  - ✓ PDCA 사이클과 DevOps
    - PDCA 사이클이란 현대 품질 관리의 아버지로 불리는 에드워즈 데밍(Edwards Deming)에 의해서 제창된 기업 활동 과 사업 활동의 계속적인 생산성 개선 및 제어에 이용되는 관리 방법
    - 업무를 Plan(계획) – Do(실행) – Check(평가) – Act(개선) 의 4단계로 개선하고 개선된 상태에서 또 다시 Plan(계획) 으로 되돌아가서 계속적으로 개선을 실시
    - PDCA를 1회전 돌리는 것으로 다음 회전에서는 더욱 높은 것을 목표로 할 수 있음
    - DevOps는 DevOps를 지탱하고 있는 사고방식 과 정책 도구 등 여러 가지가 조합되어 이루어지고 있는데 어느 것 하나가 빠지면 DevOps 라고 애기할 수 없으며 한 번에 모든 개선을 완성형으로 도입하는 것이 아니라 매일매일 개선하고 실천해 가는 것

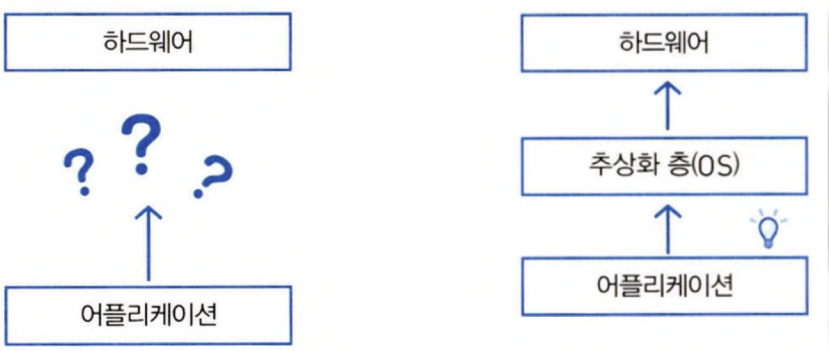


# DevOps

- DevOps
  - ✓ PDCA 사이클과 DevOps
    - DevOps 도입도 PDCA로 수행할 필요가 있으며 지속적 통합 등에서도 PDCA 방법을 이용하여 서비스를 계속적으로 개선하는 방식으로 되어 있음
    - DevOps 와 PDCA 사이클은 떼어 놓을 수 없는 관계
    - DevOps에서 PDCA 사이클을 수행하기 위한 방법 중에 애자일 개발 기법이 그러한 사례 가운데 하나
    - 지속적 통합이라고 불리는 개발물의 구축(Build)과 시험 각각의 결과에 대한 피드백(Feedback)까지 계속적으로 실행하는 방법은 개발물의 품질을 높이기 위해 PDCA 사이클을 구현한 것이라고 할 수 있음
    - 개발 부문과 운용 부문의 커뮤니케이션 및 환경과 관련된 다양한 정보에 대한 모니터링 등 계속적인 개선을 PDCA로 수행하고 있음

# DevOps

- DevOps
  - ✓ 추상화
    - 추상화라는 것은 모든 리소스가 추상화되지 않는 플랫폼의 차이도 흡수하는 것을 의미
    - 인프라의 추상화로 OS 그리고 서버 및 스토리지  와 네트워크 등의 추상화를 들 수 있음
    - 추상화를 실시함에 따라 전문성과 복집성을 감소시킬 수 있음
    - DevOps에서는 개발과 운용 간 긴밀한 연계를 필요로 하지만 추상화에 의한 인프라의 전문성과 복잡성 배제 덕택에 개발 부문에서 인프라의 구성이나 설정으로 접근할 수 있게 됨
    - 추상화를 세분화하면 2가지 해석이 존재하는데 첫 번째는 동일한 규격이나 룰(Rule)에 의해서 단일 기기 또는 복수의 상이한 프로그램이나 기기를 불러올 수 있는 것으로 이것을 표준화라고 부르며 두 번째는 실제로는 존재하지 않는 무엇인가로 되는 것으로 이것을 가상화라고 부름



# DevOps

- DevOps
  - ✓ 추상화
    - 표준화 및 가상화에 대한 접근은 OS, 서버, 스토리지, 네트워크에서도 동일한 모습으로 실행되며 개별 하드웨어 관련 노하우가 소프트웨어로 점차 도입되고 있음
    - DevOps 지원 기술에서는 인프라의 전문성을 배제하는 개발 측면에서 인프라 구성에 접근할 수 있도록 표준화, 가상화의 흐름을 맞이하고 있음

