CP1. 신뢰할 수 있고 확장 가능하며 유지보수 하기 쉬운 애플리케이션

Part 1. 데이터 시스템의 기초

1. 신뢰할 수 있고 확장 가능하며 유지보수 하기 쉬운 애플리케이션

▼ 개요

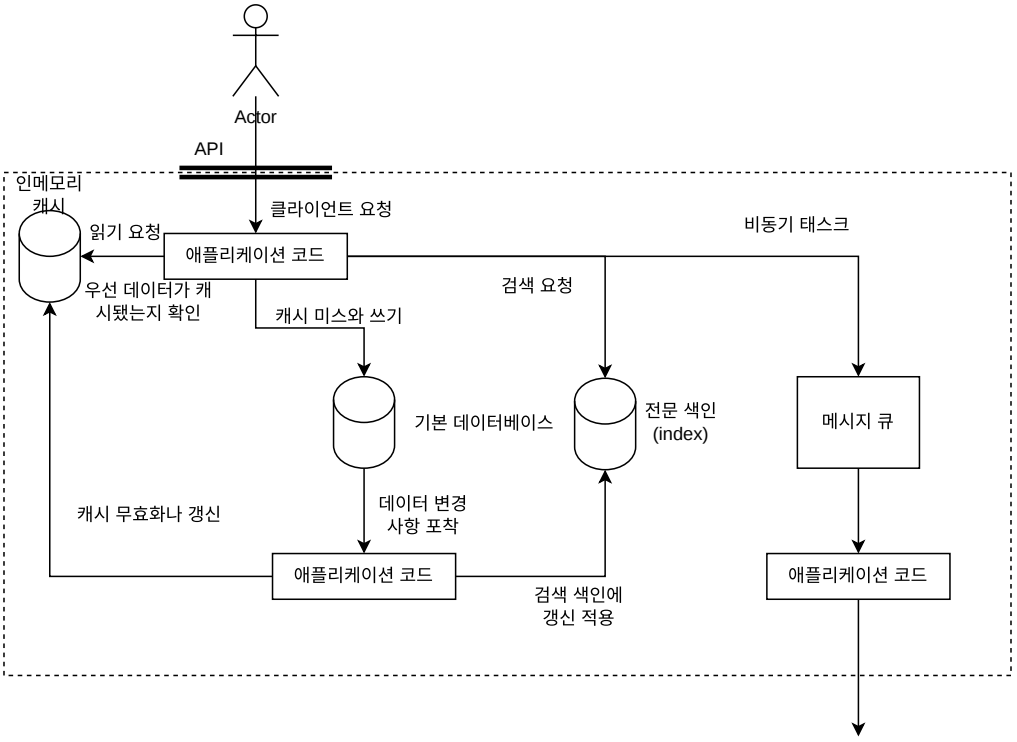
- 오늘날 애플리케이션은 **계산 중심** 보다 **데이터 중심** 적이다.
  이러한 애플리케이션의 경우 CPU 성능은 애플리케이션을 제한하는 요소가 아니며,
  더 큰 문제는
  데이터의 양, **데이터의 복잡도**, **데이터의 변화 속도** 이다.
- 일반적인 애플리케이션의 공통된 기능 (데이터 시스템이 추상화가 잘 되었기 때문)
  - Database
  - Cache (읽기 속도 향상)
  - Search Index (데이터 검색 필터링)
  - Stream Processing (비동기 처리를 위한 다른 Process 메시지 보내기)
  - Batch Processing (주기적으로 대량의 누적된 데이터를 분석)
- **중요한 건, 애플리케이션을 만들 때 어떤 도구와 어떤 접근 방식이 가장 적합한지 판단할 수 있어야 한다.**

▼ 데이터 시스템

- 구현 방식이 각각 다 다른데, 왜 **데이터 시스템**이라는 포괄적 용어를 사용할까?
  1. 새로운 도구들은 다양항 사용 사례에 최적화됐기 때문에 더 이상 전통적인 분류에 딱 들어맞지 않는다. (분류가 흐려지고 있음.)
     - 메시지 큐로 사용하는 datastore 인 레디스(Redis)

- DB 처럼 지속성을 보장하는 메시지 큐인 아파치 카프카(Apache Kafak)

2. 애플리케이션이 단일 도구로는 더 이상 데이터 처리와 저장을 모두 만족할 수 없이 과도하고 광범위한 요구사항을 갖고 있다. 대신 작업은 단일 도구에서 효율적으로 수행할 수 있는 태스크로 나누고 다양한 도구들은 애플리케이션 코드를 이용해 서로 연결한다.
    - Main DB 와 분리된 애플리케이션 관리 캐시 계층 (멤캐시디(Memcached)) 이나 엘라스틱서치(Elasticsearch)나 솔라(Solr) 같은 전문(full text) 검색 서버의 경우 메인 DB와 동기화된 캐시나 색인을 유지하는 것은 보통 애플리케이션 코드의 책임이다.
    - 그림) 다양한 구성 요소를 결합한 데이터 시스템 아키텍쳐의 예



- 서비스 제공을 위해 각 도구를 결합할 때 서비스 인터페이스나 애플리케이션 프로그래밍 인터페이스(API)는 보통 클라이언트가 모르게 구현 세부 사항을 숨긴다. 기본적으로 좀 더 작은 범용 구성 요소들로 새롭고 특수한 목적의 데이터 시스템을 만든다. 복합 데이터 시스템은 외부 클라이언트가 일관된 결과를 볼 수 있게끔 쓰기에서 캐시를 올바르게 무효화하거나 업데이트 하는 등의 특정 보장 기능을 제공할 수 있다.
- 데이터 시스템이나 서비스를 설계할 때 까다로운 문제가 많이 생긴다.
    1. 내부적으로 문제가 있어도 데이터를 정확하고 완전하게 유지하려면 어떻게 해야 할까?
    2. 시스템의 일부 성능이 저하되더라도 클라이언트에 일관되게 좋은 성능을 어떻게 제공할 수 있을까?
    3. 부하 증가를 다루기 위해 어떻게 규모를 확장할까?
    4. 서비스를 위해 좋은 API는 어떤 모습인가?
- 애플리케이션 시스템 설계시 고민해야 하는 점

- 관련자의 기술 숙련도
- 기존 시스템의 의존성
- 전달 시간 척도
- 다양한 종류의 위험에 대한 조직의 내성
- 규제 제약 등

▼ 신뢰성

▼ 확장성

▼ 유지보수성