1️⃣

01. 시스템 디자인 & 확장

: How CC(Cloud Computing) makes IT bussiness better?

시스템 디자인 : Simple SNS

- **기존 방식의 시스템 디자인 및 확장 과정**을 공부 → **클라우드 컴퓨팅의 장점**을 학습

목표? : 간단한 SNS 시스템을 디자인

- *참고 : "**시스템 디자인**"은 주어진 목적을 달성하기 위한 **시스템 아키텍쳐**, **시스템을 구성하는 모듈**, **인터페이스**, **데이터** 등을 정의하는 과정을 의미.*
- **디자인 과정**
  1. 사용자에게 SNS 서비스를 제공하기 위해 **필요한 기능을 정의**
  2. SNS 서비스 구현을 위해 **필요한 모듈**을 정의
  3. **전체 시스템** 디자인
  4. 필요시, 시스템을 **재구성**

1) Defining Functions (기능 정의)

- **목표** : 소규모 사업장을 위한 간단한 SNS 구현 (직원 수 : 100명 가량)
  - **서비스 품질(QoS)**이 우수해야 함
  - "우수한 품질"은 추상적 개념이며, **낮은 응답시간, 높은 처리용량/속도, 낮은 오류율 등 측정 가능한 기준으로 정의**해야 함
- SNS 서비스를 위해 필요한 기능?
  - 가입/등록

        - 신규 사용자 가입 시 ID/PW 생성
    - 친구 요청/수락
        - 친구 추가 요청 보내기
        - 친구 추가 요청을 승낙/거절
    - 글쓰기
        - 글쓰기
        - 내 친구로 등록된 사용자가 게시한 글 읽기

2) Identifying the modules we need (모듈)

- SNS 서비스를 제공하기 위해 필요한 **서버측 모듈/컴포넌트**는?

    - **웹 서버 (Web Server) 또는 WWW + AWS**
        - 서비스 접속 위한 **웹 페이지** 제공
        - 사용자 **생성/관리**
        - 친구관계 **관리** (요청, 수락, 거절 등)
        - **글** 쓰기/읽기
    - **데이터베이스 (DB)**
        - 사용자 **계정 정보** 저장
        - **친구관계** 저장
        - 사용자가 작성한 **글 저장**

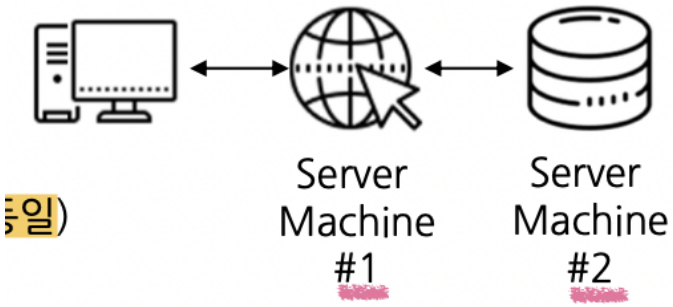
3) Designing the system (시스템 디자인)

디자인 #1

- **웹 서버(WWW), 데이터베이스(DB)**를 **한 대의 서버 컴퓨터에** 설치
- **서버 사양**
  - Single core, 2.0Ghz processor (CPU)
  - 1G RAM (memory)
  - 256G SATA (HDD)
- *If, 직원 수가 1000명으로 늘어나 응답시간이 증가하고 오류가 발생? → 시스템 분할!*
  - *가입자 수보다 중요한 것 : 동시접속사 수 고료 (실시간 리소스 관리 - Queue로 모델링, 동잡자 수 많아지면 대기 시간이 중요함)*

디자인 #2 : 시스템 분할 (partitioning the system)

- 한 대의 서버를 두 대로 분리 (기능적 분할)



  - 서버#1에는 WWW(웹서버)
  - 서버#2에는 DB

  ⇒ 기능/역할에 따른 분할 (각각의 서버 사양은 이전과 동일)
- 디자인 #1에 비해 서비스 품질이 개선된 이유? (응답시간 감소, 오류 감소)
  - 시스템 부하 감소
  - 기존에는 한 대의 컴퓨터가 WWW와 DB 없무 처리를 모두 담당했으나, 지금은 각각의 서비스가 담당하는 일이 줄어듦