# DevOps

- DevOps
  - ✓ 추상화
    - OS 가상화
      - OS의 추상화는 하나의 OS 위에서 여러 사람이 서로 다른 어플리케이션을 만들고 싶다는 의도로 시작되었는데 1979년 파일 시스템(File System) 분리가 가능한 chroot가 등장한 것이 그 원형으로 파일 시스템 분리에 머물렀던 chroot에 반해서 2000년 FreeBSD 라는 OS에서 Jail이라는 방안이 생겨나고 일부 시스템의 변경 권한(root 권한)을 분리할 수 있게 됨
      - 2001년에는 Jail 방안을 기반으로 한 Linux-VServer 개발이 시작되어 하나의 서버에서 CPU 시간 과 메모리 및 네트워크 등 자원(Resource)을 분할하여 복수의 Linux 가상 서버를 작동시키는 방안이 생겨났으며 2005년에는 Solaris Containers가 등장하였고 2008년에는 Linux에도 LXC(The Linux Containers Project)가 생겨나 개별 프로세스 단위로 네임 스페이스(Name Space)에 의한 자원 분리를 실시함으로써 가상화를 실현하는 컨테이너라는 개념이 퍼짐
      - 2013년 dotCloud(현재는 Docker)에서 LXC를 이용하여 구현된 컨테이너인 Docker가 오픈 소스화되어 폭발적으로 보급되었으며 Docker는 Go 언어로 구현하고 플랫폼 간 이식성(Portabillity)을 향상시키고 있음
      - OS 추상화는 계속 진화되고 있음
      - OS 가상화는 DevOps에 있어서 개발과 운용의 긴밀한 연계를 위해 인프라 구성을 소프트웨어로 끌어모아 관리를 일원화하거나 구성을 변경할 때 사용하는 Infrastructure as Code에서도 가장 많이 이용되는 기술 중 하나

# DevOps

- DevOps
  - ✓ 추상화
    - ● OS 가상화
      - ◆ OS 가상화 방식

# DevOps

- DevOps
  - ✓ 추상화
    - ● 서버 가상화
      - ◆ PC 위에 가상 PC를 만드는 컨셉으로 1999년에 VMware에서 VMware 1.0이 등장하면서 모든 물리적 하드웨어를 소프트웨어로 표현하고 OS 위에서 완전히 다른 OS를 구동시킬 수 있게 됨
      - ◆ 가상화된 하드웨어를 가상 머신, 가상 머신을 탑재한 원래의 OS를 가상화 호스트라고 부르는데 하이퍼바이저(Hypervisor)라고도 불림
      - ◆ Linux나 Windows와 같은 OS를 설치한 후에 하이퍼바이저 소프트웨어를 넣는 경우 와 물리 하드웨어 위에 하이퍼바이저를 넣는 베어메탈 하이퍼바이저 방법이 있음
      - ◆ 2003년에는 Xen5이라는 리눅스 커널(Linux Kernel)을 하이퍼바이저화하는 기술이 등장했고 2006년에는 KVM이라는 가상화 기술이 등장했으며 이듬해 2007년에는 리눅스 커널에 KVM이 통합되어 리눅스 디스트리뷰션(Linux Distribution)에서는 표준적인 서버 가상화를 도입할 수 있게 됨
      - ◆ 2009년에는 VMware vSphere가 등장해서 가벼운 가상화 전용 OS(베어메탈 하이퍼바이저)로 크게 주목을 받았고 실제 대기업에도 이용됨

# DevOps

- DevOps
  - ✓ 추상화
    - 서버 가상화
      - OS 추상화와 서버 추상화 양쪽의 장점을 얻기 위해서 가상 머신 위에 컨테이너를 구성하는 하이브리드 구성이 주목을 받고 있는데 Google이 2015년에 Borg라는 컨테이너형 아키텍처를 발표했는데 하드웨어 추상화인 가상 머신 위에 컨테이너를 전개함으로써 하드웨어 자원에 대한 제약 사항에 신경 쓰지 않고 컨테이너에 의해서 하드웨어를 최대치까지 사용하는 것이 가능
      - 서버 가상화로 인해 보다 물리적 제약에서 벗어나서 서버 및 인프라 구성을 설계할 수 있게 됨
      - 오늘날에는 물리적 하드웨어에 대한 고려는 크게 줄었으며 인프라 담당자 밖에 모르는 지식 또한 점점 줄어들고 있음

# DevOps

- DevOps
  - ✓ 추상화
    - ● 스토리지 가상화
      - ◆ 하나의 물리 스토리지에 논리 스토리지를 여러 개 생성하거나 정책에 기반하여 동작하는 내용을 배포 및 결정할 수 있는 스토리지 기기를 SDS(SoftWare Defined Storage)라고 부름
      - ◆ EMC와 NetApp 등 스토리지 기업에서 많은 스토리지용 OS가 소프트웨어로 제어할 수 있도록 API(Application Programming Interface)를 두고 있어 보다 소프트웨어와 가까워지고 있음
      - ◆ 스토리지는 전문 엔지니어가 각 서비스에 대한 용량을 계산하고 데이터 배치를 결정하거나 디스크 증설 계획을 세우는 등 개발 부문에서는 인프라 담당자가 무엇을 하는지 전혀 모르는 상태였지만 SDS로 가상화된 스토리지를 소프트웨어로 제어할 수 있게 되어 보다 개발과 운용이 긴밀하게 연계될 수 있는 환경이 됨

