도버/쿠버네틱스 실전활용

2023년 1월 6일 금요일 오후 3:53

도버/쿠버네티스 실전활용하기

230106, 김충섭 강사님(http://subicura.com , 깃허브 purple.io)
배포시기에 활용할 수 있는 도구(현업에서도 많이 사용하니까 잘 듣자!)
오늘 목표 : DevOps가 뭔지, 컴퓨터에서 개발한 것을 배포 하려면 서버를 관리할 줄 알아야 한다. 서버를 관리한다는 게 뭔지, 그리고 서버를 관리할 때 요새는 도커와 쿠버네티스를 많이 사용하기 때문에 이게 뭔지와 활용법에 대해 알려줄 예정!

DevOps를 해야하는 이유

- Development(개발) + Operations(운영)의 합성어
- 소프트웨어 개발자와 정보기술 전문가 간의 소통, 협업 및 통합을 강조하는 개발환경이나 문화
- 소프트웨어 개발 조직과 운영조직간의 상호 의존적 대응
- 단순히 개발과 운영을 같이 하는 것이 아니라, **조직이 소프트웨어 제품과 서비스를 빠른 시간에 개발 및 배포하는 것이 목적**

개발 프로세스

- **Lv.1 개발만 잘 하면 되는 시기 (개발하고 배포!)** : Developer(코드 작성) → Server(Cloud)
- **Lv.2 협업을 시작하는 시기(소스 관리)** : Developer(코드 작성) → Git(Merge Request or Push) → Server(Cloud)
- **Lv.3 배포에 자동화를 도입하는 시기(자동 빌드, 자동 배포)** : Developer(코드 작성) → Git(Merge Request or Push) → CI/CD(Test Build Deploy) → Server(Cloud) CI/CD가 소스를 자동으로 다운 받아서 빌드하고 특정 서버에 배포까지 해줌
- **Lv.4 롤백 등 배포 전략을 확장하는 시기(여러개의 배포 스테이징)** : Developer(코드 작성) → Git(Merge Request or Push) → CI(Test Build) → CD(Deploy) → Server(Cloud)
- **Lv.5 운영이 중요해지는 시기(모니터링)** : Developer(코드 작성) → Git(Merge Request or Push) → CI(Test Build) → CD(Deploy) → Server(Cloud) → Monitoring(Metrics Log) 인기가 너무 많아져서 서버가 다운되고 이럴경우,,, 모니터링이 필요

개발과 배포는 생각보다 복잡하다.

- 개발팀의 규모에 따라 다름
- 서비스의 복잡도에 따라 다름

→ DevOps를 하자
DevOps = 어떻게 하면 우리가 더 빨리 할 수 있을까! 를 고민해보는 철학과 방법론, 문화(단순히 도커, 쿠버네티스가 좋더라~가 아니라 이런 도구를 사용하면 더 빨라지는지를 고민해

봐야함)

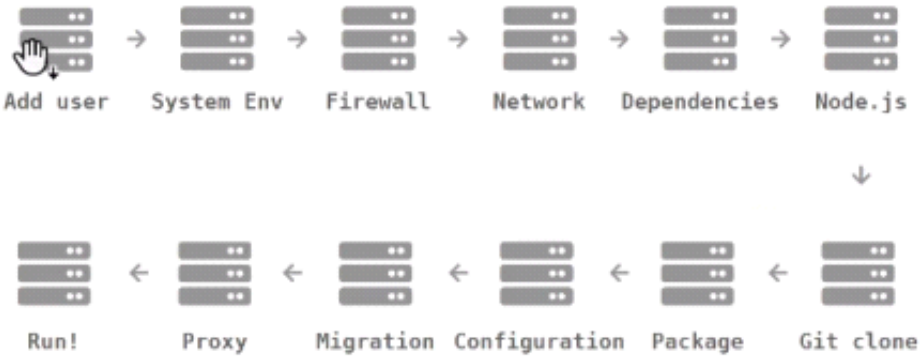
DevOps 도구

- DevOps ≠ 도커와 쿠버네티스를 잘 사용하는것 X, 좋은 도구를 잘 사용해서 조금 더 빠르고, 효율적으로 자동화하고 공유하고 축적하는 것!
- 새로운 기술을 배우는 이유는 개발과 배포를 개선하기 위함임!
- 개선되지 않으면 이 기술을 도입하는 것에 대해 재검토해봐야 함!