- *If, 사용자 수가 더 증가하여 다시 서비스 품질이 떨어지면?*
    - *Scale Up 방식으로 시스템을 개선*

4) Revising the system (시스템 개선)

Scaling 방법

1. **Vertical Scaling : 수직적 확장 (scale-up/down)**
    - 기존의 시스템 장비/부품을 고사양으로 대체
2. **Horizontal Scaling : 수평적 확장 (scale-out/in)**
    - 더 많은 장비를 추가하여 성능 확장 (이론적 Good)

VS (수직적 확장)	HS (수평적 확장)
- VS 수행 중 서비스 중단 가능 - VS는 무제한 수행 불가 (장비, 기술적 한계로 제한됨) - 부품 교체 시, 기존 부품 비용 낭비를 초래	- 효율적으로 시스템 성능 확장 가능 - 병렬 처리 가능 - 부하 분산 처리 가능 - 시스템 확장 제한 X - 성능 확장/감소 작업 쉬움 - 범용적 장비를 통해 성능 확장이 가능, 감소시에는 회수한 장비 다른 목적으로 사용 가능 → 비용 낭비 X

디자인 #3 : Vertical Scaling

- 각 부품 성능 개선
    - Single core, 2.0Ghz CPU **⇒ Dual core, 3Ghz**
    - 1G RAM (memory) **⇒ 4G RAM**
    - 256G SATA (HDD) **⇒ 512 M.2 SSD**
- 디자인 #2에 비해 서비스 품질 개선 (응답시간 감소, 오류 감소
    - **서버 부품이 고성능으로 교체, 시간당 연산량/처리량 증가**
- *VS에는 한계가 있음 ⇒ Scale Out 방식 사용!*

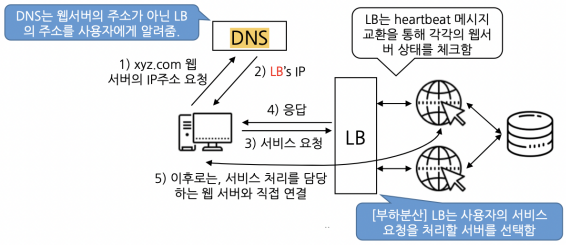
디자인 #4 : 웹서버 늘리기 (Doubling WWW)

- 웹 서비스를 위한 서버 1대 더 추가
  - 각 웹 서버가 처리하는 부하 반으로 줄이기 (서버 부하 분산)

⇒

- *부하 분산 방법?*
- *누가 부하 분산?*
- *서버 2대면, 사용자는 어느 웹 서버 접속?*

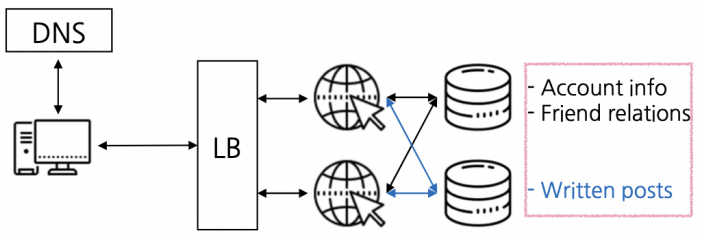
- **LB (Load balancer, 부하 분산자)**를 도입
- LB는 정해진 정책에 따라 부하를 분산시킴
  - Ex) RR (round-robin) 기법
    - 사용자의 요청에 따라 순차적으로 서버에 요청 전달



