가상 면접 사례로 배우는

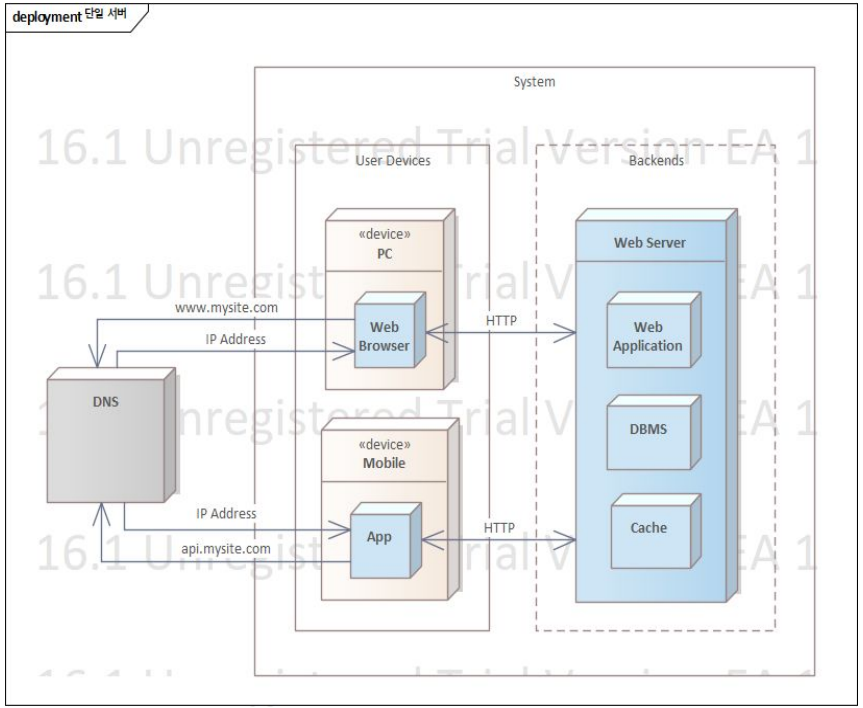
대규모 시스템 설계 기초

이정민

1. 사용자 수에 따른 규모 확장성

단일 시스템에서 대규모 시스템으로 규모를 확장하여 설계하는 방법

단일 서버



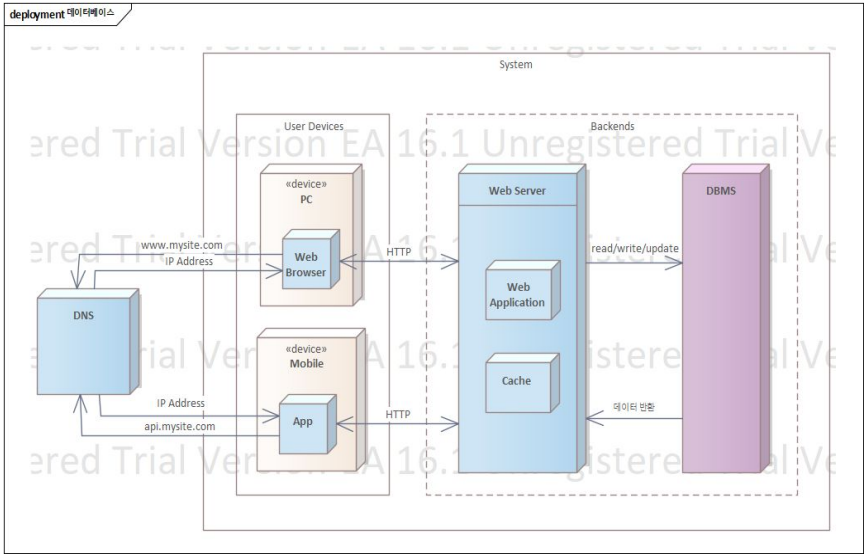
■ 간단한 시스템 설계

- 모든 컴포넌트가 단 한 대의 서버에서 실행
- 처리 흐름
  - 사용자 단말이 DNS에 도메인 이름으로 IP 주소 요청
  - DNS가 사용자 단말에게 IP 주소 반환
  - 사용자 단말이 해당 IP 주소로 특정 기능 요청(HTTP)
  - 웹 서버는 사용자단말에게 응답(HTML, JSON) 반환
- 웹 애플리케이션 구성
  - 프리젠테이션 레이어(HTML/Javascript 등)
  - 비즈니스 및 데이터 레이어(Java, Python 등)

데이터베이스

■ 데이터베이스 서버 분리

○ 웹/모바일 처리, 데이터 처리 독립적으로 확장가능



■ 어떤 데이터베이스를 사용할 것인가?

