Ch 1. 시스템과 인프라 기초 지식

BOM KIM

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

INCHEON NATIONAL UNIVERSITY

1.1 시스템 기반의 기초 지식

- Docker
  - 애플리케이션 실행 환경을 작성•관리하기 위한 플랫폼
    - 플랫폼 = 여러가지 기능을 제공하는 공통 실행환경
- 시스템 기반
  - 애플리케이션이 동작하는 데 필요한 하드웨어와 OS, 미들웨어 등의 인프라

도커란 컨테이너 가상화 기술을 사용한 애플리케이션의 실행 환경을 생성하고 관리하는 툴이라고 할 수 있습니다.

개발한 앱을 릴리스하여 최종 사용자(엔드유저, End-User)가 사용하기 위해서는
시스템 기반을 구축하고 그 위에 애플리케이션 실행 환경을 올려야 하는 과정을 거쳐야지 사용 가능한데요

도커에 들어가기 앞서 시스템 기반을 다루기 위해 필요한 인프라 기술에 대한 내용을 먼저 공부하게 되었고
그 내용을 토대로 발표해보도록 하겠습니다.

1.1 시스템 기반의 기초 지식

- 클라우드가 등장하면서 시스템 개발 흐름에 변화
- 분산환경에서는 인프라 엔지니어가 수동으로 운용하지 않고 자동화된 틀을 사용해 오케스트레이션함
- 오케스트레이션 (Orchectration)
  - 컴퓨터 시스템과 애플리케이션, 서비스의 자동화된 설정/관리/조정을 의미.

클라우드가 등장하면서 시스템 개발 흐름에 변화를 불러왔습니다.

클라우드를 구성하는 대부분의 기술은 한 대의 물리 호스트 상에서 움직이는 시스템과는 달리
분산 환경에서 가동시키는 것이 기본인데요.

분산환경에서 엔지니어가 수동으로 운용하지 않고 자동화된 틀을 사용해 오케스트레이션 하도록 변화하였습니다.

1.1.1 시스템 기반의 구성 요소

- 시스템 요구사항

- 기능 요구사항 (Functional Requirement)
  - 시스템의 기능으로서 요구되는 사항.

- 비기능 요구사항 (non-function Requirement)
  - 기능 요구사항 이외의 모든 요구사항

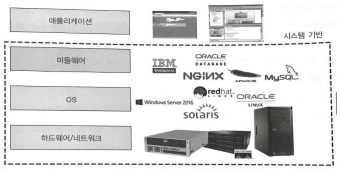
시스템에서 요구되는 사항은 크게 2가지로 나눌 수 있습니다.
기능 요구사항과 비기능 요구사항인데요,

기능 요구사항은 시스템의 기능으로서 요구되는 사항을 가리킵니다.
시스템이나 소프트웨어에서 어떤 기능을 하는지, 무엇을 할 수 있는지를 모아놓은 것을 의미하고요,

비기능 요구사항은 기능 요구사항 이외의 모든 요구사항으로 시스템의 성능, 신뢰성, 확장성, 운용성, 보안 등과 같은 요구사항을 가리킵니다.

1.1.1 시스템 기반의 구성 요소

- 인프라의 기본 구성 요소
- 하드웨어 (Hardware)
  - 시스템 기반(인프라)을 구성하는 물리적 요소



- 네트워크 (Network)
  - 시스템 사용자가 원격지에서 서버로 액세스할 수 있도록 연결하기 위한 요구사항

- OS (Operating System)
  - 하드웨어, 네트워크 장비를 제어하기 위한 기본 소프트웨어
  - 클라이언트 OS (Client OS) : Windows, Androiad, MacOS 등
  - 서버 OS (Server OS) : Linux Server, Unix Server, Windows Server 등

- 미들웨어 (Middleware)
  - 서버 OS 상에서 서버가 특정 역할을 다하기 위한 기능을 갖고 있는 소프트웨어
  - 상용 소프트웨어 / 오픈 소스

시스템 기반의 구성 요소는 앱을 실행하기 위한 기반이 되는 부분이고
하드웨어는 서버 장비 본체, 스토리지, 전원 장치 등 인프라를 구성하는 물리적 요소입니다.

네트워크는 라우터, 스위치, 방화벽, 케이블 배선 등 사용자가 원격으로 접근할 수 있도록 서버를 연결하는 도구들 입니다.

OS는 하드웨어와 네트워크 장비를 제어하기 위한 기본적인 소프트웨어로
클라이언트 OS와 서버 OS 두 가지 유형으로 나눌 수 있습니다.

마지막으로 미들웨어는 서버 OS에서 서버가 특정 역할을 수행할 수 있도록 동작하는 소프트웨어를 말하는데요,
상용 소프트웨어와 오픈 소스로 나뉘고 인프라를 설계할 때 어떤 미들웨어를 어떻게 사용할지 선택하는 것이 중요합니다.

•클라이언트 OS : 사용자가 쉽게 사용할 수 있도록 GUI 기능과 멀티미디어 기능 탑재. ex) Windows, Android, Mac OS 등
•서버 OS : 장시간 구동에도 문제가 없도록 만들어져 있으므로 시스템을 빠르고 안정적으로 운용하는 데 도움. 대량 처리를 효율적으로 수행할 수 있도록 도움. ex) Windows Server, Unix 서버, Linux 서버 등

클라이언트 운영 체제는 컴퓨터 데스크톱 및 랩톱 및 스마트 폰과 같은 기타 휴대용 장치 내에서 작동하는 시스템입니다. 프린터, 모니터 및 카메라와 같이 다양한 하드웨어 구성 요소를 관리 할 수 있습니다.
***
이러한 운영 체제는 한 번에 한 명의 사용자 만 지원합니다. 클라이언트 운영 체제는 서버 운영 체제에서 서비스를 가져올 수 있습니다. 또한 클라이언트 운영 체제는 서버 운영 체제와 비교할 때 최소 가격으로 다중 처리 성능을 제공합니다. Windows, Mac 및 Android는 클라이언트 운영 체제의 몇 가지 예입니다.

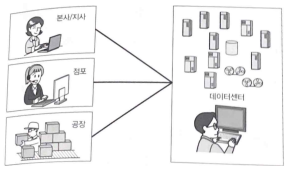
서버 운영 체제는 서버에서 작동하는 운영 체제입니다. 이것은 운영 체제의 고급 버전으로서 다른 장치 나 연결된 클라이언트 시스템에 다양한 서비스를 제공 할 수 있는 더 많은 기능을 제공합니다.

서버 운영 체제에서는 대부분의 프로세스가 OS 명령에서 실행됩니다. 고급 하드웨어, 소프트웨어 및 네트워크 구성 서비스가 있습니다. 이러한 운영 체제는 비즈니스 및 웹 응용 프로그램을 설치 및 배포하는 데 도움이됩니다. 또한 연결된 클라이언트 장치를 관리하고 모니터링 할 수 있습니다. 또한 서버 운영 체제는 다른 관리 프로세스를 실행합니다.