# DevOps

- DevOps
  - ✓ 추상화
    - ● 네트워크 가상화
      - ◆ 네트워크 추상화의 수단으로써 이전부터 알려진 것이 VLAN(Virtual Local Area Network)인데 1994년에 구축된 것이 시초이며 이는 하나의 물리 스위치를 복수의 논리적인 스위치로 나누기 위한 기술
      - ◆ 스위치는 하나의 시스템에 다수 설치되는 경우가 많은데 모든 스위치에 동일한 설정을 하지 않으면 네트워크가 구성되지 않음
      - ◆ 대규모 네트워크의 유연한 변경에 대한 수요가 커졌으며 앞선 문제를 해결하기 위한 수단으로써 SDN(Software Defined Network)이 제창됨
      - ◆ SDN의 구현 방법은 몇 가지가 있지만 공통적인 것은 스위치를 Control Plane(설정 등의 관리 부문) 과 Data Plane(패킷 전송 부문)으로 분리하는 것
      - ◆ 정책이나 설정 내용의 제어를 담당하는 Control Plane은 소프트웨어에 의해서 관리됨
      - ◆ 개발 부문은 낮은 학습 비용으로 소프트웨어 지식을 구사하면서 네트워크 설정을 변경할 수 있음
      - ◆ 방화벽(Firewall) 및 부하 분산(Load balancing)의 모습은 이전에는 전문 엔지니어만 만질 수 있었지만 소프트웨어에 의한 제어로 이동함에 따라 인프라 담당자뿐만이 아니라 개발 부문도 내용을 파악하거나 변경할 수 있게 되었음

# DevOps

- DevOps
  - ✓ 자동화
    - 자동화는 기계적으로 프로그램을 통해 제어하여 사람의 조작을 필요로 하지 않는 작업을 의미
    - 어플리케이션 빌드(Build)나 어플리케이션 테스트 등 매번 같은 조작을 필요로 하는 경우에 그 동작을 프로그래밍한 후 프로그램으로 일련의 동작을 수행함으로써 실현
    - 추상화되고 소프트웨어 화된 인프라는 설정도 소프트웨어에서 완료할 수 있게 되었기 때문에 프로그래밍이 가능
    - 컨테이너와 가상화 등의 기술이 일찍부터 적용된 서버나 OS 또한 프로그래밍으로 조작이 가능한 것은 물론이고 스토리지 와 네트워크 등 하드웨어에 대해서도 REST API라고 불리는 리소스를 URL로 나타내는 HTTP 프로토콜을 사용하여 자원의 상태를 취득하거나 설정을 변경할 수 있는 API(Application Programming Interface)를 제공하는 것이 일반적
    - REST API를 사용하여 리소스 상태를 변경하는 절차를 프로그램으로 만듦으로써 기존의 변경 절차서 등을 작성하여 수작업으로 했던 작업을 모두 기계적으로 자동으로 수행할 수 있게 되었음
    - 추상화되고 소프트웨어 화된 OS 그리고 서버 및 스토리지 와 네트워크 등은 지정된 파라미터를 바탕으로 모든 설정을 자동으로 수행할 수 있도록 진행되고 있음
    - 서버를 한 대 추가하려고 할 때 컨테이너 등의 기술을 이용하여 OS를 작동하고 스토리지를 설정하며 네트워크에 편입하는 등 일련의 작업을 자동으로 수행하는 것이 가능
    - 자동화는 오픈 소스 형태의 다양한 소프트웨어를 조합하는 것으로 가능

# DevOps

- DevOps
  - ✓ 공통화
    - 이슈 트레킹 시스템
      - 커뮤니케이션 도구의 통일화는 애자일 개발을 하면서 발전했으며 DevOps에서는 개발과 운용의 연계 때문에 더욱 중요하게 여겨짐
      - 정보의 일원화와 가시화는 긴밀한 연계를 실시하기 위한 필수 요소로 최근에는 단지 대화의 원활한 진행 만을 목적으로 하지 않고 운용 측면에서도 시스템 연계를 쉽게 해 주는 도구가 많이 등장하고 있는데 이슈 트래킹 시스템(ITS : Issue Tracking System)은 티켓(Ticket) 트래킹 시스템으로도 불리며 과제가 적힌 티켓이 해소되는 순간 곧바로 채팅 도구와 같은 커뮤니케이션 도구로 통지되도록 되어 있음
      - 유명한 도구로 지라(JIRA)와 레드마인(Redrmine), 트랙(Trac) 등이 있음
      - 장애가 발생했을 때 어느 서버의 어떤 컴포넌트에서 장애가 발생했는지 자동으로 이슈 트래킹 시스템에 전송하고 돌발상황에 대한 관리를 하며 통지까지 한 번에 담당하는 페이저듀티(PagerDuty) 같은 서비스도 등장하고 있음
      - 이슈 트래킹 시스템인 지라 와 레드마인 및 트랙은 모두 애자일 개발에 대응하는 기능을 가지고 있음
      - 애자일 개발에서 백로그(Backlog) 관리는 버그(Bug) 관리나 TO DO 관리와 동일한 형태로 되어 있으며 추가적인 사양이 기재된 User Story 등을 취급할 수 있음