서버를 관리한다는 것

자체 서버 운영 → 설정관리도구 등장 → 가상머신 등장 → 클라우드 등장 → Paas 등장 → 도커 등장 → 쿠버네티스 등장 → 서비스메시 등장
서버의 상태를 관리하기 위한 노오오오력

자체 서버 운영



Node.js 예시
→ 문서로 관리하려니까 너무 힘들어~!!!

상태관리 도구 등장

→ 배우려면 너무 힘들고 한 서버에 다른 버전 여러개 설치 못함

가상머신 등장

→ 처음부터 다시 세팅하려면 힘들고, 서버 이미지 공유 하기도 힘들고, 느려!

클라우드 등장

AWS, Google Cloud, Azure, ...
하드웨어 파편화 문제 해결
가상화된 환경만으로 아키텍처 구성이 가능해짐
이미지를 기반으로한 다수의 서버 상태 관리
- 상태관리에 대한 새로운접근
- 서버 운영의 문제는 여전히 그대로 남아있음
마치 전기를 사용하듯 편리

Paas 등장

Vercel, Heroku, Netlify, AWS Elastic Beanstalk, Google Cloud App Engine, ...
서버를 운영하는 것은 복잡하고 어렵다
잘 구성해 놓은 곳에 소스 코드만으로 배포
일반화된 프로비저닝 방법을 제공
- 프로비저닝 과정에 개입할 수 없음
- 예) Heroku의 buildpacks
PaaS는 서버 운영의 은총알?

Paas 단점

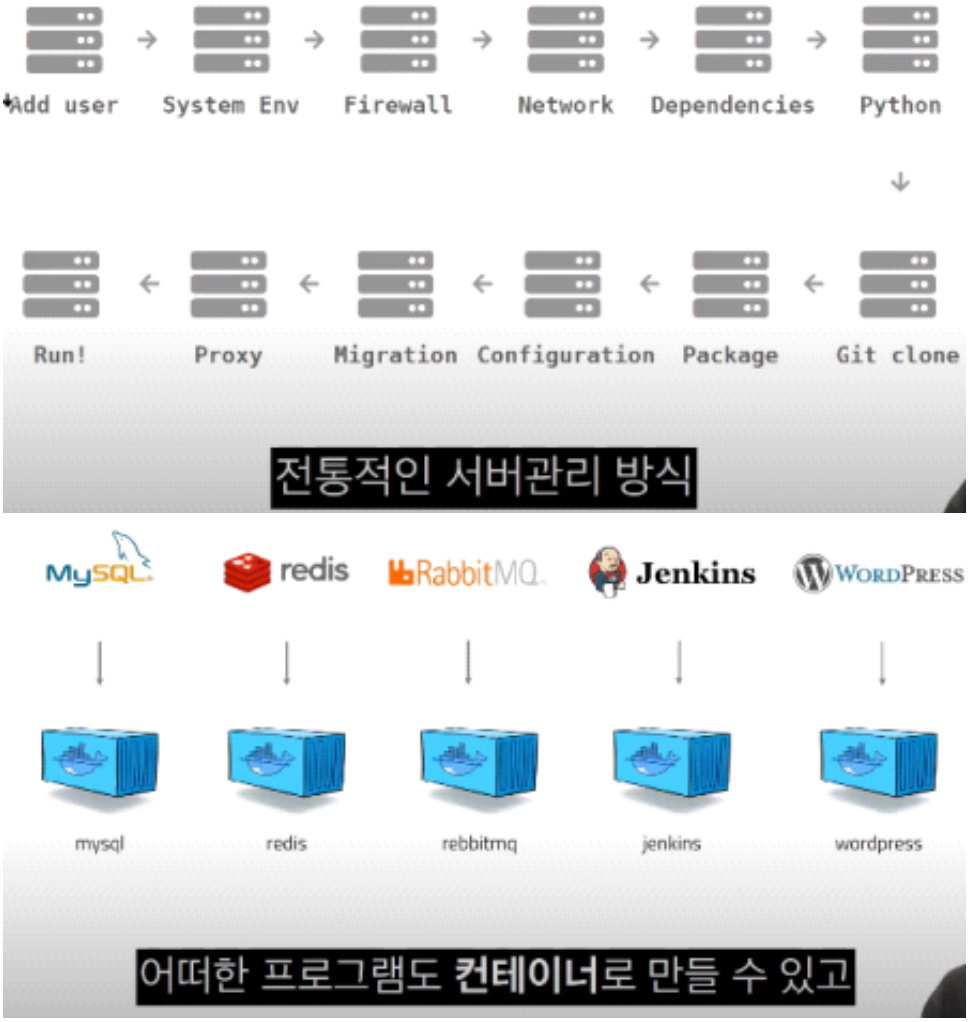
애플리케이션을 PaaS 방식에 맞게 작성해야함
서버에 대한 원격 접속 시스템을 제공하지 않음
서버에 파일 시스템을 사용할 수 없음
서버 패키지를 설치할 수 없음
로그 수집을 제한적인 방식으로 허용 (STDOUT)
애플리케이션 배포에 대한 새로운 패러다임

