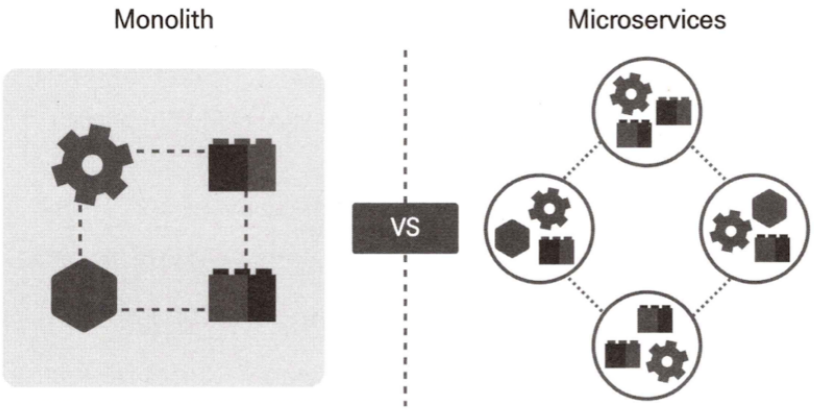
1장. 마이크로서비스 아키텍처, NoSQL, 레디스

NoSQL 등장

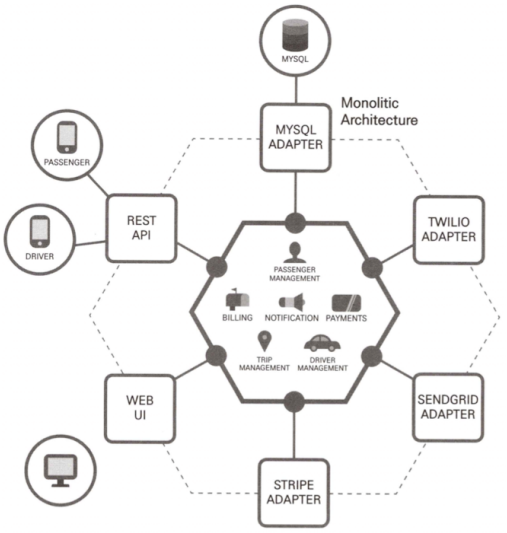
개요

- 소프트웨어의 핵심은 데이터이며, 데이터 저장소는 애플리케이션의 성능과 확장성, 가용성, 신뢰성과 직접적으로 연관을 갖는다.
- 마이크로서비스 어카틱처로 변화하면서 데이터 저장소 특징 역시 다양하게 발전하고 있다.

Monolith vs Microserivces

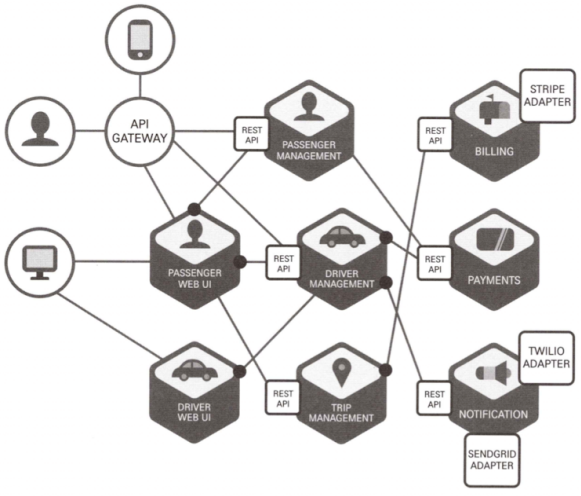


모놀리틱 아키텍처



- 작은 규모에 적합
- 단점
  - 하나의 모듈 수정하면 전체 다시 빌드 및 배포 → 빌드 시간 늘어남
  - 트래픽, 트랜잭션 요구사항에 유연하게 대처 불가
  - 프레임워크, 언어 변경이 전체 애플리케이션에 영향 끼침
  - 작은 기능 변경하는 것도 민첩하게 대처하기 어렵다. 업데이트와 릴리즈가 느려짐

마이크로서비스 아키텍처



- 독립된 각각의 모듈을 조립해 하나의 서비스를 만드는 아키텍처
- 기능별로 작게 나뉘어진 서비스가 독립적으로 동작하는 서비스를 의미

- 업데이트, 테스트, 배포, 확정은 각 서비스별로 독립적으로 수행
- 단점
  - 소규모 팀에서는 관리 복잡도와 운영 부담이 증가할 수 있다.