○ 관계형(Relational DBMS)

- 자료를 테이블의 열/컬럼으로 표현, SQL/Join 사용
- 예) Oracle, MySQL, PostgreSQL 등

○ 비관계형(NoSQL)

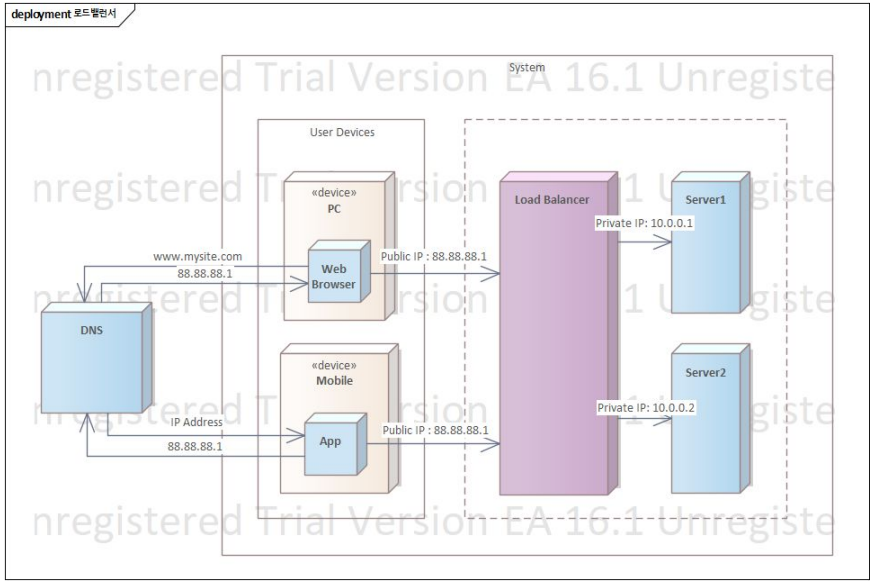
- 키-값 저장소 - Redis, Amazon DynamoDB
- 그래프 저장소 - Neo4J
- 컬럼 저장소 - HBase, Cassandra
- 문서 저장소 - MongoDB, CouchDB
- 대용량/고성능 요건, 비정형 데이터(JSON 등)에 적합

수직적 규모 확장 vs 수평적 규모 확장 (1/4)

- 수직적 규모 확장 - 스케일 업(scale up)
  - 단일 서버의 자원을 추가하여 성능을 향상
    - CPU, RAM 등을 추가/업그레이드
    - 트래픽의 양이 적을 경우 단순하게 확장 가능
  - 심각한 단점
    - 확장에 한계가 있음 (CPU, 메모리 무한대로 증설불가)
    - 서버 장애 시 웹사이트/앱 서비스 중단됨

- 수평적 규모 확장 - 스케일 아웃(scale out)
  - 더 많은 서버를 추가하여 성능을 향상
    - 로드 밸런서로 트래픽을 여러 서버로 분산
    - 데이터베이스 클러스터를 설정하여 성능을 향상
  - 대규모 애플리케이션 지원에 적합
  - **부하분산기 또는 로드밸런서 도입 필요**

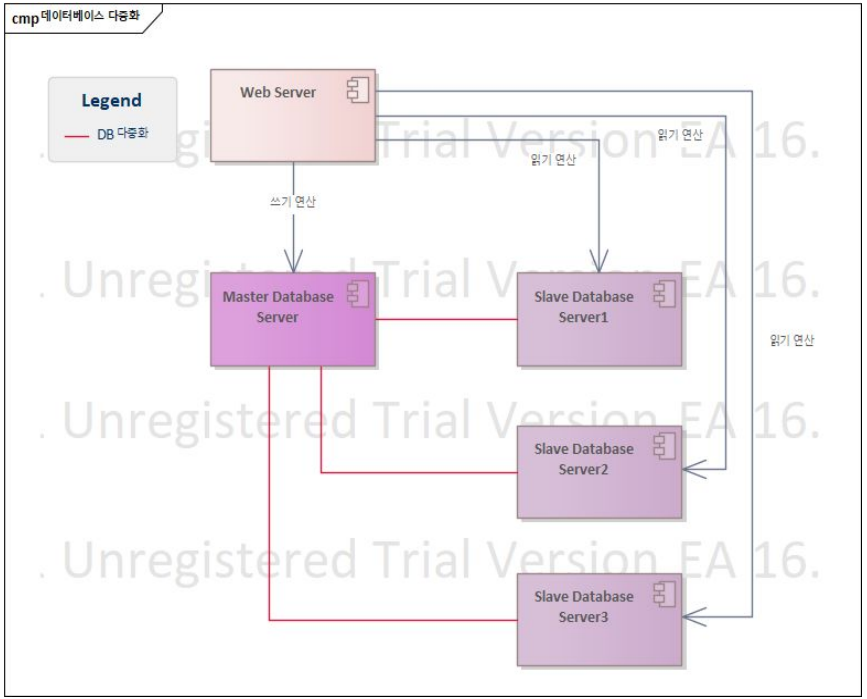
수직적 규모 확장 vs 수평적 규모 확장 (2/4)