이 중 강사님이 가장 추천하는 루트

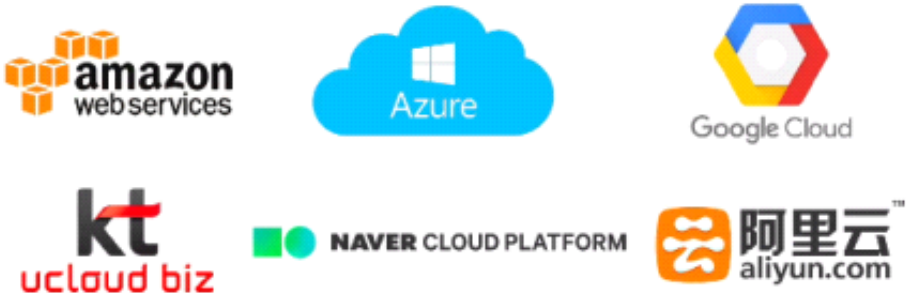
PaaS를 사용하다가 사용자가 너무 많아지거나, 새로운 기능이 필요할 때 클라우드로 넘어가는것을 추천!

도커와 쿠버네티스의 등장

도커란 무엇인가

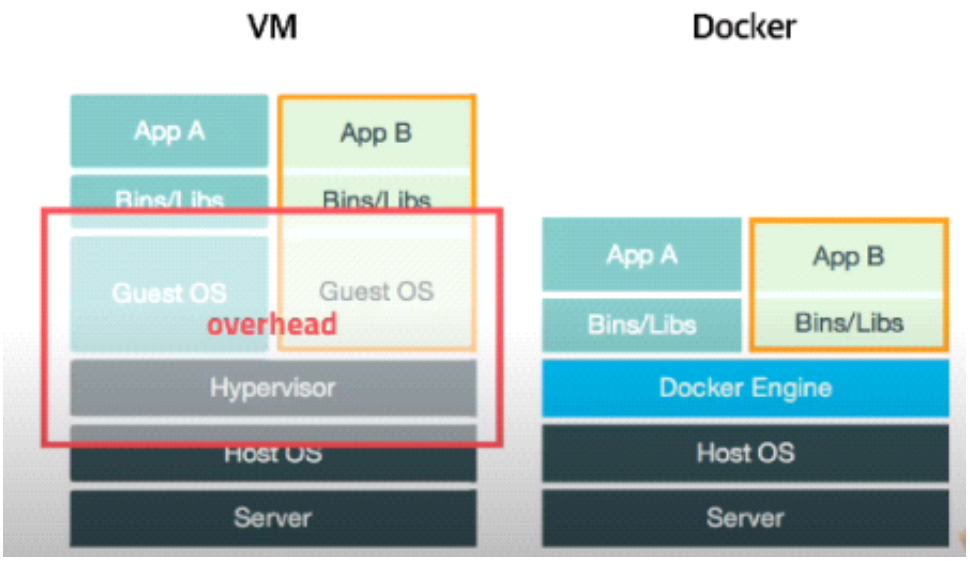


→ 컨테이너만 실행하면 해당 프로그램이 실행되는 시스템



- 가상머신처럼 독립적으로 실행되지만, 가상머신보다 빠르고, 쉽고, 효율적!
- '자원 격리'가 핵심 기능! : 이 프로그램을 띄울 때 다른 프로그램이 간섭하지 못하게 하는 역할
  - 프로세스/파일, 디렉토리 → 가상으로 분리
  - CPU, Memory, I/O → 그룹별로 사용할수 있게 제한
  - 리눅스 기능을 이용해 빠르고 쉬운 서버 관리 → 도커로 쉽게 가능해짐!