데이터 저장소 요구 사항의 변화

RDBMS (Relational Database Management System)

- **개요**
  - 그동안 모놀로틱 서비스에서 가장 많이 사용된 데이터베이스
  - 고정된 스키마를 가지며, 각 테이블 간의 관계를 정확하게 규정한다.
  - 애플리케이션이 커질수록 매우 복잡해지며 데이터를 추출하기 위한 쿼리 또한 복잡해진다.
  - 복잡성으로 인해 쿼리 성능 문제가 발생하며 성능 향상을 위해 쿼리, 인덱스, 테이블 구조를 자주 최적화해야 한다.
- **특성**
  - ACID
    - Atomicity
      - 원자성으로 트랜잭션이 완벽하게 실행되거나 아예 실행되지 않음을 보장
    - Consistency
      - 일관성으로 트랜잭션은 실행 전후로도 제약 조건을 만족시킴을 보장
    - Isolation
      - 독립성으로 트랜잭션 실행 시 다른 트랜잭션의 개입이 없음을 보장
    - Durability
      - 성공적으로 수행된 트랜잭션은 영원히 반영돼야 함을 보장

비정형 데이터 증가 (관계형 데이터베이스만 고집할 이유는 없다)

- 비정형 데이터는 다차원적이거나 깊은 계층 구조를 가질 수 없어 관계형 데이터베이스의 정형화된 테이블에서는 관리하기가 어렵다.

- 관계형 데이터베이스로의 변환 작업이 쉽지 않다.
    - 그래프 데이터 → 관계형 : 다시 계산해야한다.
    - 시계열 → 관계형: 쓰기 성능 이슈
    - JSON → 관게형: 다양한 필드가 있어서 필드를 먼저 분석하고 컬럼을 정의해야 해서

NoSQL 이란?

개요

- No SQL 혹은 Not Only SQL
- Standard Query Langauge를 사용하지 않는 데이터 저장소
- 여러 타입이 있지만 공통적인 건 관계가 정의돼 있지 않은 데이터를 저장한다.

특징

대부분의 NoSQL이 갖고 있는 일반적인 특징을 간단히 알아보자

실시간 응답

- 사람은 100ms 넘어가면 지연을 느낀다. 따라서 데이터 저장소에서 데이터를 가져올 때는 0~1ms 이내에서 가져오는 것이 좋다.
- 마이크로서비스에서는 특히 빠른 응답 속도가 중요하다 → 개별 서비스가 빠르게 동작하지 않으면 서비스 자체가 병목 현상을 유발할 수 있기 때문

확장성

- 서비스의 업그레이드로 인한 계획적 확정뿐만 아니라 새해 첫날의 트래픽 트랜잭션 증가에 유연하게 확장할 수 있다.

고가용성

- 대부분의 애플리케이션에서 데이터 저장소는 매우 중요하며, 사용할 수 없는 상황은 곧 서비스 장애로 이어진다.
- 신속하게 복구돼 항상 사용할 수 있는 상태를 유지해야 한다.

클라우드 네이티브

- Cloud Provider 가 제공하는 DBaas(Database-as-a-service)를 사용하면 직접 설치, 운영할 필요 없이 바로 사용할 수 있다.

단순성

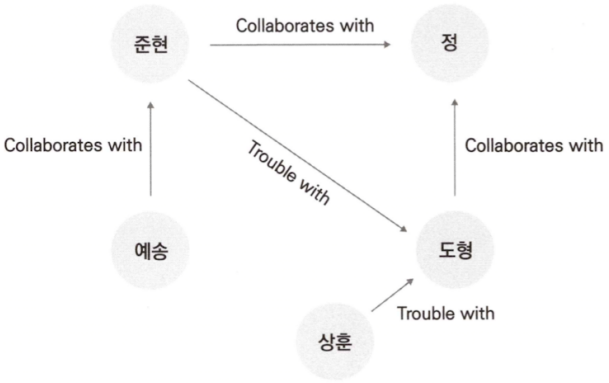
- 서비스 별로 적절한 데이터

유연성

- 디지털 산업의 발전 → 다양한 데이터 유형 → 비정형 데이터 많아짐 → 적합한 형식으로 저장할 수 있는 데이터 저장소 → NoSQL 에 반영
- 관계형 데이터베이스보다 다양한 방식으로 비정형 데이터를 저장할 수 있는 방법을 제공

