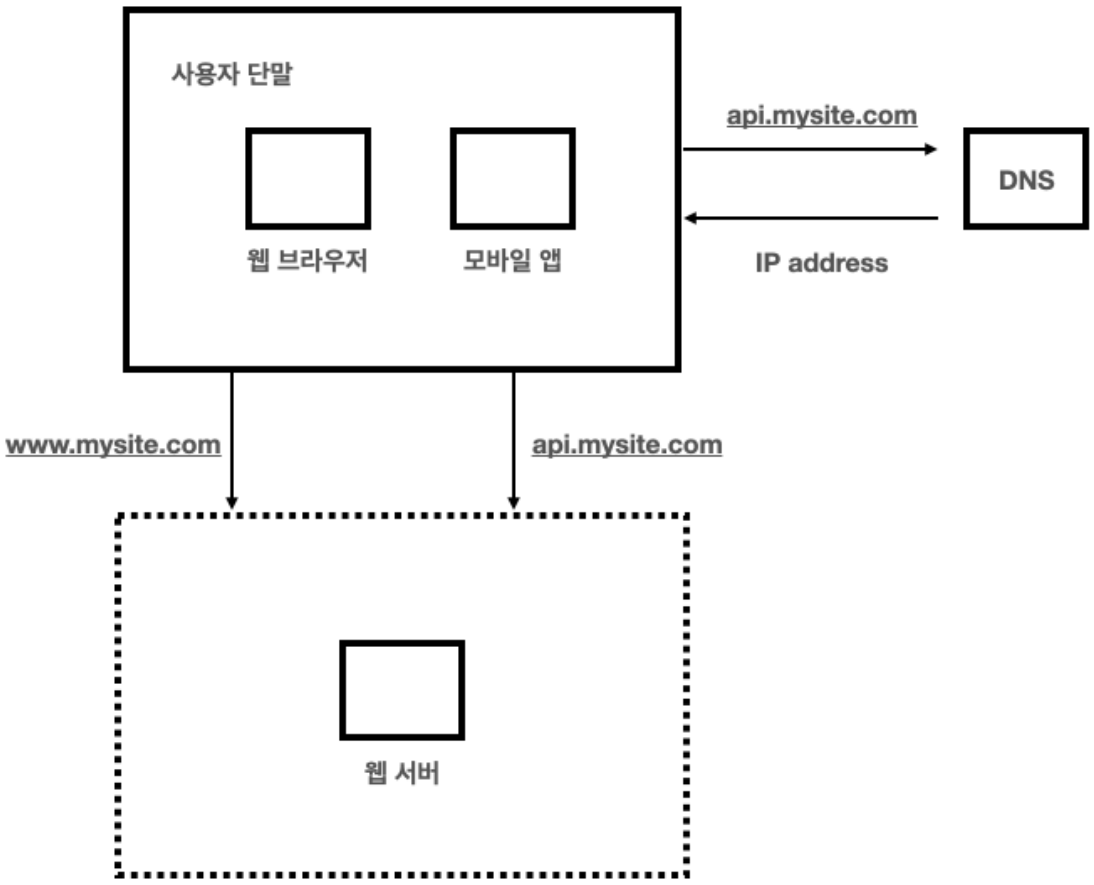
1장. 사용자 수에 따른 규모 확장성

> 목표 : 규모 확장성과 관계된 설계 문제를 푸는 데 쓰일 유용한 지식들 마스터

단일 서버



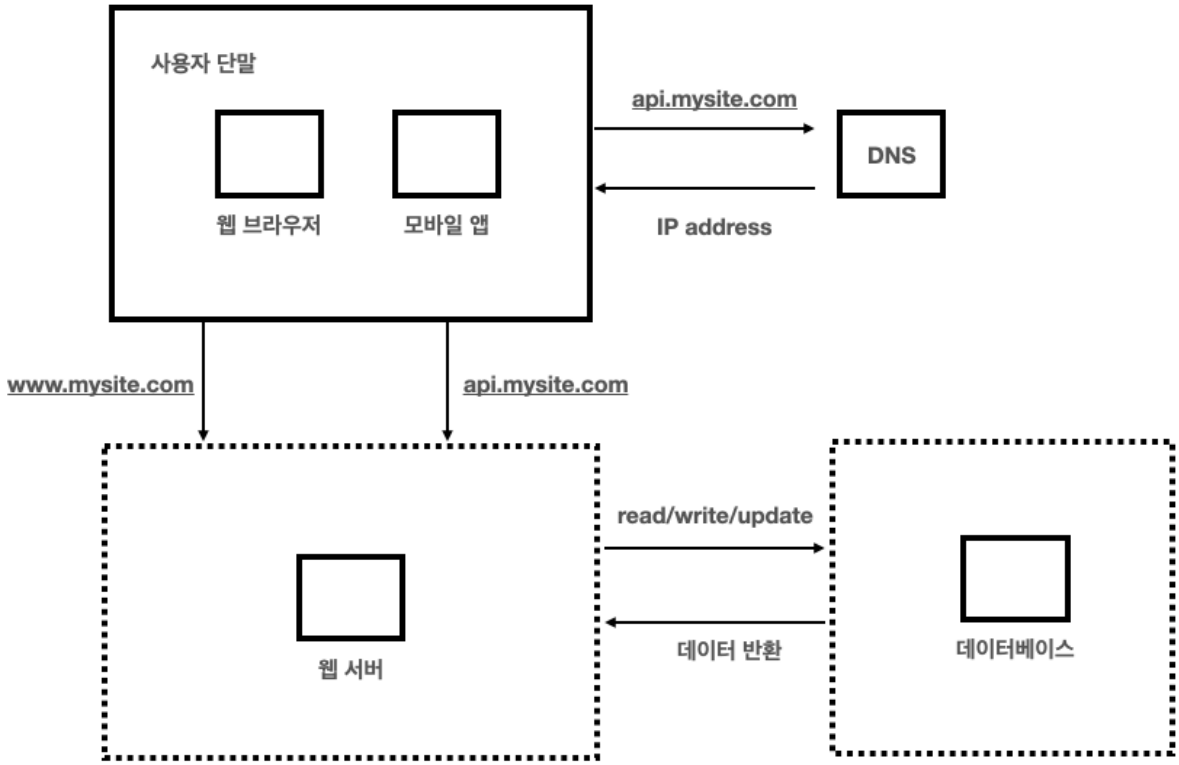
1. 사용자는 도메인 이름을 이용하여 웹 사이트에 접속
2. 해당 접속을 위해서 DNS(Domain Name Service)를 통해 도메인 이름 → IP 주소 변환
3. 해당 IP주소를 HTTP(Hyper Text Transfer Protocol) 요청이 전달된다
4. 요청을 받은 웹 서버는 HTML 페이지나 JSON 형태의 응답 반환

데이터베이스

사용자가 늘면 서버 하나로는 충분하지 않아서 여러 서버를 두어야 한다.

