1장 새로운 인프라 환경이 온다 - 원영

`컨테이너 인프라 환경` : **컨테이너**를 중심으로 구성된 인프라 환경

`컨테이너(container)`

- **하나의 운영 체제 커널**에서 다른 프로세스에 영향을 받지 않고 **독립적으로 실행**되는 프로세스 상태를 의미
- 이렇게 구현된 컨테이너는 가상화 상태에서 동작하는 프로세스보다 **가볍고 빠르게 동작함**

1.1 컨테이너 인프라 환경이란

1.1.1 모놀리식 아키텍처 (monolithic architecture)

- **하나의** 큰 목적이 있는 **서비스/애플리케이션에 여러 기능**이 통합되어 있는 구조
- **소프트웨어가 하나의 결합된 코드로 구성됨** (초기단계 설계 용이/개발 단순/코드관리 간편)

⛔ **단점**

- 서비스를 운영하는 과정에서 수정이 많을 시, 다른 서비스에 영향을 미칠 가능성 高
- 서비스에 기능이 추가될수록 단순했던 서비스간의 관계가 복잡해질 수 있음
- **⇒ 해결방안 : 마이크로서비스 아키텍처**

1.1.2 마이크로서비스 아키텍처 (MSA, Microservices Architecture)

- 시스템 전체가 하나의 목적을 지향하는 바는 모놀리식 아키텍처와 동일
- 개별 기능을 하는 작은 서비스를 각각 개발해 연결하는 데서 차이를 보임
- 보안, 인증 등과 관련된 기능이 독립된 서비스를 구성하고 있고, 다른 서비스들도 독립적으로 동작할 수 있는 완결된 구조임

☑️ **장점**

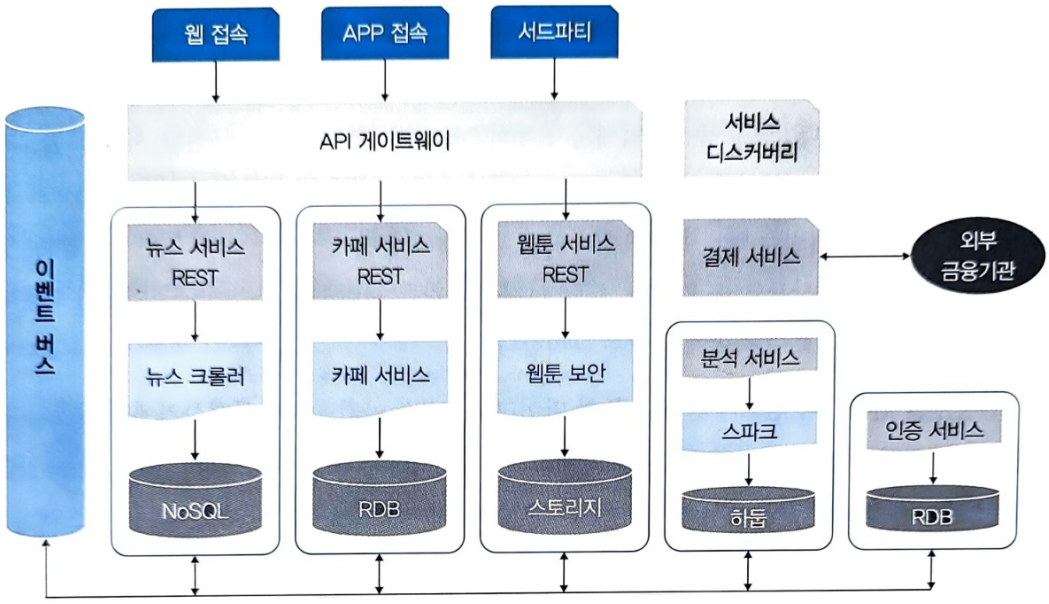
- **개발된 서비스 재사용이 쉬움**
- 향후 서비스 변경 시 다른 서비스에 **영향을 미칠 가능성 줄어듦**
- **사용량의 변화에 따라 특정 서비스만 확장 가능**
  (∴ 사용자의 요구사항에 따라 가용성을 즉각적으로 확보해야 하는 **IasS 환경에 적합**)

⛔ **단점**

- 모놀리식 아키텍처보다 복잡도가 높음
- 각 서비스가 서로 유기적으로 통신하는 구조로 설계되므로 네트워크를 통한 호출 횟수가 증가해 성능에 영향을 줄 수 있음