# DevOps

- DevOps
  - ✓ 공통화
    - 이슈 트레킹 시스템
      - 애자일 개발에서 이용되는 작업(Task)별 진척(미착수, 실행 중, 작업 완료)을 포스트잇(Post-it)으로 실시하는 방식인 화이트 보드(Whiteeboard) 등에 GUI 형태로 대응하거나 버전 관리 시스템과 연계하여 어떤 Story가 어떤 개발에 대응되었는지를 시스템으로 연동시키는 것이 가능한 것도 있음
      - 백로그 중에서 대응의 우선 순위를 정하고 드래그 앤 드롭(Drag and Drop)으로 정렬하거나 Story Point라고 불리는 팀에게 있어서는 대응 범위 정도를 나타내는 숫자를 바탕으로 다각적인 그래프를 작성할 수 있는 도구도 있음
      - 지라는 대시보드(Dashboard)라고 불리는 그래프 생성 및 상태 파악 기능에 특화되어 있기 때문에 팀 개발 관리를 위해 여러 가지 자료를 만들지 않고 팀원이 실제 개발에서 참조하는 태스크(Task), 사양(Spec)이 일원화 관리되는 장소에서 그대로 진척을 파악할 수 있게 되어 있음
      - 이러한 특징에서 볼 때 Iteration 과 Sprint라고 불리는 짧은 개발 사이클에서 개발을 계속하는 것과 디지털 데이터를 이용하여 유연하게 과제를 관리하거나 우선순위 설정에 대응할 수 있는 과제 관리 시스템은 매우 궁합이 좋다고 말할 수 있음

# DevOps

- DevOps
  - ✓ 공통화
    - ● 이슈 트레킹 시스템
      - ◆ 개발물이나 인프라 코드의 커밋(Commit), 모든 태스크에 대해서도 티켓으로 발행하는 것을 권장하는 개발 방법으로 티켓 구동 개발이 알려져 있음
      - ◆ 설계 정보나 일일 회의록 등의 공통화는 각 팀이 개별적으로 텍스트나 엑셀에 기재하고 제한된 공유 폴더로 관리했던 설계 정보 및 회의록을 Wiki와 같은 Web을 이용한 도구로 정보를 일원화 관리하는 것으로 이를 통해 상호 간에 수행하고 있는 것을 폭넓은 범위에서 공개할 수 있으며 검색하기 쉽고 동시에 편집이나 편집 내용의 이력 관리도 가능하게 되었는데 대표적인 도구로써 레드마인의 위키(Wiki) 기능과 컨플루언스(Confluence)가 있으며 이러한 도구도 이슈 트래킹 시스템과 마찬가지로 외부 시스템과 연계가 가능하도록 설계되어 있고 커뮤니케이션 도구에 갱신 정보를 통지하거나 URL을 직접 공유함으로써 특정 정보를 직접 참조할 수 있음

# DevOps

- DevOps
  - ✓ 공통화
    - 커뮤니케이션 도구
      - 커뮤니케이션 도구에 대해서는 채팅을 이용한 개발 스타일이 침투하고 있음
      - 개발 부문과 운용 부문이 동일한 커뮤니케이션 도구를 사용함으로써 이러한 정보를 바탕으로 대화가 쉬워졌으며 또한 이슈 트래킹 시스템이나 회의록 등에 대한 정보의 Web화로 인해 URL을 기반으로 관계자와 정보 교환을 할 수 있게 됨
      - 도구로써 옛날에는 IRC가 이용되었고 현재에는 대표적인 도구로 스카이프(Skype)와 슬랙(Slack), 챗워크(ChatWork) 등이 각 회사의 업무에서 이용되고 있음
      - 최근 주목받고 있는 것은 채팅 도구에 bot(robot의 약칭으로 로봇이란 뜻)의 투입하는 것인데 채팅 도구에 bot 계정을 등록함으로써 채팅 도구로 구현하기 곤란한 시스템 간 연계를 bot으로 실현할 수 있는데 예를 들면 장애를 탐지하는 시스템과 연계하여 장애 발생 시 bot이 어느 시스템에서 장애가 발생했는지 말할 수도 있음
      - bot은 시스템 간 연계 목적 뿐 만이 아니라 사람이 했던 간단한 작업을 대체할 수 있어서 채팅 상의 어떤 단어에 대응하여 발언 내용에 따른 동작을 하도록 프로그래밍하면 예를 들면 특정 단어에서 서비스를 재시작하거나 서버를 추가하는 등의 작업을 팀에서 대화를 하면서 실현할 수 있게 됨
      - 이로써 누가 무엇을 하고 있는지 명확하게 될 뿐만 아니라 작업까지의 단계가 훨씬 줄어들어 더욱 효율적인데 채팅 도구와 bot의 연계로 시스템 간 연계 뿐 만 아니라 사람이 수행하는 작업을 대체하는 등 많은 효과가 나타나기 때문에 ChatOps라고 불리기도 함