하나는 웹/모바일 트래픽 처리 용도고, 다른 하나는 **데이터베이스**용이다.

웹 계층과 데이터 계층을 분리하여 각각을 독립적으로 확장해 나갈 수 있게 된다.



어떤 DB를 사용할 것 인가?

데이터베이스는 크게 관계형 DB, 비 관계형 DB로 나뉜다.

1. 관계형 DB

- MySQL, 오라클 DB, PostgreSQL 등이 있다.
- 자료를 테이블, 열, 칼럼으로 표현
- SQL를 사용하여 여러 테이블에 있는 데이터를 그 관계에 따라 조인(join) 하여 합칠 수 있다.

2. 비 관계형 DB

- NoSQL이라고 불리고, 대표적으로 CouchDB, Neo4j, Cassandra, HBase, Amazon DynamoDB 등이 있다.
- 다시 네 부류로 나눌 수 있다 → 키-값 저장소(key-value store), 그래프 저장소(graph store), 칼럼 저장소(column store), 문서 저장소(document store)
- 일반적으로 조인 연산은 지원 x

다음 경우에는 NoSQL이 적절한 선택일 수 있다.

- 아주 낮은 응답 지연시간이 요구됨
- 다루는 데이터가 비정형이라 관계형 데이터가 아님
- 데이터를 직렬화하거나 역직렬화할 수 있기만 하면 됨
- 아주 많은 양의 데이터를 저장할 필요가 있음

▼ Graph Database(Neo4j)

데이터가 서로 복잡하게 연결되어 있고, 그 데이터가 다른 데이터와 어떻게 연결되고 참조하는지를 표현해야 할때 적용하면 효과적이다. 전통적인 RDBMS에서는 다대다의 관계이 경우를 말한다. 실시간 추천엔진, 콘텐츠 및 자산관리 시스템, ID 및 엑세스 관리 시스템, 규정 준수 및 위험관리 솔루션, 소셜네트워크와 같은 관계를 보고자 할때 사용한다.

→ **수많은 조인을 줄여줌**

수직적 규모 확장 vs 수평적 규모 확장

수직적 규모 확장의 특징 및 한계

‘스케일 업’ 이라고도 하며, 서버에 고사양 자원을 추가하는 행위를 말한다. 서버로 유입되는 트래픽 양이 적을 때는 좋은 선택이지만, 몇 가지 심각한 단점이 존재한다.

- 한 대의 서버에 CPU나 메모리를 무한대로 증설할 방법은 없다.
- 장애에 대한 자동복구 방안이나 다중화 방안을 제시하지 않는다. 장애 발생 시, 웹 사이트/앱은 완전히 중단된다.

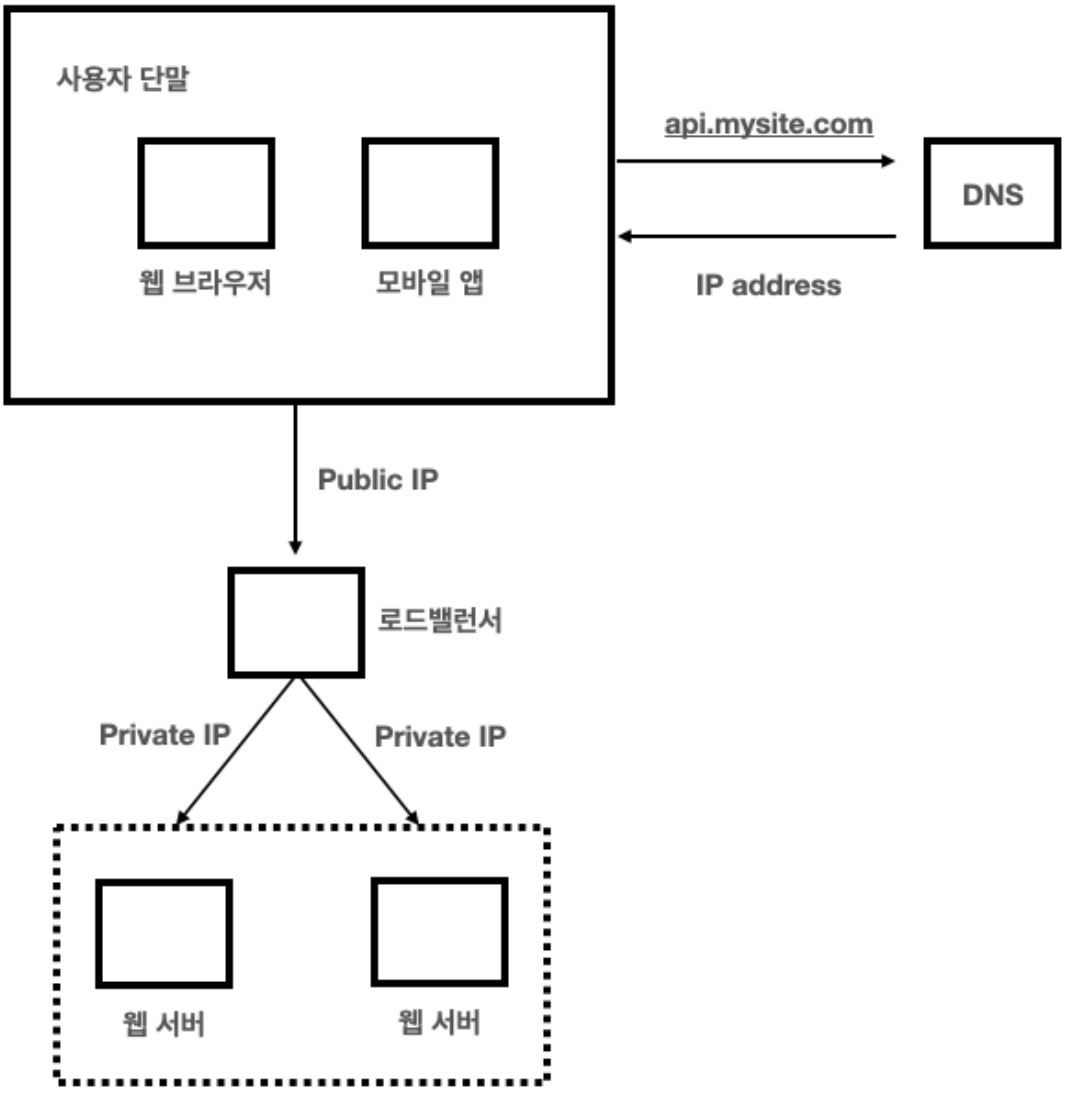
위의 단점 때문에, 대규모 애플리케이션을 지원하는 데는 **수평적 규모 확장법**이 보다 적절하다.