NoSQL 데이터 저장소 유형

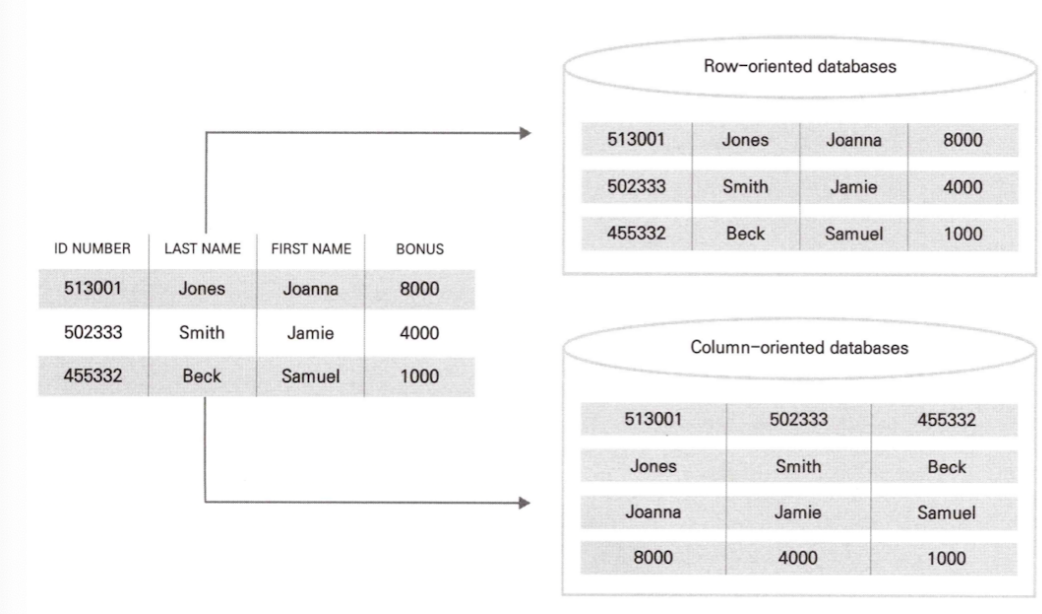
GDB(Graph Database)



- 엔티티 간의 관계를 효율적으로 저장하도록 설계
- 노드, 에지, 속성으로 데이터를 나타냄
  - 노드: 데이터 엔티티
  - 에지: 관계, 방향이 있음
- 복잡한 관계를 효율적으로 탐색할 수 있습니다.
- 사용 사례

    - 추천 서비스 (SNS 유저 간 친구 관계)
    - https://d2.naver.com/helloworld/8446520
    - https://blog.wadiz.kr/그래프-데이터베이스로-친구-서비스-도입하기/
- 예
    - Neo4j
    - OrientDB
    - ArangoDB

Column-oriented



- 열 기준으로 저장한다는 철학으로 설계되었으며, 컬럼 지향적, 와이드 칼럼 유형으로도 불린다.
- 하나의 열에 중첩된 키 값 형태로 저장될 수 있어서 기존 관계형 데이터베이스보다 유연한 스키마로 저장할 수 있다.
- 대량 데이터 집계 쿼리를 다른 유형보다 빠르게 처리할 수 있다.
- 예
    - Apache Cassandra
    - Apache HBase

- Google Bigtable

Document-Oriented Databases

A Document

{
"BookID" : "978-1449396091"
"Title" : "Redis – The Definitive Guide",
"BookID" : "Salvatore Sanfilippo",
"Year" : "2021",
}

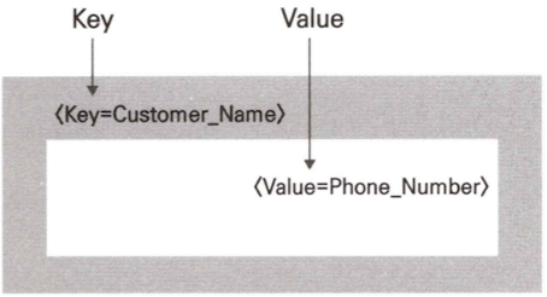
Key	Value
BookID	978-1449396091
Title	Redis – The Definitive Guide
Author	Salvatore Sanfilippo
Year	2021

- JSON, BSON, XML과 같은 문서 형식을 사용하여 데이터를 저장한다.
- 직관적이고 효율적이다.
- 문서는 ID로 구별되며, 컬렉션 안에서 다양한 구조의 문서를 저장할 수 있습니다.
- 스키마가 따로 정해져 있지 않기 때문에 유연성이 크다
- 계층적 트리와 같은 구조를 갖는다.
- 데이터 저장과 검색에 효과적이다.
- 예
  - MongoDB
  - CouchDB
  - Apache Couchbase
  - AWS DocumentDB
- 사례 예시
  - https://www.youtube.com/watch?v=qiElyjkUrKY

키-값 저장소(Key-Value Stores):