# DevOps

- DevOps
  - ✓ 공통화
    - 소프트웨어 구성 관리 도구
      - 정보의 일원화는 커뮤니케이션 뿐 아니라 소프트웨어 구성 관리 도구(SCM : Software Configuration Management)를 이용함으로써 코드화 및 소프트웨어화 된 인프라의 구성 정보 및 설정 정보를 소프트웨어 구성 관리 도구에 저장하고 변경 시점 및 변경자를 버전 관리함으로써 개발과 운용 모두 동일한 정보를 바탕으로 설정 및 수정이나 확인이 가능하게 됨
      - 이로 인해 복수의 파일에 걸친 절차서 나 유지 관리(Maintenance)되고 있지 않은 설정 정보 등을 없앨 수 있게 됨
      - 소프트웨어 구성 관리 도구는 버전 관리나 Release 관리 등도 포함한 시스템을 내포

# DevOps

- DevOps
  - ✓ 지속적 통합
    - 애자일 개발에서 지속적 통합은 코드의 Build 및 정적 및 동적 테스트를 빈번하게 계속적으로 실시하는 것을 의미하는데 지속적 통합은 Continuous Integration이라고 하기 때문에 CI로 축약되기도 함
    - 지속적 통합은 Code로 인하여 발생하는 문제를 조기에 발견하고 Build 및 테스트에 관계된 작업 비용(Cost)의 낮춤 그리고 테스트 상황의 가시화 등 커다란 장점이 있음
    - Infrastructure as Code 와 관련된 방안을 조합하여 어플리케이션과 인프라 양쪽의 성과물(Code)을 Build -> 테스트 -> Release의 지속적 통합 절차에 반영함으로써 개발 부문과 운용 부문이 서로의 지식을 공유할 수 있으며 문제에 대한 가시화를 앞당길 수 있고 지속적 통합 도구를 이용함으로써 Build나 테스트 등 비슷한 작업을 자동화하고 반복적으로 실시할 수 있어 서비스를 개선하고 Release 준비까지 신속하게 진행할 수 있으며 그 결과 딜리버리 타임의 대폭적인 단축이 가능하게 되고 DevOps를 통해 신속한 비즈니스 가치 향상에 기여할 수 있게 됨
    - 지속적 통합 도구로는 오픈 소스로 Jenkins가 있고 클라우드의 지속적 통합 도구 서비스로는 트래비스 CI(Travis CI) 와 서클 CI(circle CI) 등이 잘 알려져 있는데 클라우드에 인프라 환경을 만드는 경우는 이러한 외부 서비스와 연계하여 신속하게 지속적 통합 환경을 만드는 것이 가능
    - 구축부터 시험까지의 작업(Job)이 성공하면 운영 환경으로 Release 준비까지를 완료하는 지속적 딜리버리(Continuous Delivery)라는 개념도 있는데 이것 또한 DevOps의 비즈니스 가치를 높인다는 방안을 따르고 있어 개선책이 도입되자 마자 지속적 딜리버리 방식으로 이용자가 신속하게 개선된 내용을 사용할 수 있게 됨