로드밸런서

> 부하 분산 집합에 속한 웹 서버들에게 트래픽 부하를 고르게 분산하는 역할 수행

1. 웹 서버에 바로 연결되는 설계 시, 웹 서버 다운되면 사용자는 웹 사이트 접속 불가
2. 너무 많은 사용자가 접속하면 응답 속도가 느려지거나 서버 접속 불가

→ 이런 문제 해결을 위해 **로드밸런서** 도입



사용자는 로드밸런서의 `Public IP` 로 접속한다. 서버 간 통신에는 `Private IP` 가 이용된다. 윗 그림 처럼 스케일 아웃을 통해서 서버를 여러 대로 늘리게 되면 장애를 자동복구하지 못하는 문제(no failover) 가 해소되며, 웹 계층의 가용성(availability)은 향상된다.

▼ Elastic Load Balancing의 작동 방식

로드 밸런서는 클라이언트에서 오는 트래픽을 허용하고, 하나 이상의 가용 영역에서 등록된 대상(예: EC2 인스턴스)으로 요청을 라우팅합니다. 또한, 로드 밸런서는 등록된 대상의 상태를 모니터링하고 정상 대상으로만 트래픽이 라우팅되도록 합니다. 로드 밸런서가 비정상

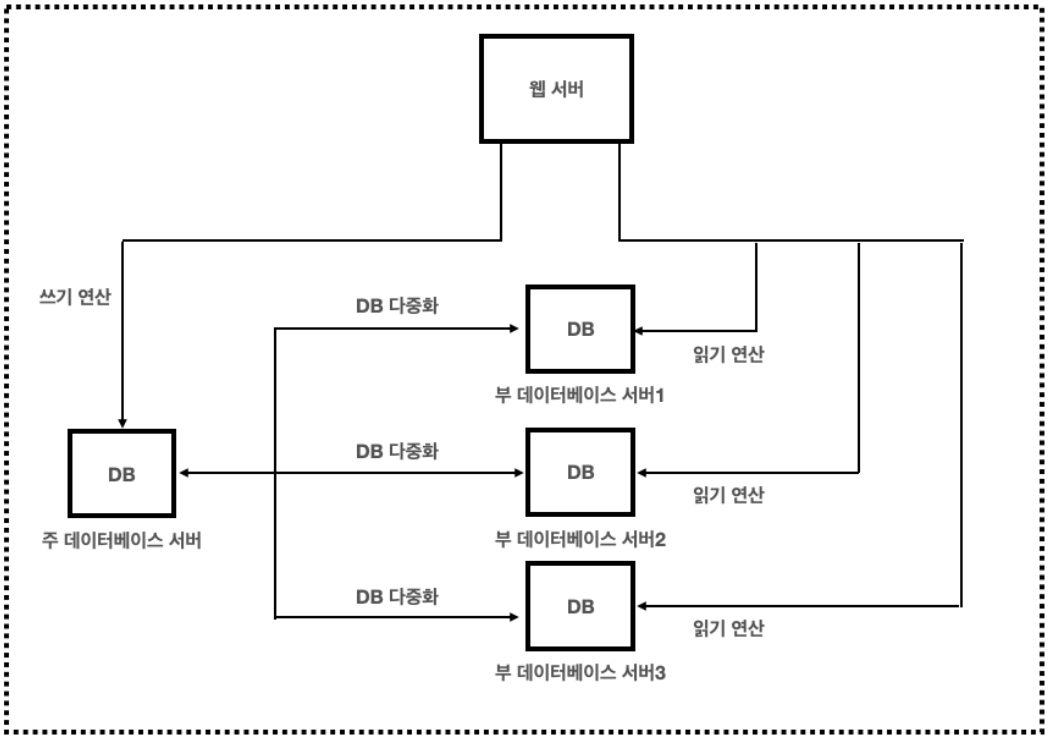
대상을 감지하면, 해당 대상으로 트래픽 라우팅을 중단합니다. 그런 다음 대상이 다시 정상으로 감지되면 트래픽을 해당 대상으로 다시 라우팅합니다.

하나 이상의 리스너를 지정하여 들어오는 트래픽을 허용하도록 로드 밸런서를 구성합니다. 리스너는 연결 요청을 확인하는 프로세스입니다. 클라이언트와 로드 밸런서 간의 연결을 위한 프로토콜 및 포트 번호로 구성됩니다. 마찬가지로 로드 밸런서와 대상 간의 연결을 위한 프로토콜 및 포트 번호로 구성됩니다.