마이크로 서비스 아키텍처의 구성 예시

❤ 그림 1-2 마이크로서비스 아키텍처의 구성 예시



- 하나의 애플리케이션 안에 포함되었던 서비스들이
  각 서비스와 관련된 기능과 DB를 독립적으로 가지는 구조
- 각 서비스는 `API 게이트웨이`와 `REST API`를 이용한 통신 방식으로 사용자(외부)의 요청을 전달
- **서비스 개수는 고정된 것이 아니기 때문에**
  어떤 서비스가 등록되어 있는지 파악하기 위해 `서비스 디스커버리`를 사용함
- 수많은 서비스의 **내부 통신을 이벤트로 일원화**하고
  이를 효과적으로 관리하기 위해 **별도로 `이벤트 버스`를 서비스로 구성함**

☑️ 장점

- 각 서비스는 필요한 기능이 특화된 DB를 선택해 개별 서비스에 할당 가능
- 고객의 요구 사항에 따라 분석 서비스를 새로 추가해야할 시에도
  기존에 있는 이벤트 버스에 바로 연결하면 되므로 매우 유연하게 대응 가능
- 이미 개발된 기능이 다른 서비스에 필요하다면 바로 재사용할 수 있음

1.1.3 컨테이너 인프라 환경에 적합한 아키텍처

- 보통 중소기업에서 진행하는 소규모 프로젝트는 모놀리식 아키텍처를 선호하는 경향 有
- But, 소규모라도 마이크로서비스 아키텍처로 설계하면 유지보수 측면에서 매우 유리

💡 **우리가 공부할 것은?** ⇒ `컨테이너 인프라 환경`

- **마이크로서비스 아키텍처로 구현하기에 적합.**
- 컨테이너를 서비스 단위로 포장해 **손쉽게 배포/확장 가능**
- 컨테이너 인프라 환경에서 제공하는 컨테이너는
  마이크로서비스 아키텍처의 서비스와 1:1로 완벽하게 대응
  [서비스와 1:1로 결합되는 컨테이너]

API 게이트웨이	뉴스 서비스	스토리지
컨테이너	컨테이너	컨테이너

- 이후 소개할 도구들을 이용하면 도입, 설계 운용 비용이 감소, 생산성 향상

1.2 컨테이너 인프라 환경을 지원하는 도구

🔎 **컨테이너 인프라 환경의 구성**

- 컨테이너
- 컨테이너 관리
- 개발 환경 구성 및 배포 자동화
- 모니터링

⇒ 이를 지원하는 도구 가운데 업계에서 가장 많이 사용하는 도구 몇가지를 알아보자!

1.2.1 도커 (Docker)

- 컨테이너 환경에서 독립적으로 애플리케이션을 실행할 수 있도록
  컨테이너를 만들고 관리하는 것을 도와주는 컨테이너 도구

- 도커로 애플리케이션을 실행하면
  운영체제 환경에 관계없이 독립적인 환경에서 **일관된 결과를 보장함**

1.2.2 쿠버네티스 (Kubernetes)

- **다수의 컨테이너(e.g. 도커)를 관리하는 데 사용**
- **제공하는 기능**
  - 컨테이너의 자동 배포 기능
  - 배포된 컨테이너에 대한 동작 보증 기능
  - 부하에 따른 동적 확장 기능
- 컨테이너 인프라 필요한 기능을 통합하고 관리하는 솔루션으로 발전됨
- 컨테이너 인프라를 기반으로
  API 게이트웨이, 서비스 디스커버리, 이벤트 버스, 인증 및 결제 등의 **다양한 서비스를 효율적으로 관리할 수 있는 환경을 제공**하고 이를 **내외부와 유연하게 연결해줌**

1.2.3 젠킨스 (Jenkins)

- **지속적 통합(`CI`, Continuous Integration)과 지속적 배포(`CD`, Continuous Deployment) 지원**
- CI/CD
  - 개발한 프로그램의 빌드, 테스트, 패키지화, 배포 단계를 모두 자동화해
    개발 단계를 표준화함
  - 개발된 코드의 빠른 적용과 효과적인 관리를 통해
    개발 생산성을 높이는 데 초점이 맞춰져 있음
- 즉, 컨테이너 인프라 환경처럼 단일 기능을 빠르게 개발해야하는 환경에 매우 적합한 도구임

1.2.4 프로메테우스 (Prometheus)와 그라파나(Grafana)