VM(가상머신) vs Docker



도커가 가져온 변화

- 클라우드 이미지보다 관리하기 쉬움
- 다른 프로세스와 격리되어 가상머신처럼 사용하지만 성능저하 (거의) 없음
- 복잡한 기술(namespace, cgroups, network, ...)을 몰라도 사용할 수 있음
- 이미지 빌드 기록이 남음
- 코드와 설정으로 관리 > 재현 및 수정 가능
- 오픈소스 > 특정 회사 기술에 종속적이지 않음

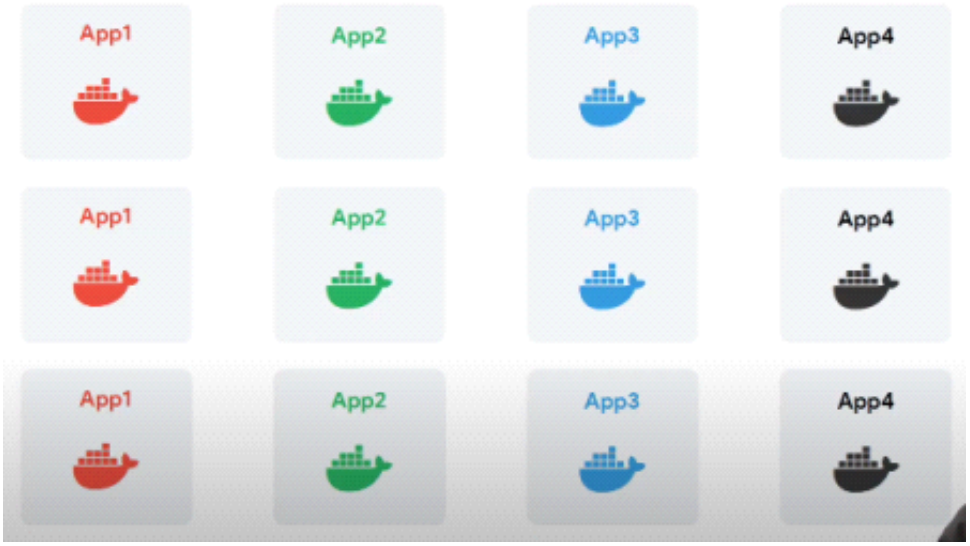
→ 물류에서 '컨테이너'의 등장과 비슷한 혁신을 가져옴!

도커의 이미지와 허브 개념

안드로이드의 apk 파일 같은 느낌! 안드로이드에서 apk 파일이 있으면 앱을 설치할 수 있는 것 처럼, 리눅스에서 앱을 실행할 때 필요한 것들을 다 모아놓은 압축된 파일을 이미지라고 한다. 그래서 이 이미지만 있으면 어디서든 들고와서 실행을 할 수 있다! 실행시키면 그걸 컨테이너라고 한다. 내 pc에서 만든 이미지를 실제 배포되는 서버에서 사용하려면 이미지를 어딘가에 저장해 두어야 한다. 그 이미지가 저장되는 곳이 '도커 허브'이다.

+) AWS에도 도커허브와 거의 동일한 ECL이라는 개념이 있음

도커 그 이후



💡 도커 사용하면 App 실행만 하면 되서 편하긴 한데,,, 이렇게 앱이 많아지면 어떡하지?

- 서버마다 도커를 실행하고 멈춰야 하는 상황 → 귀찮음
- 어떤 서버가 여유 있는지 확인하기 힘들다
- 버전 업데이트와 다운그레이드가 쉽지 않다.
- 서비스 이상이나 부하 모니터링을 하는 것도 쉽지 않음

→ 자동으로 관리해줄 수 있는 방법이 없을까?

Container Orchestration

복잡한 컨테이너 환경을 효과적으로 관리하기 위한 도구! 개발자 대신 일해줄 자동화도구

- Cluster : 중앙제어 / 노드 스케일
- State : 상태관리
- SCHEDULING : 배포관리
- ROLLOUT/Rollback : 배포 버전관리
- Service discovery : 서비스 등록 및 조회
- volume : 볼륨 스토리지

→ 지금은 쿠버네티스가 평정함!

쿠버네티스

컨테이너를 쉽고 빠르게 배포/확장하고 관리를 자동화해주는 오픈소스 플랫폼