**1장 시작하기 전에**

***1-1* 컨테이너가 IT 세상을 점령한 이유**

**1.1.1 클라우드 환경으로 이주하기**

**애플리케이션의 클라우드 환경 이주**

* 의미: 서버, 스토리지. 네트워크 심지어 전원까지 마이크로소프트나 아마존. 구글에 맡겨버릴 수 있다
* 장점
  + 무제한의 확장성을 누릴 수 있음
  + 아주 빠르게 새로운 환경에 애플리케이션을 배포할수도 있음
  + 비용도 실제 사용한 만큼만 부담하면 됨
* 방법
  + 기존:
    - 서비스로서의 인프라(laaS)
      * 장점: 각 컴포넌트를 가상 머신에서 동작 (클라우드에 종속 x)
      * 단점: 운영비 상승
    - 서비로서의 플랫폼(paas)
      * 장점: 운영비 절감 가능
      * 단점: 우리 애플리케이션의 각 컴포넌트를 하나씩 클라우드의 매니지드 서비스(managed service)로 옮기는 까다로운 프로젝트를 진행해야 함, 애플리케이션이 특정 클라우드에 종속
  + 도커: 애플리케이션의 각 컴포넌트를 컨테이너로 이주한 다음 애저 쿠버네티스 서비스나 아마존 일래스틱 컨테이너 서비스 혹은 직접 구축한 도커 클러스터에서 전체 애플리케이션을 실행할 수 있다
    - 장점: 특정 클라우드에 종속되지 않으므로 원하는 클라우드 서비스나 데이터센터. 심지어 로컬 환경에서도 운영 가능, 낮은 운영비와 이식성 확보

컨테이너로 애플리케이션을 이주하려면 어느 정도 비용이 필요

* 기존 설치 절차를 Dockerfile이라는 스크립트로 재작성
* 배포 관련 사항 역시 도커 컴포즈나 쿠버네티스에서 사용되는 애플리케이션 매니페스트로 재작성
* 코드를 수정할 필요는 없음
* 컨테이너화된 애플리케이션은 업무용 노트북부터 클라우드까지 어떤 환경에서든 기존과 동일한 기술 스택에서 그대로 동작

**1.1.2레거시 애플리케이션 현대화하기**

모놀리식 설계 -> 도커 혹은 클라우드 플랫폼의 동작에 기민성에 제약이 따름

도커로의 이주 과정

1. Dockerfile 스크립트와 도커 컴포즈 문법을 따라 애플리케이션을 **단일 컨테이너**로 옮김

* **컨테이너**는 가상 네트워크를 통해 외부에 노출되지 않고 서로 통신할 수 있다

1. **애플리케이션을 분할**해 기능별로 별도의 컨테이너에 배치할 수 있음
2. 여러 개의 컨테이너로 분할된 **분산 애플리케이션**으로 거듭나게 됨

이렇게 새로운 설계로 거듭난 애플리케이션은 마이크로서비스 아키텍처의 다양한 장점을 누릴 수 있음

ex) 핵심 기능을 작고 독립된 단위로 만들어 따로따로 다루면서 변경 내용을 빠르게

테스트, 해당 기능의 확장성을 조절, 필요에 맞는 적절한 기술 기반을 선택 가능

* 도커를 도입하면 레거시 애플리케이션의 설계를 쉽게 현대화할 수 있다

**1.1.3 클라우드 환경에 적합한 새로운 애플리케이션 개발하기**

도커는 분산 애플리케이션이든 모놀리식 설계든 기존 애플리케이션을 클라우드로 이주하는 데 유용

분산 애플리케이션을 빌드하고 실행하는 데는 별도의 개발 도구가 필요치않음 -> 도커를 설치하고, 소스 코드 저장소를 복제한 다음. 한 번의 명령으로 코드를 빌드하고 전체 애플리케이션을 실행할 수 있음

도커는 서드파티 소프트웨어를 도입하는 데도 유용

**도커 허브**: 다양한 사람이 자신이 작성한 컨테이너를 공유하는 서비스

CNCF(클라우드 네이티브 컴퓨팅 재단)는 모니터링부터 메시지 큐까지 원하는 용도에 적합한 오픈 소스 프로젝트의 목록을 제공 -> 이들 모두 도커 허브를 통해 자유롭게 사용 가능

**1.1.4 기술 혁신: 서버리스와 그 너머**

현대 IT 기술을 주도하는 요소 중 하나는 **일관성** -> 같은 도구, 같은 프로세스. 동일한 런타임

: **도커 클러스터**를 구축하면 모든 제품의 빌드, 배포, 운영을 같은 도구와 같은 방법으로 수행

도커 클러스터는 클라우드와 데이터센터 어디든 원하는 곳에서 운영

**서버리스 함수**: 곧 컨테이너 기술

서버리스 기술의 목표는 개발자가 함수코드를 작성하고 서비스에 푸시하면 서비스가 코드를 빌드하고 패키징하도록 하는 것이다. 함수사용 측에서 함수를 호출하면 서비스는 해당 함수의 인스턴스를 생성해 요청을 처리 -> 이 과정에는 빌드 서버도, 파이프라인도, 관리가 필요한 운영 환경도 필요 없음

**1.1.5 데브옵스도입하기**

운영자: 배시(Bash), 나기오스(Nagios), 파워셸(Powershell), 시스템 센터 등의 도구를 다룸

개발자:’ 메이크(Make), 메이븐Maven). 누겟(NuGet), MS빌드(MSBuild) 등의 도구를 주로 사용

-> 팀원 모두가 Dockerfile과 도커 컴포즈 스크립트를 사용하면 같은 기술과 도구로 팀을 통일

**CALMS**: 문화(Culture), 자동화(Automation), 린(Lean), 측정(Metric), 공유(Sharing)로 데브옵스를 위한 프레임워크