클라이언트 운영 체제는 데스크톱 및 기타 다양한 휴대용 장치에서 작동하는 운영 체제 인 반면 서버 운영 체제는 서버에 설치되고 사용되도록 설계된 운영 체제입니다. 따라서 이것은 클라이언트와 서버 운영 체제 간의 주요 차이점입니다.
***
또한 클라이언트 운영 체제는 서버에서 서비스를 얻을 수있는 반면 서버 운영 체제는 여러 클라이언트 또는 최종 사용자 장치에 서비스를 제공 할 수 있습니다

1.1.2 시스템의 이용 형태

- 온프레미스(on-premises) 방식
  - 시스템 구축에서부터 운영까지 자사에 데이터센터를 두고 직접 관리/수행하는 형태



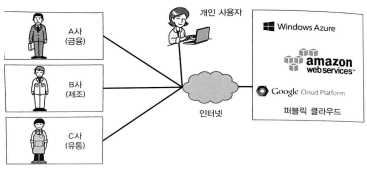
  - 전통적이고 지금도 널리 사용
  - 하드웨어,OS,미들웨어도 모두 자사에서 구입하여 라이선스 관리와 버전 업그레이드를 실행
  - 초기 시스템 투자 비용 부담이 큼
  - 시스템 사용량과 관계 없이 시스템 구동 후 운영에 드는 비용이 지속적으로 나감

시스템의 이용 형태에 대해 살펴보겠습니다.
온프레미스는 데이터 센터나 서버실에 서버를 두고 직접 관리하는 방식을 말합니다.
메인 프레임 시대부터 지금까지 대부분 기업에서 적용해 온 형태이기도 합니다.
직접 관리할 수 있고 기밀성이 높다는 장점도 있지만,
갖춰야 할 장비들이 상당히 고가이고 적지 않은 만큼 초기 시스템 투자 비용이 많이 들고
시스템 이용량과 상관없이 일정 금액을 지속적으로 부담해야 하는 것이 특징입니다.

서버나 네트워크 장비를 모두 자사에서 직접 조달하여 시스템 요구사항에 맞춰 인프라를 구축하고, 자사에서 운용하는 형태입니다.

1.1.2 시스템의 이용 형태

- 퍼블릭 클라우드(Public Cloud) 방식
  - 인터넷을 통해 불특정 다수에게 제공하는 클라우드 서비스



  - 인프라 관련 초기 투자가 필요 없음
  - 시스템 이용량이 증가했을 때 인프라 기반을 쉽게 증설 시킬 수 있음.
  - 제공하는 서비스에 따라 *IaaS, *PaaS, *SaaS 등으로 나뉨
    - 이용 시간과 데이터 양에 따라 요금 지불

퍼블릭 클라우드는 인터넷을 경유해 불특정 다수에게 제공되는 클라우드 서비스 입니다.

자사에서 데이터 센터를 보유하지 않기 때문에 초기 투자가 필요 없고
이용 시간이나 데이터 양에 따라 요금을 지불하면 됩니다.

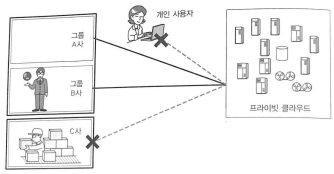
※ IaaS(Infrastructure as a Service) : 서버, 스토리지, 네트워크를 가상화 환경으로 만들어, 필요에 따라 인프라 자원을 사용할 수 있게 서비스를 제공하는 형태. OS부터 상위의 모든 플랫폼이나 어플리케이션을 사용자가 직접 올릴 수 있음.

※ PaaS(Platform as a Service) : 클라우드 컴퓨팅 서비스의 분류의 하나로서, 일반적으로 앱의 개발 및 시작과 관련된 인프라를 만들고 유지보수하는 복잡함 없이 고객이 애플리케이션을 개발, 실행, 관리할 수 있게 하는 플랫폼을 제공

※ SaaS(Software as a Service) : "on-demand software"로도 불리며, 소프트웨어 및 관련 데이터는 중앙에 호스팅되고 사용자는 웹 브라우저 등의 클라이언트를 통해 접속하는 형태의 소프트웨어 전달 모델

1.1.2 시스템의 이용 형태

- 프라이빗 클라우드 (Private Cloud) 방식
  - 특정 기업 그룹에만 제공하는 클라우드 서비스
  - 온프레미스와 퍼블릭 클라우드의 중간 형태
  - 이용자를 한정할 수 있으므로 보안을 확보하기 쉬움
  - 독자적인 기능이나 서비스를 추가하기 용이
  - 클라우드 업체가 제공하는 요구 사항에 따라 단기간에 인프라 구축 가능
  - 사용하는 만큼만 요금 지불



프라이빗 클라우드는 퍼블릭 클라우드 방식에서 이용자를 한정한 방식이라고 보면 됩니다.

이용자를 한정할 수 있기 때문에 보안이 좋고 독자적인 서비스를 추가하기도 쉽니다.

어떤 형태를 선택해야 할지는 시스템의 요구사항에 따라 결정되고
클라우드 업체가 제공하는 서비스를 시스템 요구사항에 맞춰 조합하기 때문에
단기간에 인프라를 구축할 수 있습니다.

그리고 서비스를 이용한 만큼만 요금을 지불하면 되기 때문에 초기 투자 비용도 필요 없습니다.

하이브리드 클라우드라고 온프레미스와 클라우드를 함께 사용하는 형태도 있고 그런데
시스템의 이용형태는 여기까지만 보겠습니다.

1.1.3 클라우드에 적합한 케이스

- 트래픽의 변동이 많은 시스템
  - 시스템을 사이징하기 어려운 경우에는 트래픽 양에 따라 쉽게 증설할 수 있는 클라우드에서 구성하는 것이 좋음
  - 사이징(sizing) : 트래픽 양에 따라 시스템 기반의 서버 사양이나 네트워크 대역을 가늠하는 설계
- 재해에 대비하기 위해 해외에 백업을 구축하고자 하는 시스템
- 되도록 빨리 동작해야 하는 시스템

클라우드가 적합한 시스템으로는

먼저 트래픽의 변동이 많은 시스템입니다.
직원용 시스템처럼 이용자가 한정되어 있으면 트래픽을 예상하기 쉽기 때문에
미리 마련해둘 시스템 기반의 규모나 구성을 검토하기 쉽습니다.

하지만 고객용 시스템처럼 정확한 트래픽 예측이 어려운 시스템의 경우
트래픽 양에 따라 클라우드 오토스케일 기능을 이용해
시스템을 단기간에 쉽게 증설시킬 수 있는 클라우드 시스템으로 구성하는 것이 좋습니다.

이외에도 해외에 백업을 구축하고자 하는 시스템, 서비스를 빨리 제공하고 싶은 시스템일 때 적합하다고 합니다.

•트래픽의 변동이 많은 시스템
•클라우드 오토스케일 기능
동적으로 서버 사양이나 스토리지 용량을 쉽게 증설시킬 수 있음
•해외에 백업을 구축하고 싶은 시스템
•업무 연속성 계획
재해와 같은, 사업 중단을 야기하는 리스크를 최소한으로 하기위해 미리 준비해 두는 계획
•서비스를 빨리 제공하고 싶은 시스템

1.1.4 온프레미스가 적합한 케이스

- 높은 가용성이 요구되는 시스템
  - 보장하는 것 이상의 가용성이 필요한 미션 크리티컬 시스템에서는 클라우드를 실제 환경에 적용할 수 없음
- 기밀성이 높은 데이터를 다루는 시스템
- 특수한 요구사항이 있는 시스템

온프레미스 환경이 적합한 시스템으로는

첫째 높은 가용성이 요구되는 시스템입니다.
클라우드의 시스템 가용성은 클라우드 업체가 보장하는데요
예를 들어 네트워크가 잠시라도 끊어져서는 안되는 경우에는
클라우드 업체가 보장하는 것 이상의 가용성이 필요하기 때문에
가장 중요하고 필수적인 시스템에서는 클라우드를 실제 환경에
적용할 수 없습니다.
그렇기 때문에 온프레미스가 적합하다고 볼 수 있습니다.

둘째 기밀성이 높은 데이터를 다루는 시스템일 경우입니다.
물리적인 저장 장소를 명확히 할 필요가 있는 업무 데이터일수록
퍼블릭 클라우드에 저장할 수 없기 때문입니다.