데이터베이스 다중화

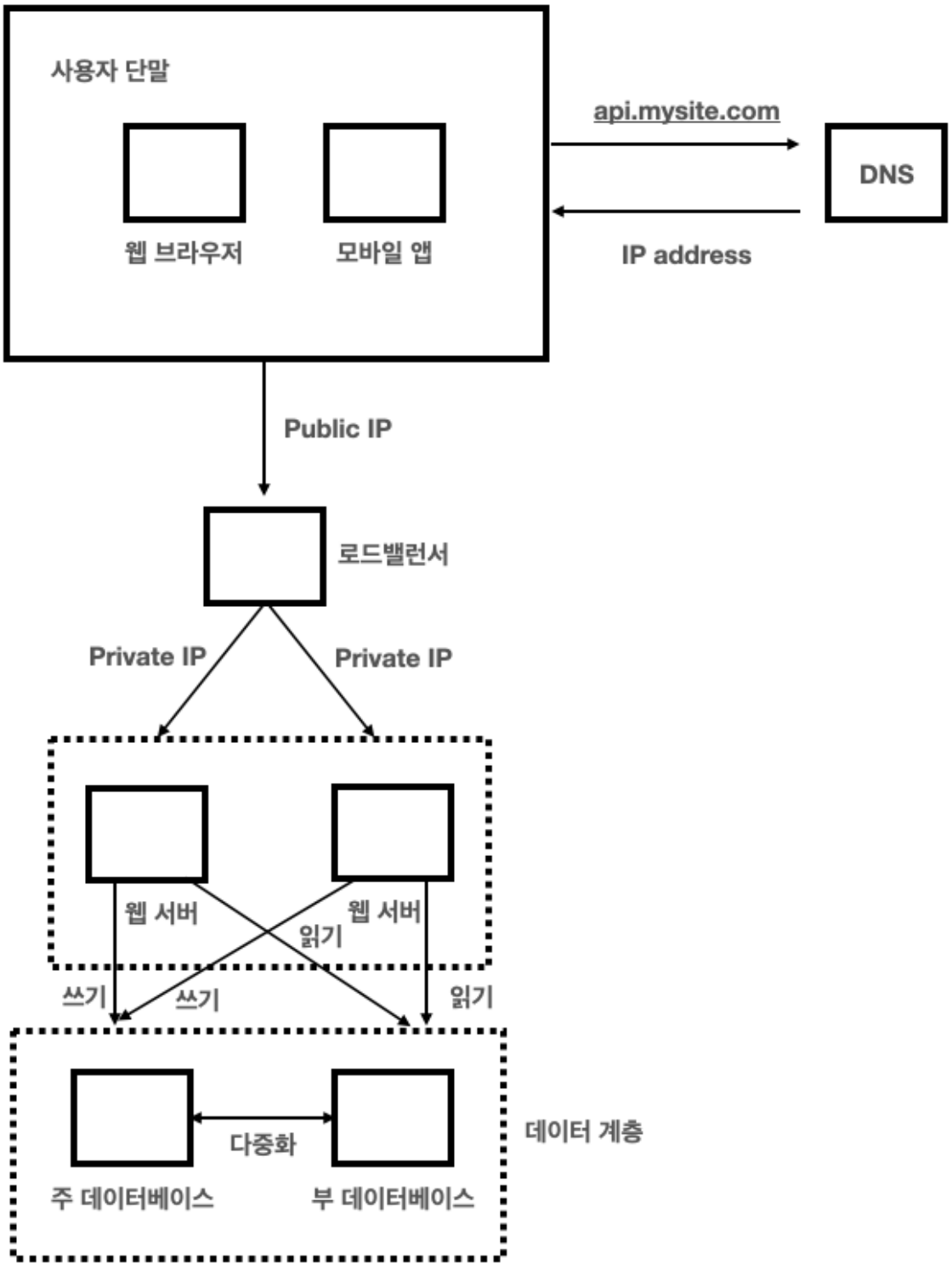
데이터베이스를 Master-Slave 관계로 설정하고, 데이터 원본은 주 서버에, 사본은 부 서버에 저장하는 방식을 주로 사용한다.

대부분의 애플리케이션에서는 쓰기 작업보다는 읽기 작업이 훨씬 많기 때문에 Slave 데이터베이스가 훨씬 많다.



데이터베이스 다중화의 장점

- 더 나은 성능: 데이터 변경 연산은 Master, 조회 연산은 Slave DB 서버로 분산된다. 병렬로 처리될 수 있는 Query 수가 늘어나므로 성능이 좋아진다.
- 안정성: DB 서버 가운데 일부가 파괴되어도 데이터 보존이 가능하다.
- 가용성: 데이터를 여러 지역에 복제해 둠으로써, 하나의 DB 서버에 장애가 발생하더라도 다른 서버에 있는 데이터를 가져와 계속 서비스할 수 있다.



해당 설계안은 다음과 같이 동작한다.

- 사용자는 DNS로부터 로드배런서의 공개 IP 주소를 받는다.

- 사용자는 해당 IP주소를 사용해 로드밸런서에 접속한다.
- HTTP 요청은 서버1이나 서버2로 전달된다.
- 웹 서버는 사용자의 데이터를 부 데이터베이스 서버에서 읽는다.
- 데이터 변경 연산(추가, 삭제, 갱신)은 주 데이터베이스로 전달한다.

응답 시간 개선 - 캐시, CDN

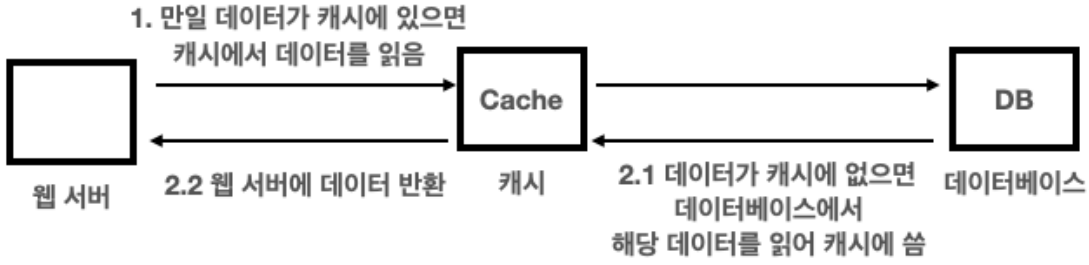
캐시

> 값비싼 연산 결과 또는 자주 참조되는 데이터를 메모리 안에 둠으로써, 뒤이은 요청이 보다 빨리 처리될 수 있도록 하는 저장소

애플리케이션 성능은 데이터베이스를 얼마나 자주 호출하느냐에 크게 좌우되는데, 캐시를 통해 이를 완화할 수 있다.

캐시 계층

데이터가 잠시 보관되는 곳으로 데이터베이스보다 훨씬 빠르다. 성능이 개선될 뿐 아니라 데이터베이스의 부하를 줄일 수 잇고, 캐시 계층의 규모를 독립적으로 확장시킬 수 있다. 다음 사진과 같이 캐시 계층에서의 작업이 이뤄진다.