  - 사용자가 지속적으로 증가 ⇒ 서비스 성능 다시 저하
  - *성능 분석 결과, DB에서 병목 현상이 발생*

디자인 #5 : DB 늘리기

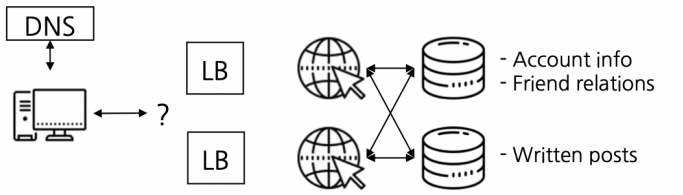
- DB를 1대 더 추가하고, 데이터 타입에 따라 역할 분리
  - DB#1 : 사용자 계정, 친구관계 정보 저장/관리
  - DB#2 : 사용자 게시 글 관리



- *If, LB 다운/비정상 종료*
  - *HA 수행! (high availability)*

디자인 #5 : LB 늘리기

- LB 1대 더 추가하고, 하나가 고장나면 다른 하나 사용하는 방식으로 시스템 안정성 높이기

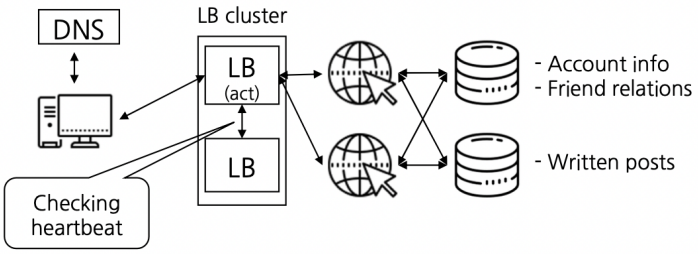


- *사용자는 어느 LB에 접근? (이중화)*
  - ***HA (High Availability)***

- **HA (High Availability, 고가용성)**
  - **서버, 네트워크, 프로그램 등 정보 시스템이 상당히 오랜 기간 동안 지속적으로 정상 운영이 가능한 성질**
  - 고가용성 = 가용성 높음 = 시스템이 고장나지 않음 = **서비스가 중단되지 않음**
  - 여분의(redundant) 장비를 추가로 사용해 HA 달성 가능
    - *"대기" 하는 장비를 두고, 장비가 고장나면 즉시대체*
- **HA 클러스터**
  - **서비스 제공하는 다수의 장비로 구성된 그룹, 시스템의 안정성을 높이고 downtime(중단시간)을 최소화하고 uptime(가동시간)을 최대화 함**

- **장애 극복(failover) : 하나의 장비가 고장나면, 클러스터 내 다른 장비가 서비스 처리**
- 모드
  - **active-active** : 두 대의 장비가 **모두 활성화되어 서비스를 제공**, 한 대의 장비에 문제가 생기면 **해당 장비는 사용 x**
  - **active-standby** : 한 대는 active, 다른 한 대는 standby 모드로 설정하고 active 장비만 서비스를 제공함. **active가 고장나면 즉시 standby가 active로 전환**해 서비스 제공
- *참고 : 클러스터는 고정된 가상 IP (virtual IP)를 대표 IP로 사용, 대표 IP는 active 장비와 논리적으로 연결*

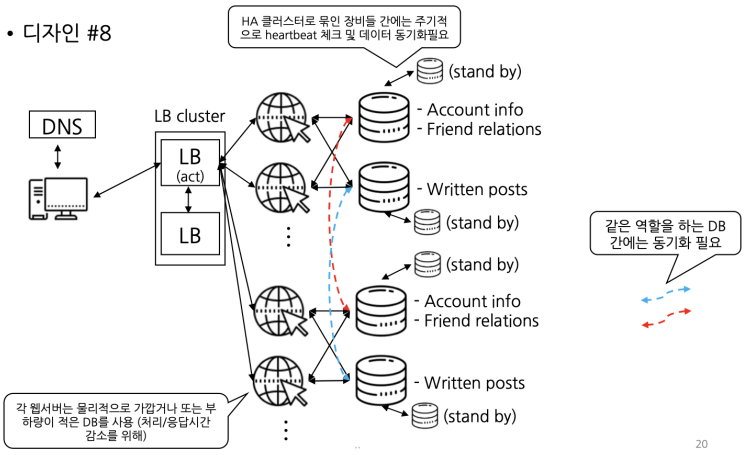
- **+ HA 예시**



  - 2대의 LB로 클러스터를 구성, active-standby 모드로 설정
  - Avtive 상태의 LB에 장애 발생시, standby 상태의 LB가 avtive로 전환