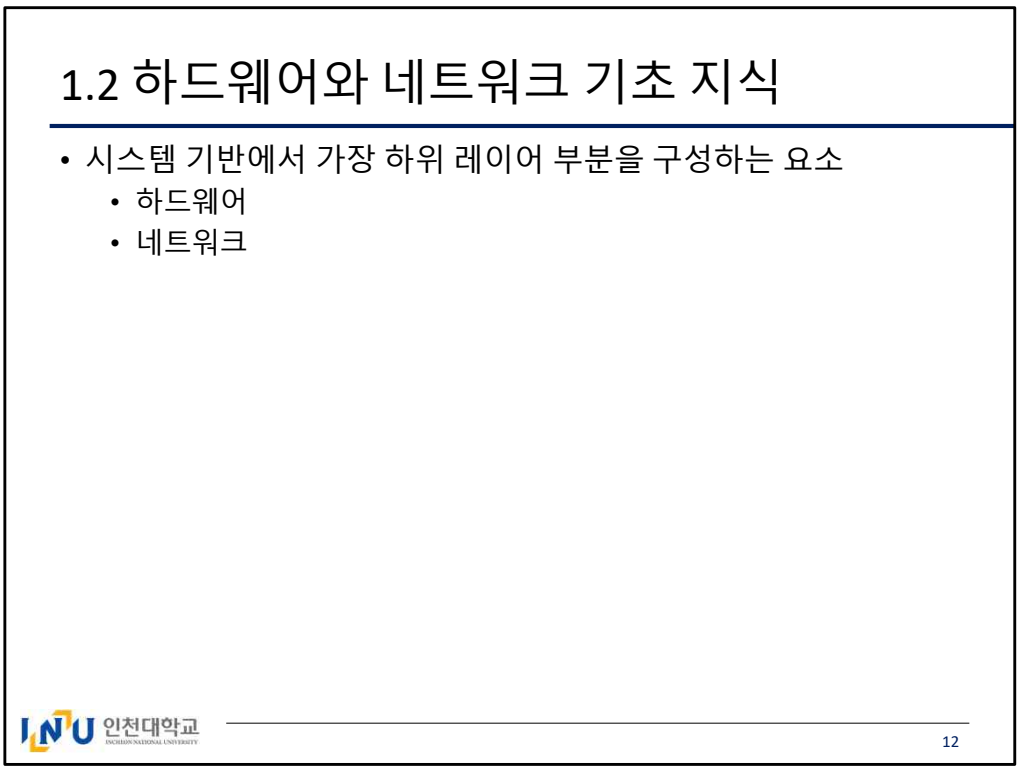
셋째 특수한 요구사항이 있는 시스템일 경웁니다.
특정 요구사항이 있거나 기준에 맞게 만들어야 할 경우에는
온프레미스 방식로 이용자에게 맞게 구축하면 됩니다.

1.1.5 시스템 기반의 구축/운용 흐름

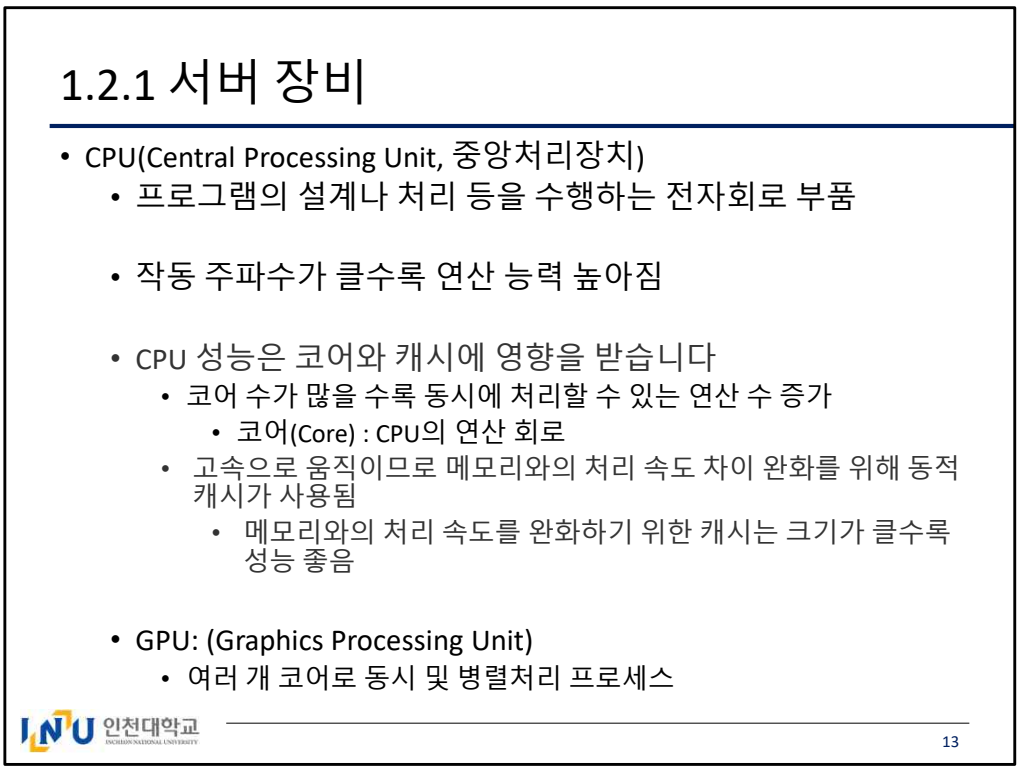
단계	설명
시스템 구축 계획 및 요구사항 정의 단계	구축 범위 선정, 요구사항 정의, 예산 책정, 프로젝트 체계화, 마이그레이션, 연계
인프라 설계 단계	아키텍쳐 설계, OS,미들웨어 선택, 시스템 운용 설계, 시스템 마이그레이션 셜게
인프라 구축 단계	네트워크 부설, 서버설치, 셋업, 테스트, 마이그레이션
운용 단계	서버프로세스, 네트워크, 리소스, 배치job모니터링, 데이터 백업, 정기유지보수, OS,미들웨어 업그레이드, 장애시 대응

다음으로 넘어가서 시스템 기반의 구축 및 운용 흐름을 간단히 살펴보겠습니다.
단계는 시스템 구축 계획 및 요구사항 정의 단계, 인프라 설계 단계, 구축 단계, 운용 단계로 이루어져 있습니다.

여기서 볼 점은 앞으로 사용할 도커라는 게
시스템 구축이나 시스템 운용에 있어서 지금까지 사람의 손으로 해왔던 많은 작업을 자동화하고,
테스트가 끝난 안전한 애플리케이션을 지속적으로 제공할 수 있는 플랫폼이라는 것입니다.



시스템 기반에서 가장 하위 레이어 부분을 구성하는 요소는 하드웨어와 네트워크입니다.



서버 장비부터 살펴보겠습니다.

CPU는 중앙처리장치로 프로그램의 설계나 처리 등을 수행하는 부품이고
성능은 코어와 캐시의 영향을 받습니다.

1.2.1 서버 장비

- 메모리(Memory, 주 기억장치)
  - CPU가 직접 액세스할 수 있는 기억장치 (기억 영역)
  - 데이터 용량이 크거나 전송 속도가 고속일수록 고성능
  - 서버용으로는 저전력에 에러 처리 기능을 가진 모델이 선택
  - 고성능 메모리일수록 비용 증가

메모리는 주 기억장치로

데이터 용량이 크거나 데이터의 전송 속도가 고속일 수록 고성능이지만,

서버용으로는 전력 소모가 적고 오류 처리가 탑재되어 있는 걸 주로 선정합니다.

그리고 CPU와 마찬가지로 온프레미스든 클라우드든 고성능 메모리일 수록 비쌉니다.