# DevOps

- DevOps
  - ✓ 모니터링
    - 시스템 상황을 실시간으로 정확히 관측할 수 있는 모니터링 기반을 구성
    - DevOps의 PDCA 사이클에서 Check(평가)를 하기 위한 기반이며 다음에 어떤 Action(실행)을 일으킬 것인가를 계획하는 재료
    - 모니터링은 원래 Resource 상황과 작동 여부 상태를 실시간으로 보기 위한 목적으로 이용되어 왔지만 계속적인 개선을 하기 위해서 보다 비즈니스에 입각한 값을 취득하고 분석하게 되었음
    - 단순한 감시 용도가 아니라 Resource의 상태를 계속적으로 취득하고 가시화하는 매트릭스 모니터링(Matrix monitoring)에서는 서버의 부하 상황(CPU나 메모리 등의 사용량) 뿐 만 아니라 서비스 이용자의 Web 페이지 접속 수 등을 수치화하여 정량적 분석을 실시할 수 있게 함
    - 예를 들면 새로 내놓은 캠페인의 효과를 알아보거나 다음 캠페인을 내놓을 경우 인프라 증강 등에 대한 기준으로 사용하기 위한 수치 값을 취득하는 것을 생각할 수 있음
    - 보다 비즈니스에 직접 관계 있는 모니터링으로 예를 들면 사이트 방문자 수와 상품을 구입한 사람 수에 대한 비율을 비교하는 전환율을 모니터링하고 정책의 효과를 측정하는 경우도 있음
    - 신규 정책을 반영한 페이지와 그렇지 않은 페이지를 나누어 효과를 알아보는 A/B 테스트를 실시하여 각각의 페이지 접속 수나 어떤 페이지에서 이탈했는지를 확인하면서 다음 정책을 검토하는 데에 모니터링이 사용됨

# DevOps

- DevOps
  - ✓ 모니터링
    - 모니터링 결과를 활용

# DevOps

- DevOps
  - ✓ 모니터링
    - CPU 사용률 및 메모리 사용량 그리고 디스크 I/O의 리소스 소비 비율 등 매트릭스를 모니터링하여 계속적 개선으로 성능 튜닝을 하는 용도로 사용하기도 함
    - 널리 알려진 감시 도구는 자빅스(Zabbix), 무닌(Munin), JP1, Hinemos 등이 있는데 모니터링 없이는 계속적 개선을 할 수 없기 때문에 DevOps에서 매우 중요한 구성 요소 중의 하나
    - 로그를 정보의 원천으로 하는 매트릭스 모니터링은 복수의 미들웨어가 조합되어 실현
    - 로그를 수집하는 미들웨어 그리고 수집한 로그에서 필요한 정보를 검색하는 미들웨어와 수집한 결과를 보기 쉽게 그래프화하는 종류의 미들웨어 조합
    - 요즘 서비스를 구성하는 기기의 숫자는 예전과 비교하여 매우 많아졌으며 서버의 증감을 더욱 편리하게 할 수 있도록 되었음
    - 고정된 로그 취득 방법에서는 변화에 대한 대응이 어렵고 로그의 양도 엄청나게 많아져서 수집 및 검색 그리고 가공을 단일 시스템에서 실시하기 힘들기 때문에 각 분야에서 뛰어난 미들웨어를 조합하는 것이 최근 주류가 되고 있음
    - 조합의 예로 Elasticsearch(로그 전문 검색), Logstash(로그 수집), Kibana(로그 가시화) 조합이 있는데 이들의 머리글자를 따서 ELK Stack으로 불리며 널리 사용되고 있으며 이러한 구성이 사실상 표준(De facto Standard)이 되고 있음
    - 이외에도 모니터링 인터페이스로써 최근 확산되고 있는 것은 그라파나(Grafana) 등이 있으며 로그 수집에 플루언티드(Fluentd)를 취급하는 것도 있음

# DevOps

- DevOps
  - ✓ 목적 의식, 공감, 자율적 사고
    - DevOps에서는 개발과 운용의 긴밀한 연계가 필요하기 때문에 공통의 목적 의식을 갖는 것이 중요
    - 기존의 개발은 새로운 기능 추가를 목적으로 하였고 운용은 서비스를 멈추지 않고 지속시키는 것이 목적이므로 개발 부문이 새로운 기능을 개발하고 서비스에 적용하려고 하여도 서비스를 안정적으로 가동하던 운용 부문에서 본다면 운용을 불안하게 할 수 있는 새로운 기능의 추가는 서로에게 상당한 부담이 됨
    - 공통의 목적이 서비스를 잘 만들어 신속하게 비즈니스 요구에 부응하는 것으로 바뀌면 개발 부문이 새로운 기능을 만들고 이를 운용 부문과 연계하여 서비스에 투입하며 계속적으로 서비스를 지속시키는 것이 서로에게 자연스럽게 될 것 이며 또한 개발 부문도 운용 부문도 어떻게 구현할 수 있을까 생각하게 될 것
    - 공통의 목표를 향하여 서비스를 잘 만든다고 해도 서로의 일을 모른 채 임의적으로 기능을 추가하는 개발이나 지속적인 운용을 위한 임의적인 정비를 해서는 안됨
    - 개발과 운용이 서로가 지향하는 것을 받아들여 어떠한 변화가 서로에게 어떤 부담이 되는가 어떻게 하는 것이 서로에게 더 좋은 방법이고 더 나은 서비스 개선이 되는가를 공감할 필요가 있음
    - 공감을 얻는 데 있어서 개발 부문과 운용 부문이 서로 움직이길 기다리고 있으면 개선할 수 없음
    - 운용 부문이 개발 부문의 의뢰를 기다리는 것이 아니라 또한 개발 부문이 운용 부문의 지적을 기다리는 게 아니라 서로가 자율적으로 움직여야만 공통의 목표 달성에 접근할 수 있음