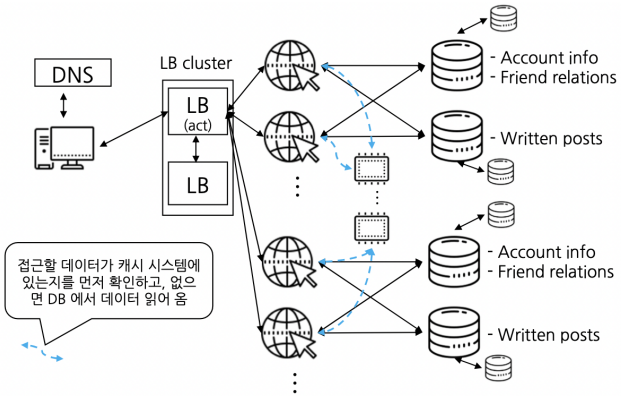
- **웹서버**에 접근하는 **사용자 증가로 인해 서비스 품질**이 떨어지거나, **웹서버 두 대가 동시에 고장**나면?
  - **디자인 #6 : 웹서버 확장** (Horizontal Scaling + HA clustering)
- **DB**로 인한 **성능 병목 현상이 발생 or DB 장애 발생**?
  - **디자인 #7 : DB 서버 확장** (Horizontal scaling + HA clustering)
- 같은 역할 DB가 다수 → 동기화 필요
  - **디자인 #8 : DB 동기화**

디자인 #8 : DB 동기화



디자인 #9 : 캐시 시스템 도입

- 응답 시간을 줄여 서비스 품질을 개선하기 위해
  - 빈번히 access되는 데이터를 캐시에 저장
- **Cache**



  - 접근 시간을 줄이기 위해 캐시 시스템 사용
  - **빈번히 access 되는 데이터를 사전에 캐시 시스템에 저장**
  - 일반적으로 DB에서 읽어오는 것보다 응답시간 빠름

그 외

- **갑자기 90% 사용자 탈퇴 시?**
  - over-provisioning (과한 구춪ㄱ)
  - 대부분 시스템 자원 사용 X → 비용 낭비 (서버 구입비, 서버 운영비 낭비)
  - **HS (horizontal scaling) 방식으로 성능 확장한 경우, 일부 장비를 제거해 다른 시스템 성능 확장에 사용 가능**

- **갑자기 사용자 100배 폭증?**
  - 서버 성능을 줄여놓은 상태이므로 **서비스 품질 크게 저하됨**
  - HS 방식으로 시스템 성능 다시 확장

- **HS 방식**을 사용하는 경우
  - 서비스에 장애 없이 성능을 높이거나 줄일 수 있지만, 이 과정에서 **서버 재배치, OS/SW 설치 등에 시간+노력이 필요**함.

- VS/HS 방식으로 성능 확장할 경우, **새로운 부품/장비를 구매 ⇒ 많은 비용 + 시간 + 노력이 필요**
  - 부품 구매 ⇒ 초기 비용이 큼
  - 특수 목적으로 제작된 서버 구매 시 오랜 시간을 기다려야 함
  - 기다리는 동안 서비스 품질 저하 막을 수 없음

- VS/HS 방식으로 시스템 확장되는 경우, **운영비 증가**
  - 서버실 운영비 (전기, 냉방), 서버실 관리자 인건비 등

- 기존의 성능 확장 방식 ⇒ 비용 + 시간 필요
- 사용자 트래픽이 큰 폭으로 변동할 경우, 신속한 대응 불가
  - 대부분의 시스템 자우너이 낭비되거나 (사용자 급격히 감소)

- 서비스 품질이 저하됨 (사용자 급격히 증가)

결론 : 클라우드 컴퓨팅 기술을 이용하면 적은 비용으로 동적으로 신속하게 시스템 성능 조절 가능