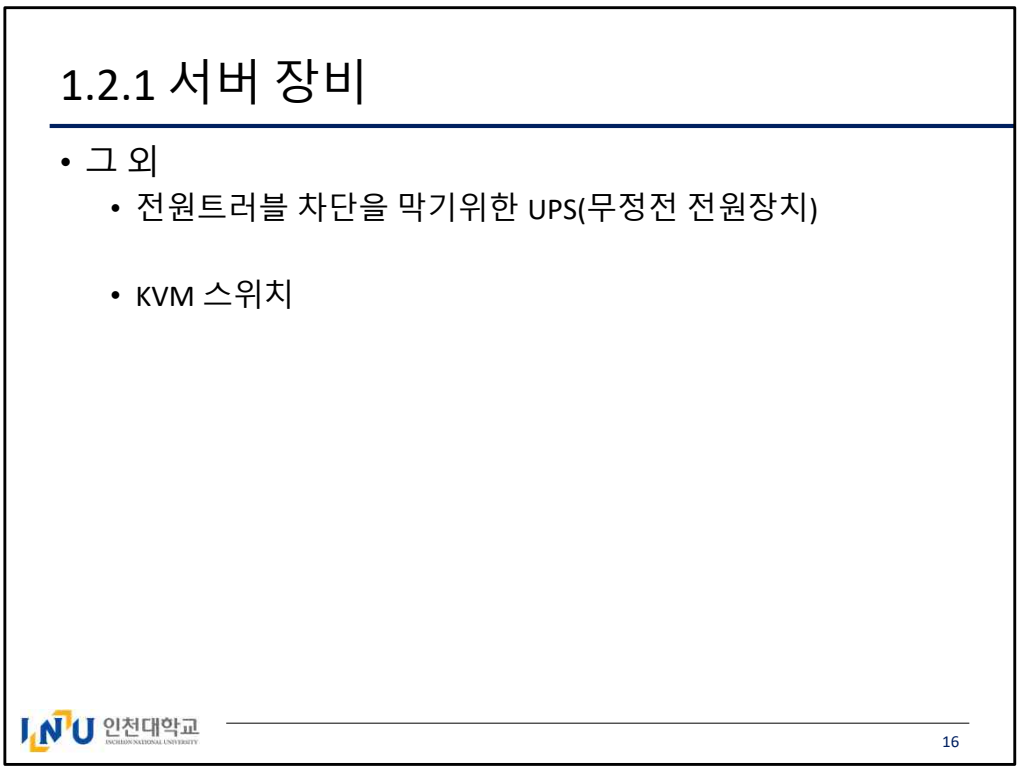
1.2.1 서버 장비

- 스토리지(Storage, 보조기억장치)
  - 영구적인 데이터 저장하는 디바이스
  - 하드디스크나 SSD등을 이용
    - 기업용도로 이용되는 Fiber Channer(FC) 하드디스크도 있음.
  - 대부분 다중화 구성을 취함

스토리지는 보조 기억장치로

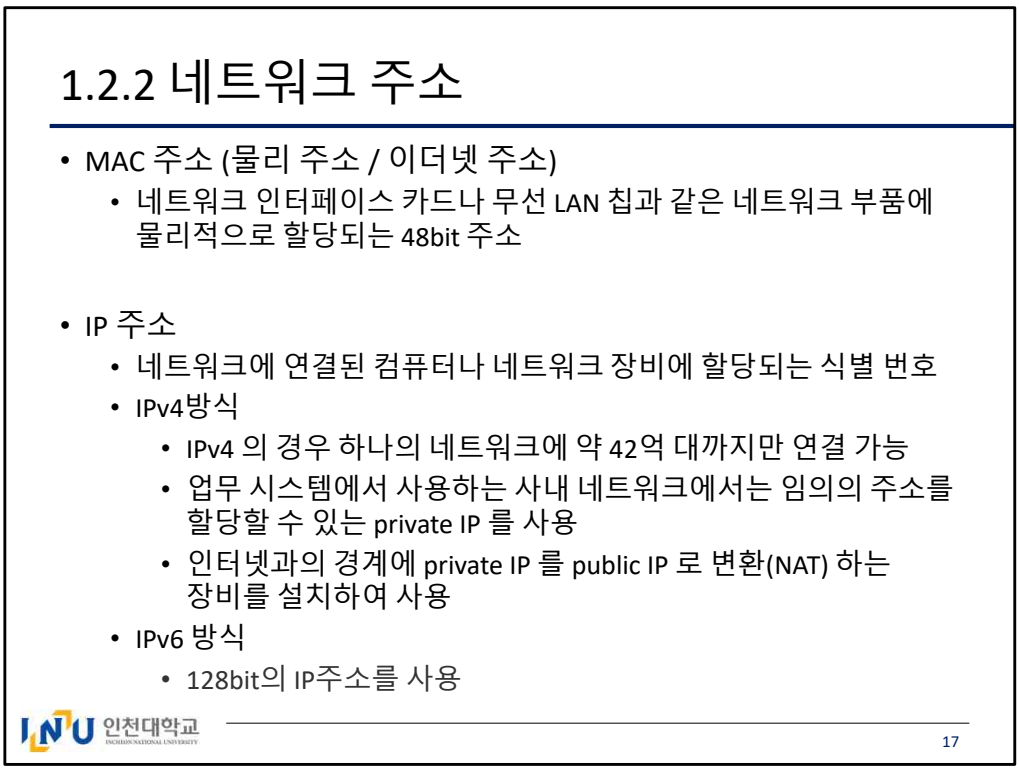
영구적인 데이터를 저장하는 디바이스 입니다.

스토리지의 용량이나 읽기 쓰기 속도가 시스템 전체의 응답 시간에 영향을 주는 경우가 많습니다.



그 외에는 이런 것들도 있습니다.

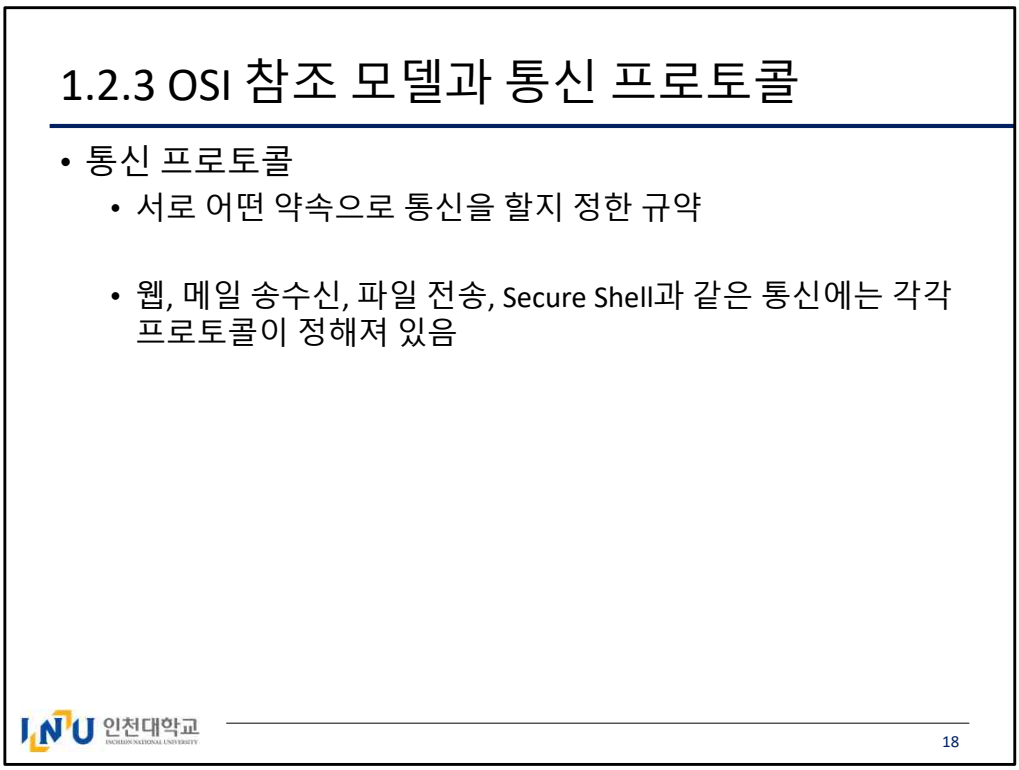
클라우드를 이용하는 경우 하드웨어 선정이 가상 머신의 사양 선정과 같기 때문에
효율적인 인프라 기반을 구축하려면 클라우드 서비스를 어떻게 조합하느냐도 중요하다고 생각 됩니다.