캐시 사용 시 유의할 점

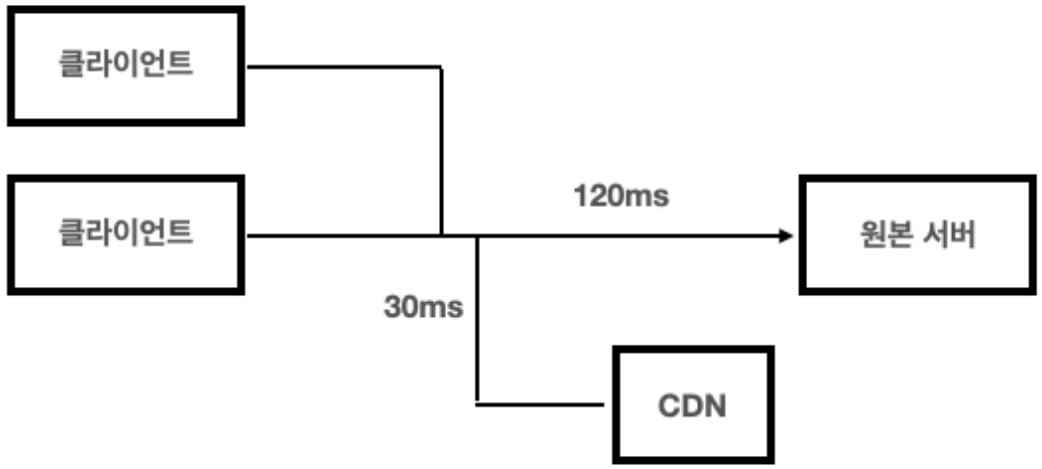
- 갱신이 자주 안 일어나지만 참조가 빈번하게 일어난다면 고려해볼 만하다.

- 캐시는 데이터를 휘발성 메모리에 두므로, 영속적으로 보관할 데이터를 두는 것은 바람직하지 않다.
- 데이터 만료 정책이 없으면 계속 캐시에 남게 된다. 너무 짧으면 데이터베이스를 너무 자주 읽게 되고, 너무 길면 원본과 차이가 날 가능성이 높아져 만료 기한을 잘 고려해서 정해야한다.
- 저장소의 원본을 갱신하는 연산과 캐시를 갱신하는 연산이 단일 트랜잭션으로 처리되지 않는 경우 일관성이 깨질 수 있다. 여러 지역에 걸쳐 시스템을 확장해 나가는 경우 캐시와 저장소 사이의 일관성을 유지하는 것은 어려운 문제가 된다.
- 캐시 서버 한 대만 두는 경우 해당 서버는 단일 장애 지점(Single Point of Failure, SPOF)이 되어버릴 가능성이 있다. 따라서, SPOF를 피하려면 여러 지역에 걸쳐 캐시 서버를 분산시켜야 한다.
- 캐시 메모리가 너무 작으면 데이터가 너무 자주 캐시에서 밀려나버려(eviction) 캐시의 성능이 떨어지게 된다. 이를 방지하려면 캐시 메모리를 과할당(overprovision) 하는 것이다. 이렇게 하면 캐시에 보관될 데이터가 갑자기 늘어났을 때 생길 문제도 방지할 수 있다.
- 캐시가 꽉 차버리면 기존 데이터를 내보내야 한다. LRU, LFU, FIFO 같은 정책들을 적용해야 한다.

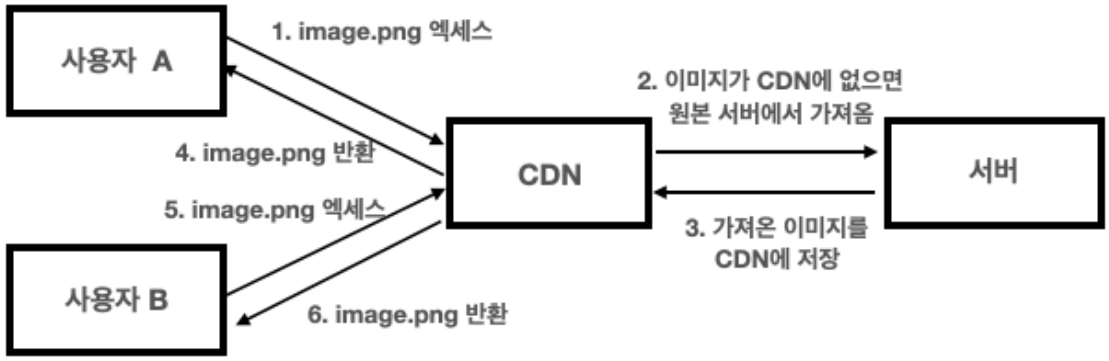
콘텐츠 전송 네트워크(CDN)

> 정적 콘텐츠를 전송하는 데 쓰이는, 지리적으로 분산된 서버의 네트워크

이미지, 비디오, CSS, Javascript 파일 등을 캐시할 수 있다.