Phone directory

Key	Value
Paul	(091) 9786453778
Greg	(091) 9686154559
Marco	(091) 9868564334



- 가장 기본적인 NoSQL 형식으로, 키에 하나의 값이 매핑되어 있다.
- 가장 단순하고 빠름.
- 키가 곧 PK 이다.
- 사례
  - 게임이나 IoT 같은 실시간 서비스에서 사용자 경험을 위해 사용한다.
- 예
  - Redis
  - Amazon DynamoDB (다이나믹 프로그래밍과 같은 용어 선택인가?)
  - AWS ElasticCache
  - Berkeley DB
  - Memcached

- NoSQL 장점
  - 개발팀이 바로 데이터 구조를 바꿀 수 있어 더 빠른 개발이 가능하다 (DBA 가 존재하는 기업의 경우)

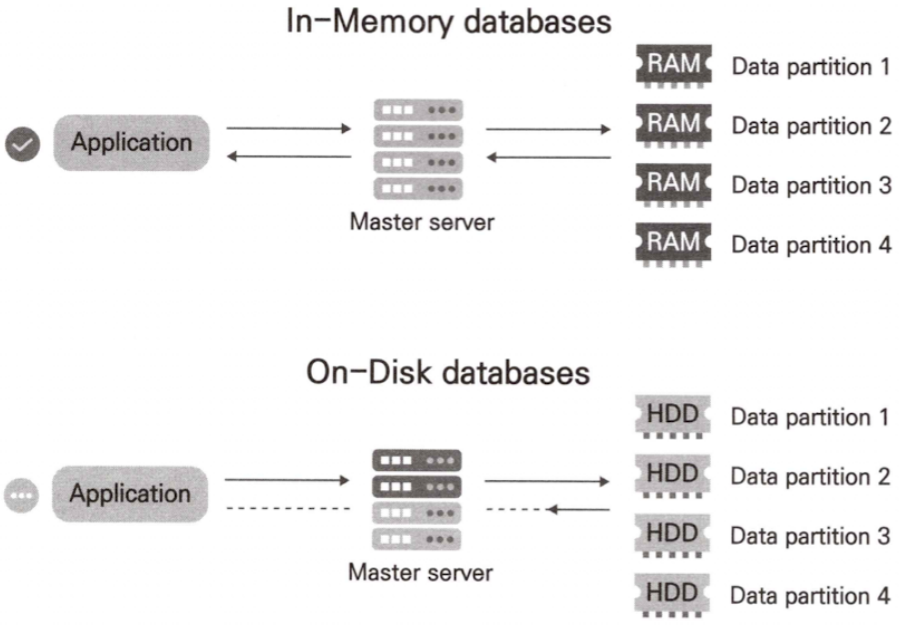
레디스 (Remote Dictionary Server)란

- 고성능 키-값 유형의 인메모리 NoSQL 데이터베이스

- 최신 버전부터는 라이센스가 변화되어서 SSPL(https://news.hada.io/topic?id=14094) 이다.

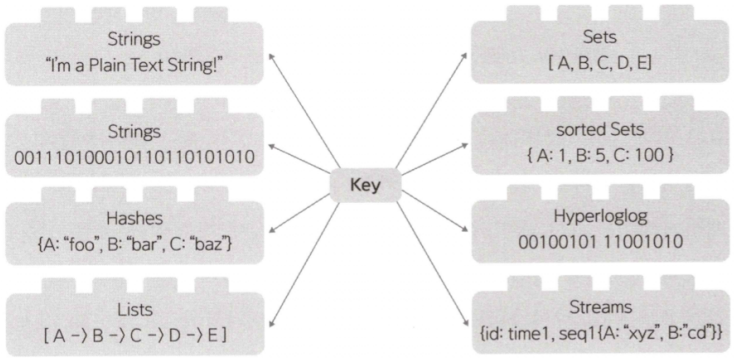
특징

실시간 응답(빠른 성능)

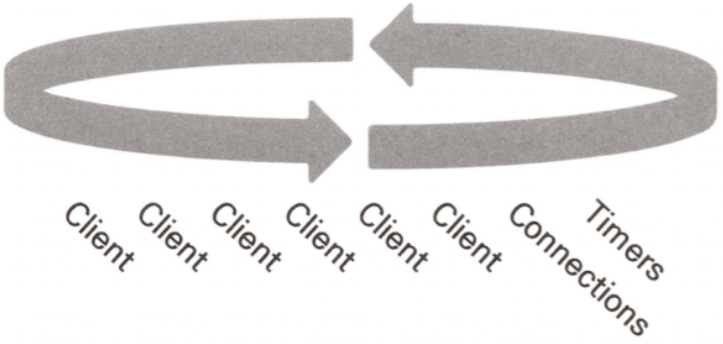


- 아무리 온디스크 데이터베이스가 캐싱을 잘 활용한다고 하더라도 그 전에는 디스크에 가서 데이터를검색해야함
- 디스크에서 데이터를 읽어오는 과정은 매우 비효율적, 접근 빈도가 늘어날 수록 성능은 계속 저하됨