서버나 클라이언트, 네트워크 장비 등을 식별하기 위해 네트워크 주소를 사용합니다.

맥 주소는 네트워크 부품에 물리적으로 할당되는 48bit 주소입니다.
앞 24비트는 네트워크 부품의 제조업체를 식별하는 번호이고,
나머지 24비트는 제조업체가 중복하지 않도록 할당하고 있습니다.
OSI 참조 모델의 2계층인 데이터 링크 계층에서 사용합니다.

IP 주소는 네트워크에 연결된 컴퓨터나 네트워크 장비에 할당되는 식별 번호입니다.
IPv4, IPv6 방식 두 가지가 있고 일반적으로 IPv4를 사용합니다.



통신 프로토콜이란 '서로 어떻게 통신할 것인가'에 대해 정의한 규약을 말합니다.

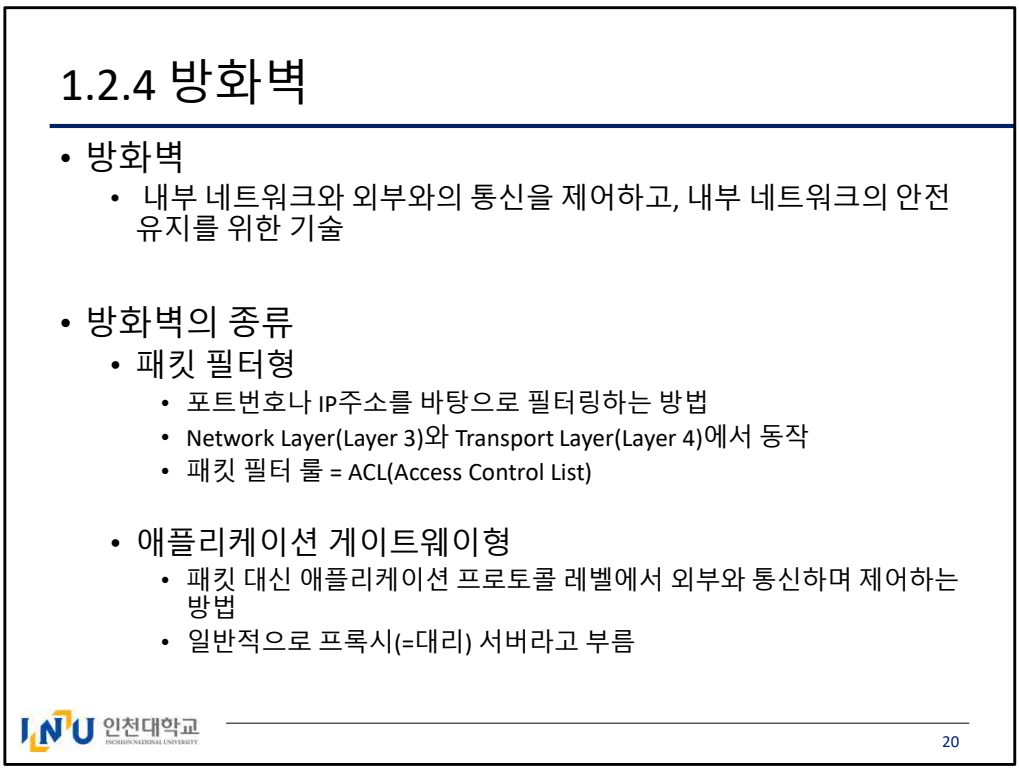
웹, 메일 송수신, 파일 전송, Secure Shell과 같은 통신에는 각각 프로토콜이 정해져 있습니다.

1.2.3 OSI 참조 모델과 통신 프로토콜

- OSI 참조 모델
  - ISO가 책정한, 컴퓨터의 통신 기능을 계층 구조로 나눈 개념 모델

계층	계층명	대표 프로토콜	대표 통신 기기
7계층 (L7)	응용 계층	HTTP, DNS, SMTP, SSH	방화벽, 로드밸런서
6계층 (L6)	표현 계층		
5계층 (L5)	세션 계층		
4계층 (L4)	전송 계층	TCP, UDP	
3계층 (L3)	네트워크 계층	IP, ICMP	라우터, L3 스위치
2계층 (L2)	데이터 링크 계층	Ethernet	L2 스위치, 브리지
1계층 (L1)	물리 계층		리피터 허브

OSI 참조 모델을 간단하게 살펴보면
OSI 7레이어는 컴퓨터 통신을 계층 구조로 나눠 놓은 것인데요,
7계층은 앱에 특화된 프로토콜을 규정하고
6계층은 데이터의 표현 형식을 규정하고
5계층은 커넥션 확립 타이밍이나 데이터 전송 타이밍을 규정합니다.
4계층은 데이터 전송을 제어하는 계층이고,
3계층은 서로 다른 네트워크 간에 통신을 하기 위한 규정이고,
2계층은 동일한 네트워크 안에 있는 노드 간의 통신을 규정합니다.
1계층은 통신 장비의 물리적 및 전기적 특성을 규정합니다.



시스템을 가동시킬 때 가장 주의해야 할 것이 바로 보안입니다.
보안을 확보하기 위해 가장 효과적인 방법 중 하나는 불필요한 통신을 차단하는 것입니다.
방화벽은 내부 네트워크와 외부와의 통신을 제어하고, 내부 네트워크의 안전을 유지를 위한 기술입니다.

방화벽에는 몇가지 종류가 있는데
패킷 필터링 방식은 통신하는 패킷을 포트 번호와 IP 주소를 기반으로 필터링 하는 방법입니다.
네트워크 계층과 전송 계층에서 동작하고 정한 룰에 기초해 패킷을 필터링합니다.

애플리케이션 게이트웨이형은 프록시 서버라고도 부르는데 패킷 대신
애플리케이션 프로토콜 레벨에서 외부와의 통신을 대체 제어하는 방법이라고 합니다.