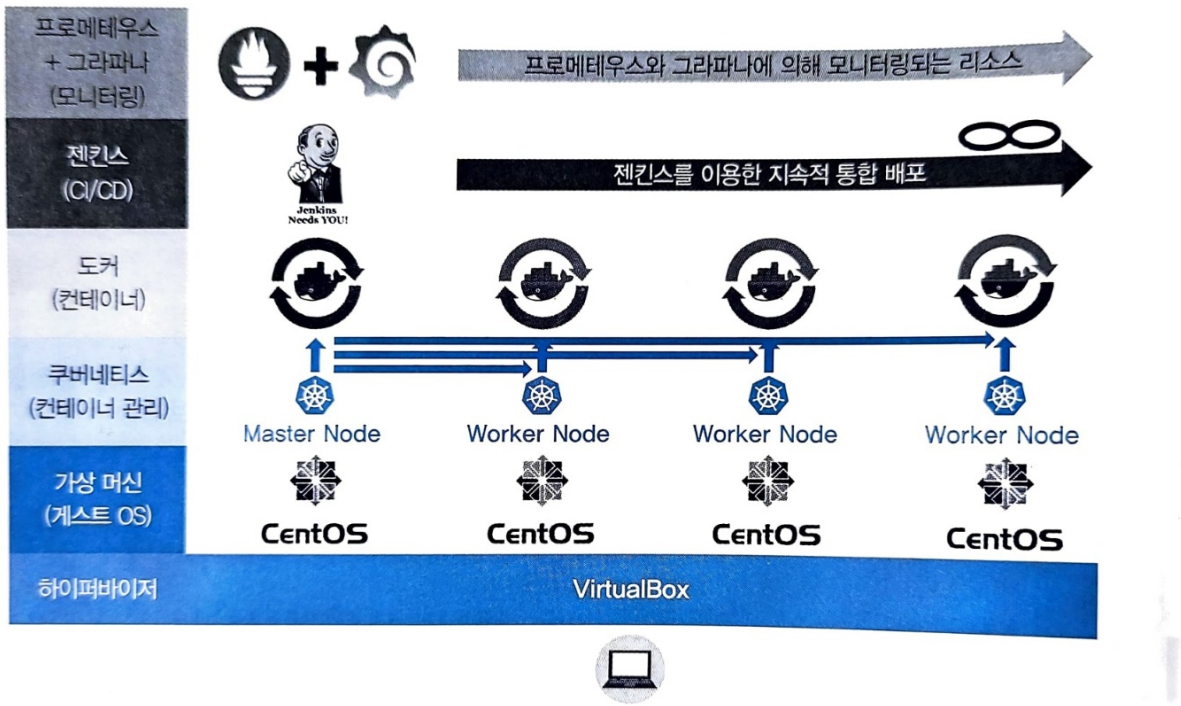
- **모니터링**을 위한 도구임
- `프로메테우스` : **상태 데이터를 수집**

- 그라파나 : 프로메테우스로 수집한 데이터를 관리자가 보기 좋게 **시각화**

- 컨테이너 인프라 환경에서는 많은 종류의 소규모 기능이 각각 나누어 개발되므로, **중앙 모니터링이 필요함** ⇒ ∴ 방법 中 1 - **프로메테우스 + 그라파나**
- 프로메테우스와 그라파나는 컨테이너로 패키징되어 동작하며 최소한의 자원으로 쿠버네티스 클러스터의 상태를 시각적으로 표현함

1.3 새로운 인프라 환경의 시작

그림 1-8 실습용 컨테이너 인프라 환경 구성



하이퍼바이저

- 가상화 기술을 사용하여 **하드웨어 리소스를 가상 머신(가상 환경)에 제공**하는 소프트웨어
- **물리적인 컴퓨터(호스트 머신) 위에 가상 머신(게스트 머신)을 생성하고 실행**하는 역할을 담당

- 하이퍼바이저는 가상 머신의 리소스 할당, 가상 네트워킹, 저장소 관리, 가상 디스크 관리 등의 작업을 처리하여 가상 머신이 독립적으로 운영 체제 및 응용 프로그램을 실행 가능
- **하이퍼바이저 유형 2가지**
  - **Bare-metal 하이퍼바이저**
    - 호스트 머신의 운영 체제 위에 직접 설치되는 하이퍼바이저
    - 하이퍼바이저 자체가 운영 체제로 작동하며 가상 머신을 관리
    - e.g.) VMware ESXi, Microsoft Hyper-V, Citrix XenServer 등
  - **Hosted 하이퍼바이저**
    - 호스트 머신의 운영 체제 위에서 실행되는 소프트웨어로, 일반적으로 응용 프로그램으로 설치
    - 호스트 운영 체제에서 가상 머신을 생성하고 관리
    - e.g.) Oracle VirtualBox, VMware Workstation, Microsoft Virtual PC