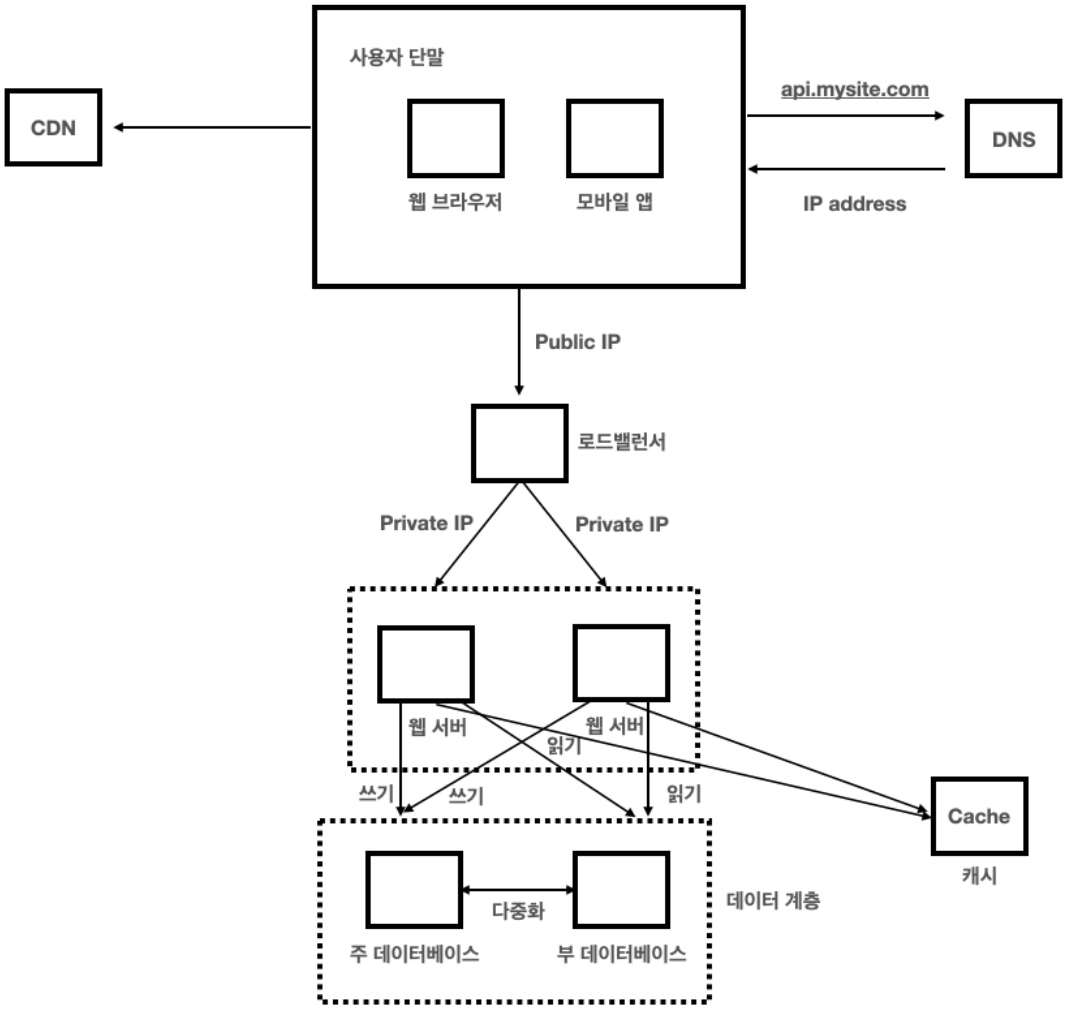
윗 사진에서 볼 수 있듯이, CDN이 사이트 로딩 시간을 개선해준다.



CDN의 동작 방법이다.

1. 사용자 A가 이미지 URL을 이용해 image.png에 접근한다.
2. CDN 서버의 캐시에 해당 이미지가 없는 경우, 서버는 원본 서버에 요청하여 파일을 가져온다.
3. 원본 서버가 파일을 CDN 서버에 반환한다.
4. CDN 서버는 파일을 캐시하고 사용자 A에게 반환한다.

5. 사용자 B가 같은 이미지에 대한 요청을 CDN 서버에 전송한다.
6. 만료되지 않은 이미지에 대한 요청은 캐시를 통해 처리된다.



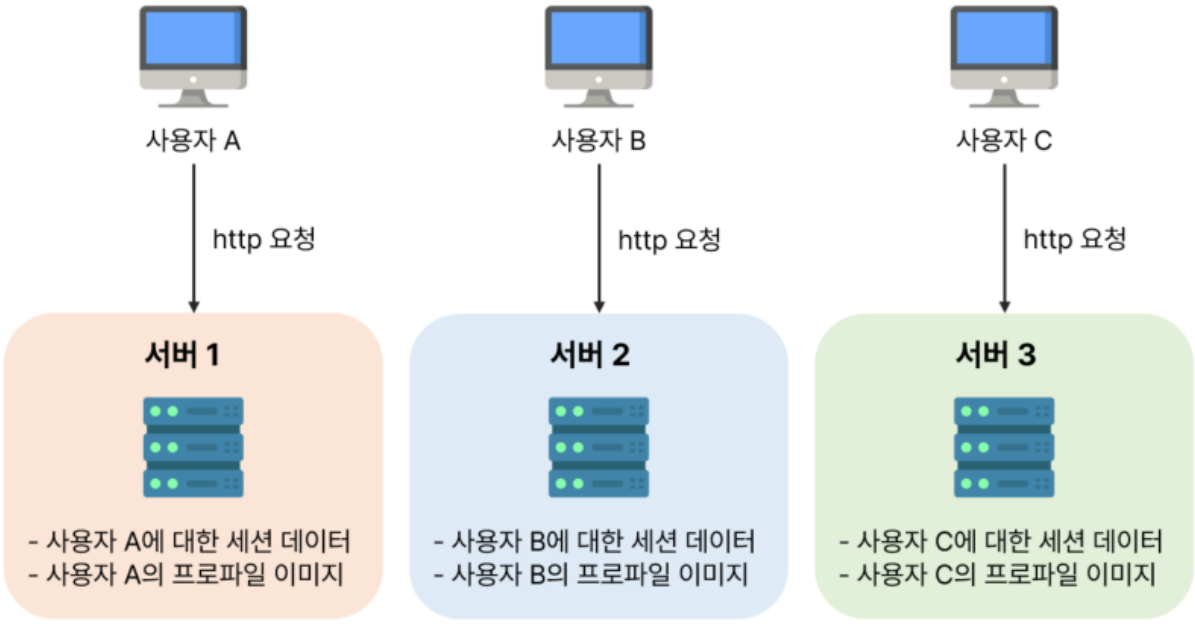
1. 정적 콘텐츠(JS, CSS, 이미지 등)는 더 이상 웹 서버를 통해 서비스하지 않으며, CDN을 통해 제공하여 더 나은 성능을 보장한다.
2. 캐시가 데이터베이스 부하를 줄여준다.

무상태(stateless) 웹 계층

이제는 웹 계층을 수평적으로 확장하는 방법을 고민해보자.

HTTP 같은 경우는 stateless 하기 때문에 세션 같은 정보는 저장해둬야 한다. 세션 정보를 서버 내부 메모리 같은 곳에서 저장할 수 있는데, 만약 서버가 여러 대라면 어떻게 세션 정보를 유지할 수 있을까?