1.2.5 라우터

- 라우터는 2개 이상의 서로 다른 네트워크 사이를 중계하기 위한 통신 장비
- OSI 7 Layer의 Layer 3인 Network Layer에서 동작
- 경로 선택 기능 보유
- 동적 경로(Dynamic Route) 방식
  - 라우팅 프로토콜에서 설정된 경로
- 정적 경로(Static Route) 방식
  - 라우터에 설정된 테이블을 바탕으로 정해진 경로

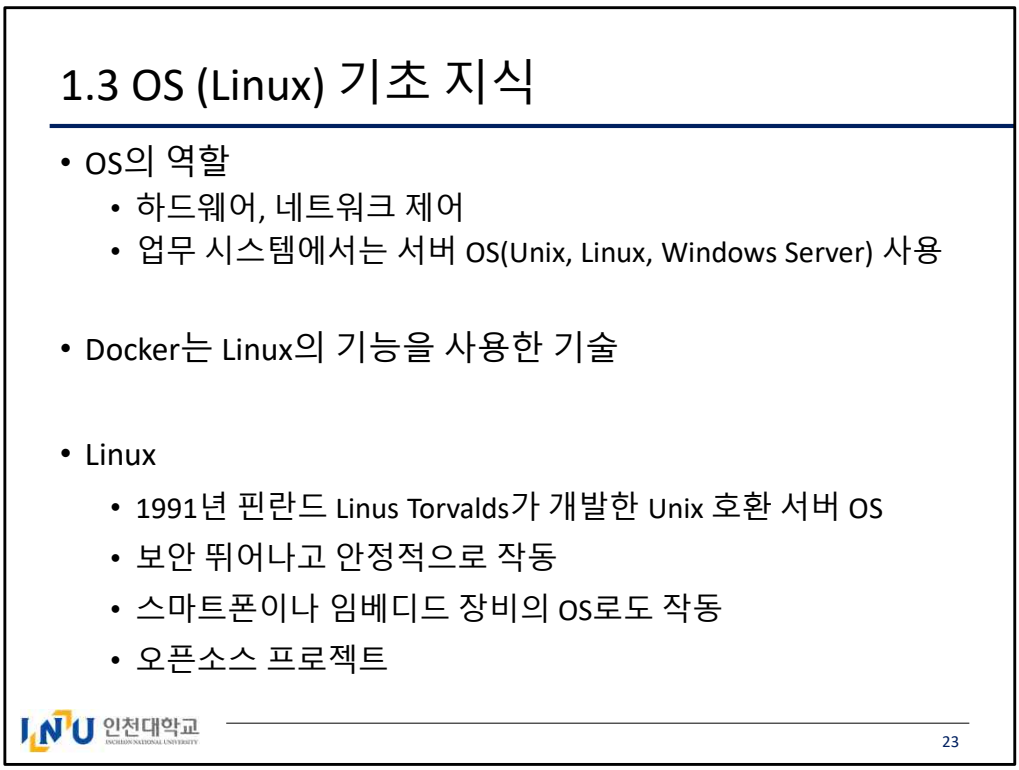
라우터는 서로 다른 네트워크를 연결하기 위한 통신 장비로 네트워크 계층에서 작동합니다.

어떤 루트를 통해 데이터를 전송할지 판단하기 위해 경로 선택 기능을 가지고 있고
라우팅 프로토콜에서 설정한 동적 경로 방식과
라우터에 설정된 라우팅 테이블 기반으로 결정하는 정적 경로 방식이 있습니다.

1.2.6 레어터 3 스위치

- 라우터와 거의 같은 기능
- 라우팅을 하드웨어에서 처리
- 빠르게 동작
- 접속할 수 있는 이더넷 포트 수가 많아 업무 시스템 등에 넓게 사용

라우터와 거의 똑같은 기능을 가진 레이어 3 스위치는
라우팅을 하드웨어로 처리하고 있으므로 빠르게 작동한다는 것이 특징이고,
연결할 수 있는 이더넷 포트수가 많아 널리 이용되고 있습니다.



OS의 역할은 하드웨어나 네트워크를 제어하는 것이라고 할 수 있고 도커도 리눅스의 기능을 사용한 기술입니다.

리눅스는 리누스 토르발스가 개발한 유닉스 호환 서버 OS입니다. 오픈 소스로 기업, 개인의 참여로 만들어지고 있고 보안성도 좋고 안정적으로 작동한다는 장점이 있습니다.

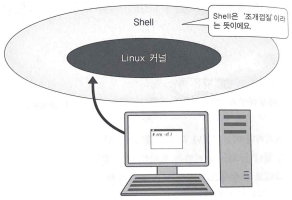
1.3 OS (Linux) 기초 지식

- Linux 커널
  - OS의 코어
  - 하드웨어나 애플리케이션 소프트웨어를 제어하기 위한 기본적인 기능을 갖고 있는 소프트웨어
  - Android도 리눅스 커널 상에 구축
- Linux 배포판
  - 패키지화 되어 배포되는 형태
  - 사용자 공간(userland) : 리눅스 커널 이외의 부분
  - 사용자 공간에서 디바이스에 직접 액세스할 수 없음 (커널을 통해 처리)

리눅스 커널은 OS의 코어가 되는 부분이고
리눅스는 보통 리눅스 배포판이라는 형태로 패키지화되어 배포됩니다.
이 배포판에는 리눅스 커널과 함께 각종 커맨드, 라이브러리, 애플리케이션이 포함되어 있습니다

1.3.1 Linux 커널

- Linux 커널을 조작하기 위해 쉘(Shell)을 사용
  - 쉘은 사용자 명령을 커맨드에 입력하고 이를 Linux 커널에 전달.
  - 쉘 스크립트 : 쉘에서 실행하고자 하는 명령을 텍스트 파일로 저장한 것



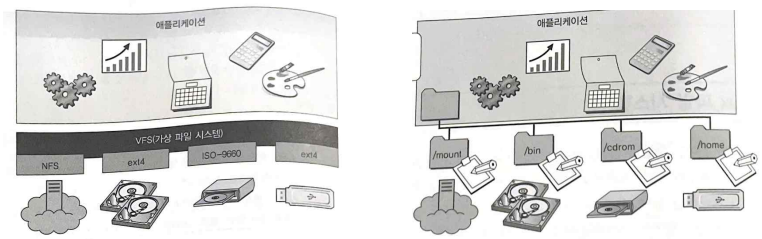
- 디바이스 관리
  - 하드웨어(CPU/메모리/디스크/입출력 장치 등)를 디바이스 드라이버(소프트웨어)로 제어
- 프로세스 관리
  - 프로세스에 PID(프로세스ID)라는 식별자를 붙여 관리하고 프로세스 실행에 필요한 CPU를 적절하게 할당
- 메모리 관리
  - 메모리 할당/해제
  - 메모리의 물리적인 용량을 넘어선 프로그램이나 데이터가 진행되면 하드디스크와 같은 보조기억장치의 가상 메모리 영역이 사용됨.

리눅스 커널은 하드웨어 제어에 관한 os의 핵심이 되는 기능을 의미하고 조작하기 위해서는 쉘을 사용해야 합니다.
쉘을 통해 커널에 명령을 전달하는 거져

리눅스 커널은 메모리 관리, 파일 시스템, 프로세스 관리, 디바이스 제어의 역할을 합니다.

1.3.2 Linux 파일 시스템

- 파일 시스템
  - Linux에서 하드디스크나 USB 메모리, CD, DVD 등과 같은 데이터에 액세스하기 위한 장치
  - Linux 커널은 VFS(Virtual File System)시스템을 사용하여 데이터에 편리하게 액세스 가능
    - 각 디바이스를 '파일'로 취급하는 것이 특징.



인천대학교 INCHEON NATIONAL UNIVERSITY

리눅스 커널이 갖고 있는 중요한 기능으로는 파일 시스템이 있습니다.