■ 로드밸런서

- 사용자 단말 접속 처리 및 트래픽 부하 분배
  - 웹 서버는 사용자 단말의 접속을 직접 처리하지 않음
  - 사용자 단말은 로드밸런서의 공개 API로 접속
  - 로드밸런서와 웹서버간 통신에 사설 IP를 사용 (보안)
- 장애복구 문제 해소 및 웹 계층 가용성 향상
  - 서버1 다운 시 모든 트래픽은 서버2로 전송되어 사이트 전체가 다운되는 상황 방지
  - 트래픽 증가 시 웹 서버 계층에 더 많은 서버 추가하면 로드밸런서가 자동으로 트래픽 분산 처리

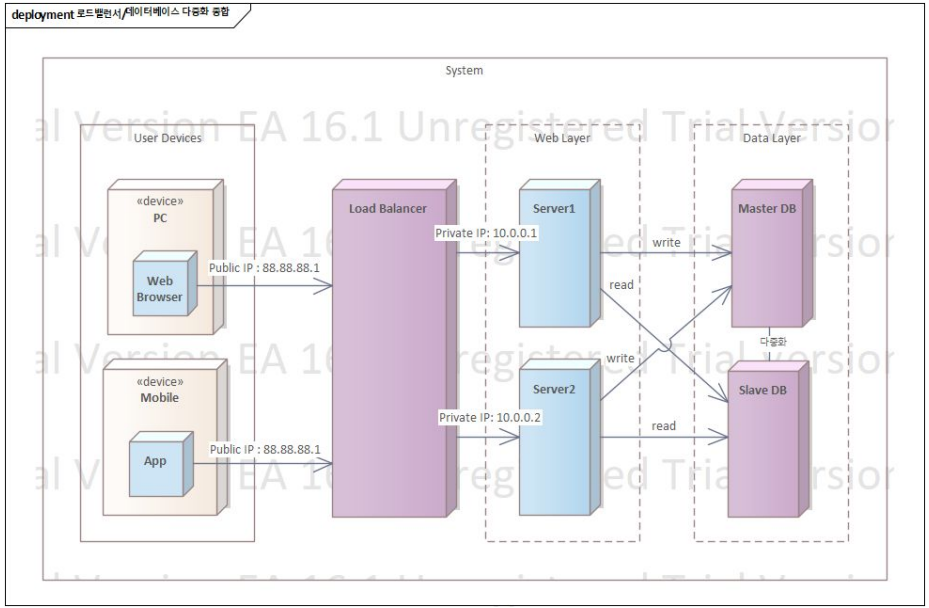
수직적 규모 확장 vs 수평적 규모 확장 (3/4)



- 데이터베이스 다중화
  - 주(master)/부(slave) 데이터베이스 관리
    - 원본은 주 DB에서 관리(쓰기 연산)
    - 사본은 주 DB에서 부 DB로 전달(부 DB는 읽기 연산만)
  - 성능, 안정성, 가용성 향상
    - 읽기연산은 다수의 부 DB로 분산, 병렬처리 증가
    - DB서버가 파괴되어도 데이버 보존(원거리 개별지역)
    - 하나의 DB에 장애가 발생해도 서비스 제공 가능
  - 부 DB 장애 시 (신규 부 DB로 추가/대체 전까지)
    - 부 DB 다수인 경우 읽기 연산 다른 부 DB 분산
    - 부 DB 한대인 경우 읽기 연산 주DB가 한시적 수행
  - 주 DB 장애 시 (신규 부 DB 추가)
    - 부 DB가 새로운 주 DB로 변경 (읽기/쓰기 병행)

수직적 규모 확장 vs 수평적 규모 확장 (4/4)

- 로드밸런서와 데이터베이스 다중화 적용



캐시

- 캐시 계층
  -
- 캐시 사용 시 유의할 점
  -

콘텐츠 전송 네트워크(CDN)

-
-

무상태(stateless) 웹 계층

- 상태 정보 의존적인 아키텍처
  -
- 무상태 아키텍처
  -

데이터 센터

메시지 큐

-

-

로그, 메트릭 그리고 자동화

-

-

데이터베이스의 규모 확장

- 상태 정보 의존적인 아키텍처
  -
- 무상태 아키텍처
  -

백만 사용자, 그리고 그 이상

- 시스템 규모 확장을 위한 기법
  - 웹 계층은 무상태 계층으로
  - 모든 계층에 다중화 도입
  - 가능한 한 많은 데이터를 캐시할 것
  - 여러 데이터 센터를 지원 할 것
  - 정적 콘텐츠는 CDN을 통해 서비스 할 것
  - 데이터 계층은 샤딩을 통해 그 규모를 확장 할 것
  - 각 계층은 독립적 서비스로 분할 할 것
  - 시스템을 지속적으로 모니터링하고, 자동화 도구들을 활용 할 것