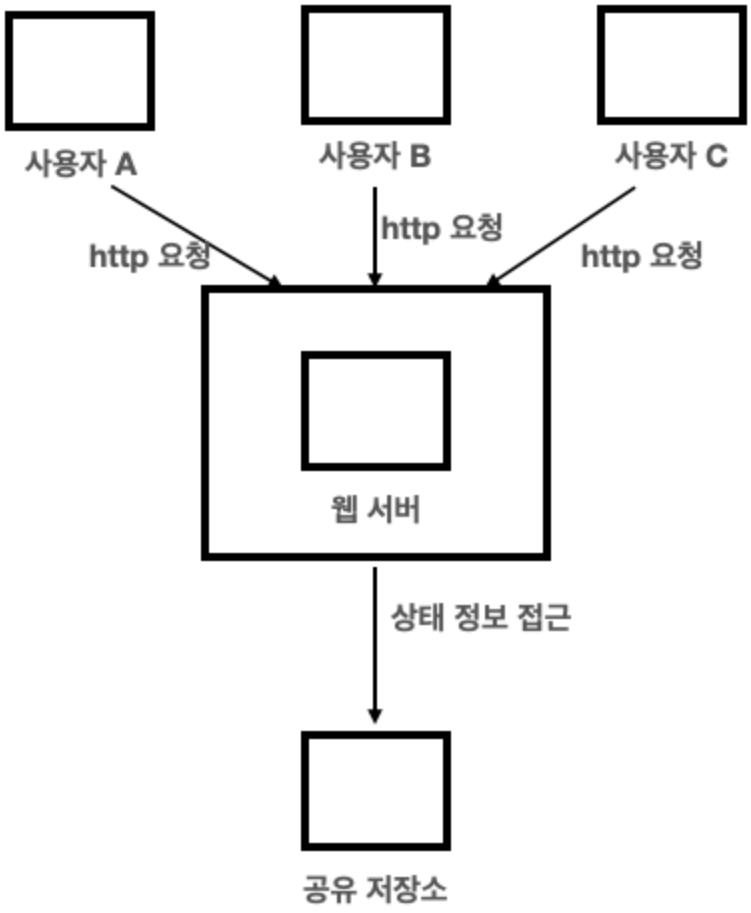
상태 정보 의존적인 아키텍쳐



사용자 A를 인증하기 위해 HTTP 요청은 반드시 서버 1로 전송되어야 한다. 사용자2와 3도 각각 서버 2, 서버 3으로 전송되어야 한다.

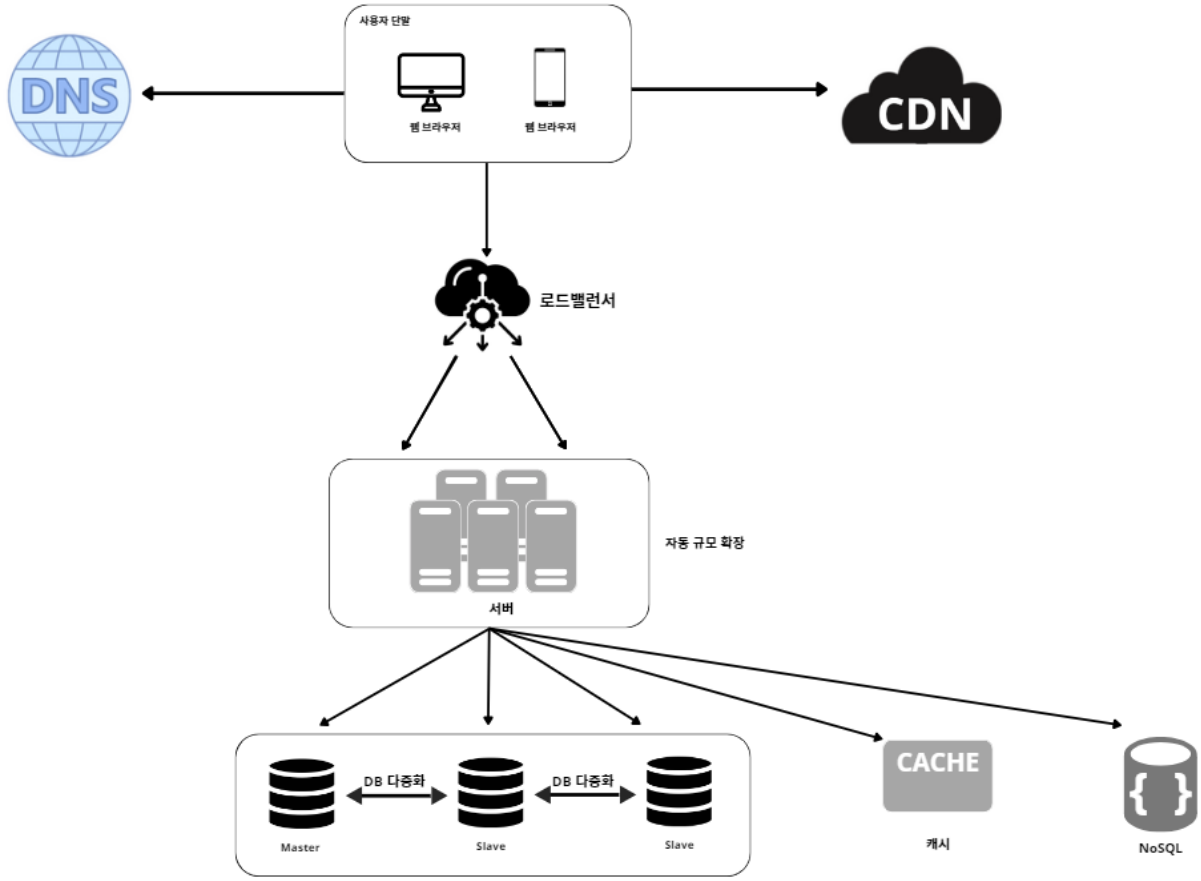
문제는 같은 클라이언트로부터의 요청은 항상 같은 서버로 전송되어야 한다. 대부분의 로드밸런서 이를 지원하기 위해 고정 세션(sticky session) 이라는 기능을 제공하는데, 이는 로드밸런서에 부담을 준다. 게다가 로드밸런서 뒷단에 서버를 추가하거나 제거하기도 까다로워진다. 이들 서버의 장애를 처리하기도 복잡해진다.