리눅스 커널은 가상 파일 시스템을 사용하여 데이터에 편리하게 액세스가 가능한데여,
애플리케이션은 디바이스의 차이를 의식하지 않아도 되는 겁니다

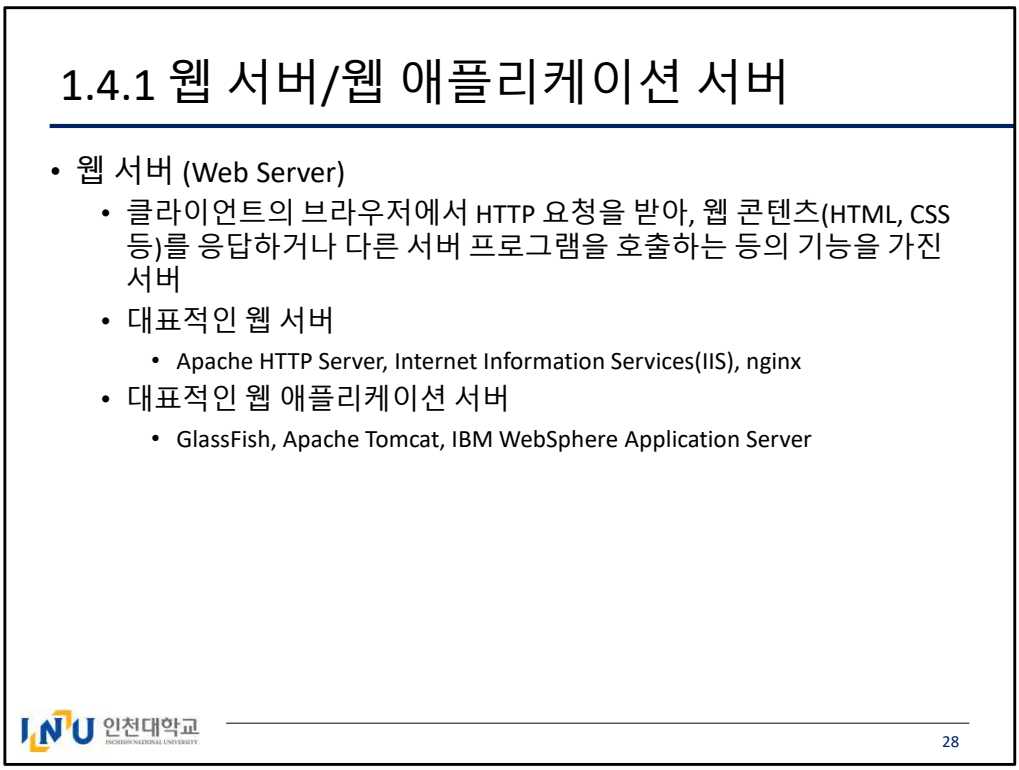
VFS에서는 각 디바이스를 ' 파일 ' 로 취급하기 때문에
애플리케이션에서 보면
어디에 데이터를 저장하든 모두 파일로 보이도록 해주는 것이 특징입니다.

1.4 미들웨어 기초 지식

- 미들웨어
  - OS와 업무 처리를 수행하는 애플리케이션 사이에 들어가는 소프트웨어
  - 웹서버, DBMS, 시스템 모니터링 툴 , …
- 목적과 용도에 따라 폭넓은 종류 이용
  - OS가 갖고 있는 기능을 확장한 것
  - 애플리케이션에서 사용하는 공통 기능을 제공하는 것
  - 각종 서버 기능을 제공하는 것 … 등

미들웨어는 OS와 업무 처리를 수행하는 애플리케이션 사이에 들어가는 소프트웨어를 의미합니다.

목적과 용도에 따라 다양한 종류가 이용되어 있고
OSS로 개발되어 있는 것과 사용으로 제공되는 것 등 지원하는 제품도 다양합니다.



웹서버란 클라이언트의 브라우저가 보내온 HTTP 요청을 받아, 정적인 컨텐츠를 응답으로 반환하는 서버를 말합니다.

아파치 서버는 폭 넓게 사용되는 오픈소스 웹 서버이고 nginX는 소비 메모리가 적고 리버스 프록스와 로드 밸런서 기능을 갖추고 있는 오픈소스 웹서버입니다.

1.4.1 웹 서버/웹 애플리케이션 서버

- 데이터베이스 서버(데이터베이스 관리 시스템, DBMS)
  - 시스템이 생성하는 다양한 데이터를 관리하기 위한 미들웨어
  - RDBMS(Relational Database Management System)
    - 관계형 데이터베이스 관리 시스템
    - 관계형 데이터베이스
      - 데이터를 2차원 표 형식으로 관리하는 데이터베이스
    - MySQL, PostgreSQL, OracleDatabase, …
  - NoSQL
    - RDBMS와는 다른 새로운 방식을 통틀어 일컫는 말
    - 병렬 분산처리, 유연한 스키마 설정, …
    - KVS(Key-Value 스토어), 도큐먼트 지향 데이터베이스, XML 데이터베이스, …
    - Redis, MongoDB, Apache Cassandra

데이터베이스 서버는 시스템이 생성하는 다양한 데이터를
관리하기 위한 미들웨어 입니다.

데이터의 검색, 등록, 변경 등과 같은 기본 기능 외에 트랜잭션 처리 등도 포함한다는 점에서
데이터베이스 관리 시스템이라고 부르는 경우도 있습니다.

관계형 데이터베이스 관리 시스템인 RDBMS와 병렬분산처리나
유연한 스키마 설정 등이 특징인 NoSQL이 있습니다.

1.4.1 웹 서버/웹 애플리케이션 서버

- 시스템 감시 툴
  - 시스템의 감시 대상인 서버나 장비의 상태를 감시하여 미리 설정한 경계값을 초과한 경우 정해진 액션을 실행
  - zabbix, Datadog, Mackerel

시스템을 안정적으로 가동시키기 위해서 시스템 관리자는
시스템이 어떤 상태로 가동되고 있는지 감시할 필요가 있는데 그 때 감시를 이 시스템 감시 툴로 수행합니다.

시스템 감시 툴이란 시스템의 감시 대상인 서버나 장비의 상태를 감시하여
미리 설정한 경계값을 초과한 경우 정해진 액션을 실행하는 툴을 말합니다.

1.5.1 인프라 구성 관리

- 인프라 구성 관리
  - 인프라를 구성하는 하드웨어, 네트워크, OS, 미들웨어, 애플리케이션 구성 정보를 관리하고 최적의 상태를 유지하는 것
- 변경 가능한 인프라 관리의 문제점
  - 인프라 구축 규모가 크면 클수록 인프라 구성 관리에 대한 부담 증가
- Immutable Infrastructure(불변 인프라)
  - 운영 환경에 한번 설치된 서버를 변경 불가능한 것이고 일회용으로 취급하는 개념
  - 변경이 필요한 경우 아예 새로운 서버를 구축하는 방법
  - 서버를 항상 깨끗한 상태로 유지 가능
  - 인프라 환경의 변경이 쉬움
  - 서버를 쉽게 삭제하고 늘릴 수 있음

인천대학교 INCHEON NATIONAL UNIVERSITY

31

인프라 구성 관리는 인프라를 구성하는 하드웨어, 네트워크, OS, 미들웨어, 애플리케이션의 구성 정보를 관리하고 적절한 상태로 유지하는 작업을 말합니다.

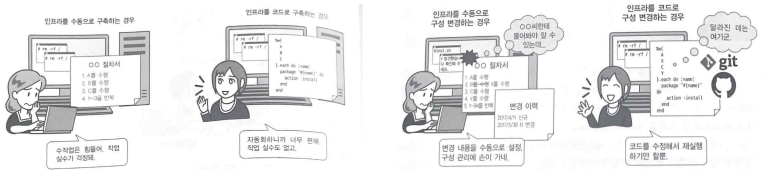
인프라 구축 규모가 크면 클수록 인프라 구성관리에 대한 부담도 늘어나게 되는데
클라우드 시스템과 여러 가상화 기술의 등장으로 인프라 구축 방법에 많은 변화가 왔습니다.

한번 설치된 서버는 변경 불가능한 것이고 일회용이라고 취급해버리는 개념인
Immutable Infrastructure이란 환경으로 변하고 있는 건데요,
서버를 변경하지 못하도록 했을 때 얻을 수 있는 가장 큰 장점은 바로 "서버를 항상 깨끗한 상태로 동일하게 유지" 하는 것이고

인프라의 변경 이력을 관리하는 게 아니라 현재 동작하고 있는 인프라의 상태만 관리하면 되도록 바뀌었습니다.