# DevOps

- ❖ 조직과 DevOps
  - ✓ DevOps가 대응해야 하는 조직 및 팀의 과제
    - ● 속인성 배제
      - ◆ 속인성 있는 작업이란 어떤 특정한 사람에게 크게 의존하는 작업을 의미
      - ◆ 그 사람만 아는 것 또는 그 사람이 없으면 곤란한 것은 DevOps의 개발과 운용 간 긴밀한 연계에 저해 요인이 되기 때문에 배제하지 않으면 안됨
      - ◆ 운용이라는 관점에서 보았을 때 속인성이 있는 것은 치명적인데 특정한 사람밖에 모르는 설정 방법이나 Deploy 방법이 있다거나 특정한 사람밖에 모르는 서비스가 변경되고 있다면 팀으로서의 정보는 매우 취약하고 지속적인 서비스 제공에 위협이 됨
      - ◆ 이러한 지식의 공유가 안 되는 문제에 대한 해결책으로는 서버와 스토리지, 네트워크 등 인프라 자원의 설정 순서를 정리한 구축 절차서에 모든 정보를 기록하는 것 외에는 다른 수단이 없었지만 지금은 구성 관리 도구로 구축 절차를 가시화함으로써 이런 문제를 해결할 수 있는데 Code이기 때문에 프로세스상의 조건 분기도 모두 표현할 수 있으며 지금까지 사람이 판단하여 작업하던 프로세스는 어떤 사람이라도 반복적으로 실행할 수 있게 됨
      - ◆ 인프라 구성 관리 도구 Ansible의 개발 팀장인 미첼 드한이 말하는 구성 관리 도구: 이 회의실에 있는 모든 사람이 간단하게 Rolling Update 할 수 있도록 하는 도구

# DevOps

- ❖ 조직과 DevOps
  - ✓ DevOps가 대응해야 하는 조직 및 팀의 과제
    - ● 속인성 배제
      - ◆ 롤링 업데이트(Rolling Update)란 복수의 컴포넌트로 이뤄진 시스템에서 부분적으로 컴포넌트를 업데이트 해나감으로써 시스템을 정지시키지 않고 시스템 전체 업데이트를 도모하는 방법
      - ◆ 일반적으로 롤링 업데이트를 하기 위해서는 고도의 지식을 필요로 하는 등 전문성이 강하기 때문에 의견 공유가 어려우며 팀으로서 대응도 쉽지 않은 일이었지만 구성 관리 도구로 인해 그러한 작업에서조차도 팀 누구나가 실시할 수 있게 되는 등 속인성 배제가 가능

# DevOps

- ❖ 조직과 DevOps
  - ✓ DevOps가 대응해야 하는 조직 및 팀의 과제
    - ● 팀 간의 오버헤드 삭감
      - ◆ 서버팀, 네트워크팀, 스토리지팀처럼 여러 팀이 관계하고 있는 경우 틀림없이 그 조직은 서비스 개선에 대한 커다란 오버헤드를 가지고 있는 상태
      - ◆ 전문적인 팀으로 구성된 조직에서는 팀 간의 거래에 오버헤드가 크기 때문에 팀 간 전달을 위해 문서를 만들거나 승인 행위가 발생하거나 또한 각 팀에서 소요 기간 산정 시 버퍼를 설정하는 등 불필요한 기간이나 작업이 대량으로 발생하게 됨
      - ◆ 개발팀과 운용팀의 긴밀한 연계에서는 이러한 오버헤드를 제거하고 함께 요건을 받아 의논하고 크로스 체크하며 서로의 작업을 알게 된 상태에서 서비스 개선을 진행시키는 관계를 목표로 하는데 친밀한 연계를 도와주는 도구는 다양한데 예를 들어 버전 관리 도구 나 인프라 구성 관리 도구를 이용하는 경우 인프라를 코드화함으로써 개발팀과 운용팀에서 보게 되는 구성 정보가 일원화되고 공유되며 이로 인해 전문성이 배제됨으로써 인력을 유연하게 활용할 수 있어서 조직 구성을 간략하게 하는 데 도움이 될 수도 있음
      - ◆ 채팅 도구나 공통의 이슈 트래킹 시스템 등의 도입으로 개발 부문과 운용 부문의 활동 내역, 구성 관리 정보, 시스템 상황 등 모든 정보가 일원화되어 팀 간 정보 전달 오버헤드가 줄어들어 더욱 신속하게 개선 정책을 내놓을 수 있는 체제를 구축할 수 있게 됨