단순성



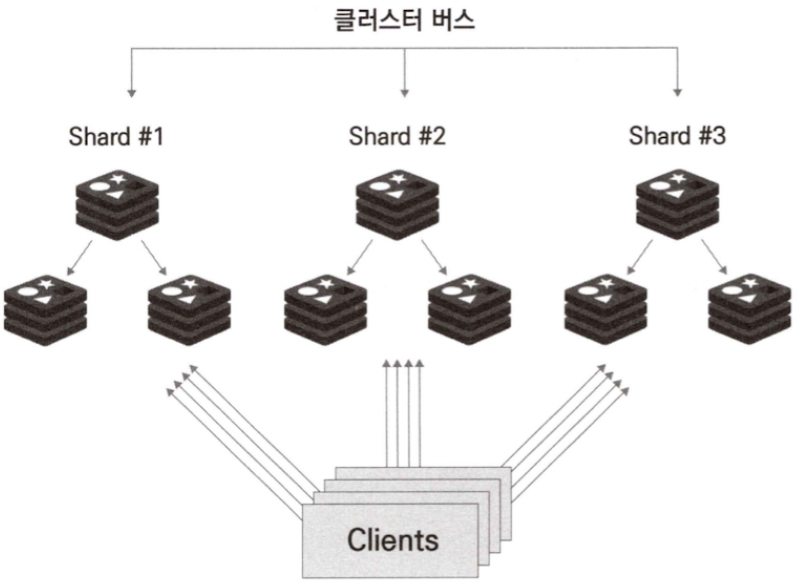
- 키-값 형태로 데이터를 관리할 수 있는 데이터 저장소이어서 추가적인 데이터 가공 없이 애플리케이션에서 데이터 쉽게 사용할 수 있다.
- 임피던스 불일치 (Impedance mismatches: 데이터 베이스 끼리의 충돌을 의미) 해소
  - 내장된 다양한 자료 구조를 통해 불일치를 해소해 개발을 편리하게 할 수 있따.
  - 임피던스 전압을 가했을 때 전류가 흐르는 것에 반항하는 정도
- 싱글 스레드로 동작한다 → 하나의 코어만 있어도 사용할 수 있다.
  - 메인 1개, 별도 스레드 총 3개로 동작
  - 다만 다른 작업이 오래 걸리는 커맨드 수행하면 대기해야한다는 단점이 있다. (Single ***point*** of ***failure***)



고가용성

- 자체적으로 High Availability 기능 제공하여 복제를 통해 데이터를 여러 서버에 분산시킬 수 있다.
- Sentinel 을 통해 장애 상황을 자동으로 Fail-over 시킨다.

확장성



- 클러스터 모드 사용하면 손쉬운 수평 확장이 가능하다.
- 데이터는 클러스터 내에서 자동으로 샤딩된 후 저장되며 복제본이 생성될 수 있다.
- 클라우드 환경에 특화된 애플리케이션 개발 및 운영 방식을 클라우드 네이티브라고 하는데 레디스는 이런 환경에서 따른 데이터 액세스 처리를 지원하는 구조로 MSA 연계에서 큰 장점을 지닌다

마이크로서비스 아킥테처와 레디스

데이터 저장소서의 레디스

- 영속성 고민할 수 있으나 Append Only File 와 Redis DataBase 형식으로 디스크에 저장했다가 장애 발생 시 유실된 데이터를 복구할 수 있다.

메시지 브로커로서의 레디스

- 마이크로서비스는 서로 다른 서비스 간에 지속적인 통신이 필요하다.

- pub/sub 기능을 통해 간단한 메시징 기능 사용할 수 있으며, 전달된 데이터는 삭제되기 때문에 모든 상황에 적합하지 않지만 fire-and-forget 패턴이 필요한 간단한 알림 서비스에서 활용 가능
- list 자료 구조
    - 메시징 큐로 활용
    - 데이터가 들어오면 읽어갈 수 있는 블로킹 기능 사용 가능
- stream 자료 구조
    - 스트림 플랫폼으로 활용
    - 카프카에서 영감

테스트