1.5.2 코드를 사용한 구성 관리

- Infrastructure as Code ：인프라의 구성을 코드로 관리하는 것
- 구성 관리 시 여러 대의 서버를 모두 수작업으로 설정하여 관리하는 것은 비효율적이므로 코드를 사용하여 구성 관리
- 인프라 구성 관리를 코드로 하고 형상관리로 이력 관리
  - 애플케이션 개발에서의 소스코드 관리와 같이 버전 관리툴을 사용하여 변경 이력을 일원화하여 관리 가능.



- Docker는 Dockerfile에 인프라의 구성 정보를 기술 가능

구성 관리 시 여러 대의 서버를 모두 수작업으로 설정해 관리하는 것은 비효율적이므로
코드를 사용해서 구성 관리하게 됩니다.

셋업 작업의 대부분은 시간이 걸리는 단순 작업인데 이걸 수작업으로 하게 되면
아무리 작업 체크나 테스트를 신경 써서 하더라도 환경 설정의 작업 실수를 완전히 막기는 어렵습니다.

그렇기 때문에 코드에 쓰인 대로 자동으로 설정해주도록 해서 누가 실행해도 똑같은 상태의 인프라가 환경을 구축할 수 있도록 하면 보다 효율적으로 할 수 있는 건데요,

도커에서 도커파일에 인프라의 구성 정보를 쓸 수 있는 거와 같다고 보시면 됩니다.

1.5.3 대표적인 인프라 구성 관리 툴

- OS의 시작을 자동화하는 툴 (Bootstrapping)
  - OS 설치
  - 가상 환경 설정
  - 네트워크 구성 설정
  - KickStart, Vagrant, …
- OS나 미들웨어의 설정을 자동화하는 툴 (Configuration)
  - OS 설정(보안/서비스 시작, …)
  - 미들웨어(각종 서버)의 설치 및 설정
  - Chef, Ansible, Puppet, Itemae, …
- 여러 서버의 관리를 자동화하는 툴 (Orchestration)
  - 애플리케이션 배포
  - 서버군의 오케스트레이션
  - 분산 환경의 서버 관리 툴 : Kubernetes

인프라 구성 관리를 자동으로 관리하는 툴 중
툴이 갖고 있는 기능을 역할에 따라 크게 나누면 3가지로 나눌 수 있습니다.

OS의 시작을 자동화하는 툴은
서버 OS를 설치하거나 가상화 툴을 설치 및 설정하는 작업을 자동화하기
위한 툴로 KickStart, Vagrant 등이 있습니다.

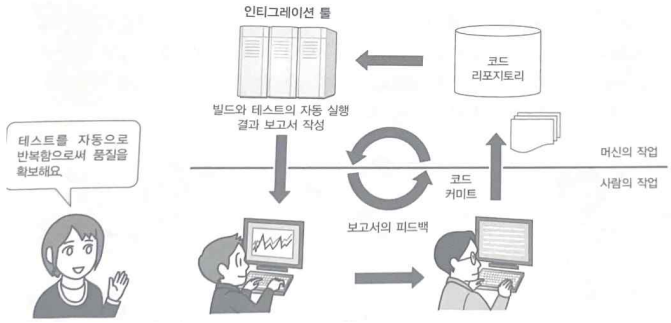
OS나 미들웨어의 설정을 자동화하는 툴은 보안과 관련된 설정을 자동화 하기
위한 툴입니다.

여러 서버의 관리를 자동화 하는 툴은 여러 대의 서버로 구축된 분산 환경의
서버들을 관리하기 위한 툴입니다.

여기엔 컨테이너 오케스트레이션의 사실상 표준이라고 할 수 있는,
컨테이너 가상 환경에 있어서 여러 컨테이너를 통합 관리하는 툴인
쿠버네티스가 있습니다.
쿠버네티스는 도커의 컨테이너를 효율적으로 관리해줍니다.

1.5.4 지속적 인티그레이션/지속적 딜리버리

- 지속적 인티그레이션 (Continuous Integrantion)
  - 애플리케이션의 코드를 추가/수정할 때 마다 테스트를 실행하고 확실하게 작동하는 코드를 유지하는 방법
  - 소프트웨어의 품질 향상을 목적으로 고안된 개발 프로세서
  - 단위 테스트 : 소프트웨어의 특정 부품에 대해 사양서에 정해진 대로 작동하는 지 확인하는 테스트



지속적인 인티그레이션이란
애플리케이션의 코드를 추가하거나 수정할 때마다 테스트를 실행하고
확실하게 작동하는 코드를 유지하는 방법을 말합니다.
소프트 웨어를 품질 향상을 목적으로 고안된 프로세서죠

소프트웨어의 개발이 진행됨에 따라서
단위 테스트의 수가 늘어나게 되므로 이런 테스트를 자동화하기 위해
인티그레이션 툴을 사용 합니다.

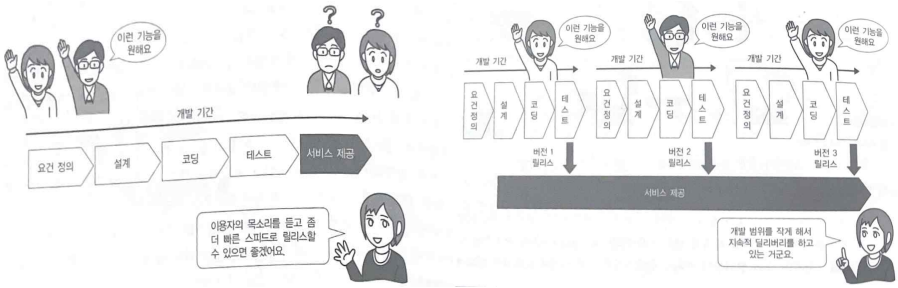
하지만 단위 레벨에서 테스트가 끝난 모듈이 다른 환경에서도 똑같이
작동한다는 보장은 없습니다.

애플리케이션을 실행하려면 OS의 설정이나 네트워크 주소, 영구 데이터의
저장 장소, 애플리케이션을 실행할 계정의 권한 등 인프라 환경에
의존하는 부분이 많기 때문인데요

이런 인프라 구성에 관한 부분을 코드로 관리할 수 있다면 개발 멤버가
항상 동일한 환경에서 개발할 수 있으므로 지속적인 인티그레이션에서
필요하는 환경의 구성 관리가 더욱 쉬워질 겁니다.

1.5.4 지속적 인티그레이션/지속적 딜리버리

- 지속적 딜리버리 (Continuous Delivery)
  - 모든 기능을 한 번에 다 만드는 것이 아닌 기능을 추가할 때마다 애플리케이션 제품 환경에 배포하고, 시스템 이용자의 피드백에 기초해 다음 개발 기능을 결정.
  - 짧은 사이클의 개발과 릴리스를 반복함으로 이용자가 원하는 애플리케이션 적시에 제공 가능



폭포형 애플리케이션 개발은 요건 정의, 설계 코딩, 테스트라는 프로세스를 거쳐
애플리케이션의 개발을 끝낸 후에 제품 환경에 애플리케이션을 배포하여 서비스를 릴리스 합니다.

하지만 요건 정의 부터 서비스 릴리스까지 오랜 시간이 소요되기 때문에 막상 릴리스한 시점에서
애플리케이션이 이용자가 원하는 니즈(요구)를 만족시키지 못하는 경우도 있습니다.

그래서 모든 기능을 한 번에 다 만드는 것이 아니라 기능을 추가할 때마다
애플리케이션을 제품 환경에 배포하고 이용자의 피드백을 베이스로
지속적인 딜리버리를 받으면서 다음 개발할 기능을 결정한다는 스타일이 생겨나게 된 겁니다.

제 발표는 여기서 마무리하겠습니다.