# DevOps

- ❖ 조직과 DevOps
  - ✓ DevOps가 대응해야 하는 조직 및 팀의 과제
    - ● 품질을 높임
      - ◆ 가정
        - ▪ 기획팀의 지시로 개발팀이 인프라 운용팀에게 얘기하지 않고 어떤 캠페인 기능을 포함시켰다고 가정하면 실제 장비에 설정된 기능에는 엑세스 숫자 이상으로 서버 자원을 낭비하는 문제가 발생할 가능성이 있음
        - ▪ 서버의 부하가 급격히 증가하면서 서비스가 전혀 동작하지 않게 됨
        - ▪ 운용팀은 수정된 사실을 모르고 있었기 때문에 액세스 수는 그다지 변화하지 않았는데 급격히 서버의 부하가 높아진 현상을 발견해도 그 이유를 알 수 없음
        - ▪ 로그를 추적하여 URL을 특정해도 캠페인 기능은 정상적으로 톱 페이지에 포함되어 있었기 때문에 해결책을 찾을 수 없음
        - ▪ 최종적으로 운용팀은 다양한 방안을 강구하면서 어떻게든 원인을 찾아내기 위해 개발팀에 연락을 취해야만 하는데 이로 인해 버그가 수정되고 해소되기까지 많은 시간이 걸렸고 그 사이 서비스는 전혀 동작하지 않음
      - ◆ 개발팀과 운용팀이 Release 시점, Release 내용 Release로 인하여 발생하는 영향 범위 등을 파악하여 함께 모니터링했다면 보다 빠른 단계에서 원인을 찾아 버그를 수정할 수 있음

# DevOps

- ❖ 조직과 DevOps
  - ✓ DevOps가 대응해야 하는 조직 및 팀의 과제
    - ● 품질을 높임
      - ◆ 장애 뿐 만 아니라 성능에 대해서도 마찬가지인데 개발 팀은 어플리케이션만을 운용 팀은 인프라 운용 만을 생각하는 상태에서 개발 및 구축한 결과 서로에게 기대하는 Connection Pool 숫자가 맞지 않아 성능이 나오지 않는 경우도 있음
      - ◆ 함께 서비스를 설계하는 것은 매우 중요하며 액세스 숫자에 대한 예상 그것에 대응하기 위한 미들웨어, 네트워크 설정, 서버 수, 데이터베이스 사용법 검토 등 생각해야 하는 것이 많음
      - ◆ 이것을 개발팀이 요구사항 정의서에 기재한 후 운용팀에게 전달하여 대응하도록 해도 결국 중요한 어플리케이션 요건을 모르는 상태에서 개발팀이 마음대로 설정 값을 기재하게 되면 결과가 좋지 못하게 됨
      - ◆ 각자의 전문 영역을 알고 있는 상태에서 설정하는 것이 중요한데 장애가 발생했을 때 분석의 기본이 되는 것은 로그이지만 어플리케이션 개발자는 자신에게 필요한 디버깅 정보만을 디버깅 차원에서 출력하고 있으며 운용 팀은 운용 팀에서 로그 용량을 계산하고 로그 수준을 경고 레벨로 적용해 버리기 때문에 장애 분석 단계에서 로그에 무엇이 일어났는지 전혀 나타나지 않아 장애 분석을 할 수 없는 경우도 있음
      - ◆ Web 서비스나 스마트 폰 어플리케이션 개발, 게임 개발 등 다양한 개발에서 DevOps의 체계를 갖추면 운용 팀과 개발 팀이 지식을 교환하고 설계하며 서로의 담당 업무를 깊이 이해하게 되어 성과물의 품질을 보다 향상시킬 수 있음

# DevOps

- ❖ 조직과 DevOps
  - ✓ 콘웨이의 법칙
    - 1967년에 컴퓨터 과학자이면서 프로그래머인 멜빈 콘웨이(Melvin Conway)가 조직과 시스템 구성의 관계를 나타내는 법칙
    - 시스템을 설계하는 조직은 그 조직의 의사 소통 구조를 그대로 복사한 설계를 하게 됨
    - 이 법칙에 따르면 한 개의 조직에 개발팀, 서버팀, 네트워크팀, 스토리지팀처럼 여러 팀을 구성하게 되면 시스템도 각 파트마다 전문성이 높게 기능이 분리된다는 것
    - 전문성이 높은 팀은 오버헤드가 매우 커지며 이 오버헤드를 만들어 내는 조직 구성에 대해서 시스템 설계가 먼저일까, 조직 설계가 먼저일까에 대한 논란이 있을 수 있음
    - 개발과 운용을 긴밀하게 연계하는 DevOps의 사고방식과 DevOps를 뒷받침하는 도구를 도입하면 시스템 구성과 조직의 위상을 동시에 변화시킬 수 있어 조직에 좋은 영향을 줌