무상태 아키텍쳐



웹 서버는 상태 정보가 필요한 경우 공유 저장소(shared storage)로부터 데이터를 가져온다. 따라서 상태 정보는 웹 서버로부터 물리적으로 분리되어 있으므로, 이런 구조는 단순하고, 안정적이며, 규모 확장이 쉽다.

[개발자 배씨 - 대규모 시스템 설계]



세션 데이터를 웹 계층에서 분리하고 지속성 데이터 보관서에 저장되도록 되어있다. 이 공유 저장소는 관계형 데이터베이스일 수도 있고, Memcached/Redis 같은 캐시 시스템일 수도 있으며, NoSQL일 수도 있다.

윗 그림은 NoSQL을 사용하였는데, 규모 확장이 간편해서다. 트래픽 양에 따라 웹 서버를 자동으로 추가할 수 있게되었다.

정리하자면, 이런 경우는 필요에 따라 외부 DB에 세션 정보를 저장하여 관리하는 경우이다.

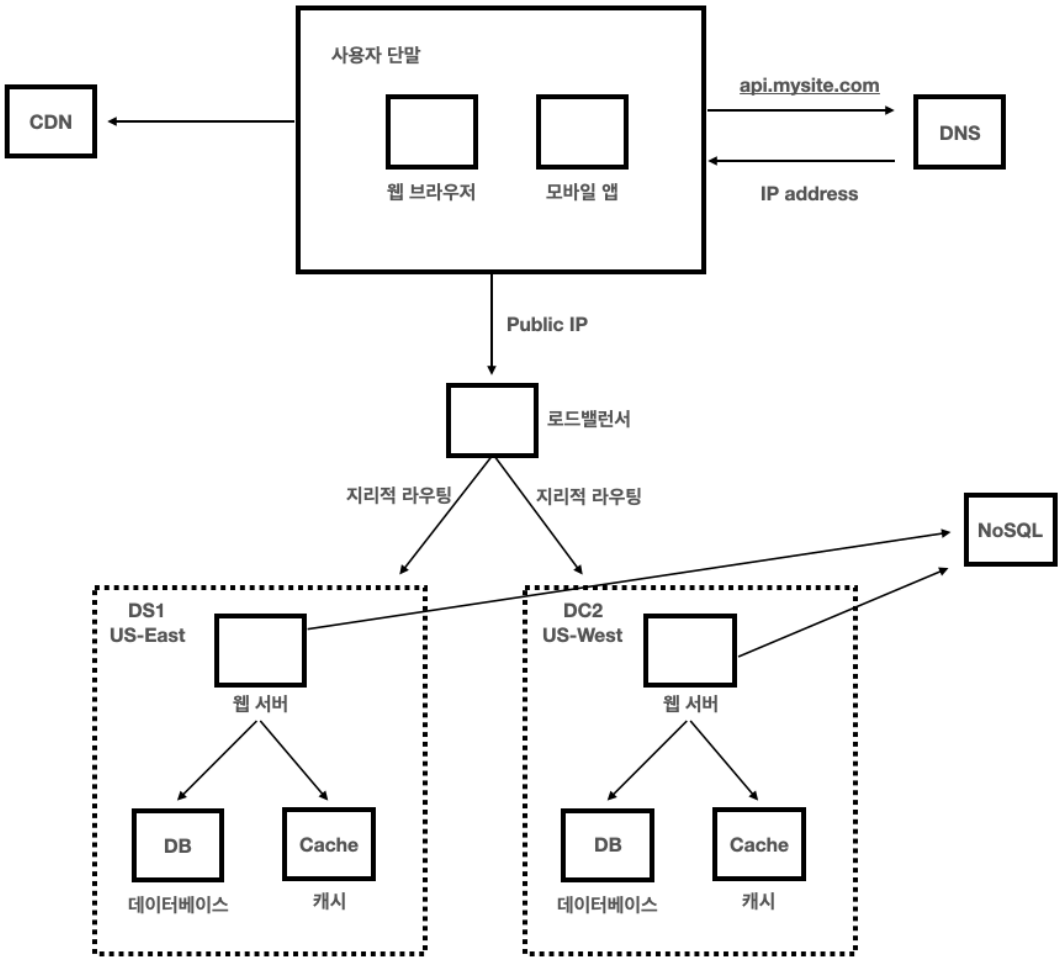
웹서버 통신(http) 특성상 사용자(브라우저)의 이전 상태 client(쿠키) or server(세션) 정보를 기록하지 않는 접속이다. 브라우저가 데이터를 전송할 때마다 연결하고 바로 끊어버리는 방식이다.

- 장점 : 서버의 확장성이 높기 때문에 대량의 트래픽 발생 시에도 대처를 수월하게 할 수 있다.

- 단점 : 클라이언트의 요청에 상대적으로 Stateful 보다 더 많은 데이터가 소모된다. (매번 요청할때마다 자신의 부가정보를 줘야 하니까)

데이터 센터

장애가 없는 상황에서 사용자는 가장 가까운 데이터 센터로 안내되는데, 해당 절차를 **지리적 라우팅**이라고 부른다.



데이터 센터 중 한 곳에 심각한 장애 발생시, 모든 트래픽은 장애가 없는 데이터 센터로 전송된다.

다중 데이터센터를 만들기 위해 고려해야 할 조건

- 트래픽 우회: 올바른 데이터 센터로 트래픽을 보내는 효과적인 방법이 필요하다.
- 데이터 동기화: 데이터 센터마다 별도의 데이터베이스를 다중화하기
- 테스트와 배포: 여러 위치에서 애플리케이션을 테스트 해보고, 자동화 배포가 모든 데이터 센터에 동일하게 설치되도록 하는 것이 중요하다.

메세지 큐

> 메세지의 무손실, 즉 메세지 큐에 일단 보관된 메세지는 소비자가 꺼낼 때까지 안전하게 보관된다는 특성을 보장하는 비동기 통신을 지원하는 컴포넌트

Publish/Producer는 메세지 큐에 발행한다. 큐에는 보통 Consumer/Subscribe 가 메세지를 꺼내서 동작을 수행한다.

메세지 큐를 이용하면 서비스 또는 서버 간 결합이 느슨해져서, 규모 확장성이 보장되어야 하는 안정적 애플리케이션을 구성하기 좋다.

▼ 메세지 큐가 이벤트 드리븐이랑 같은 맥락인가요?

수단으로 보면 된다.

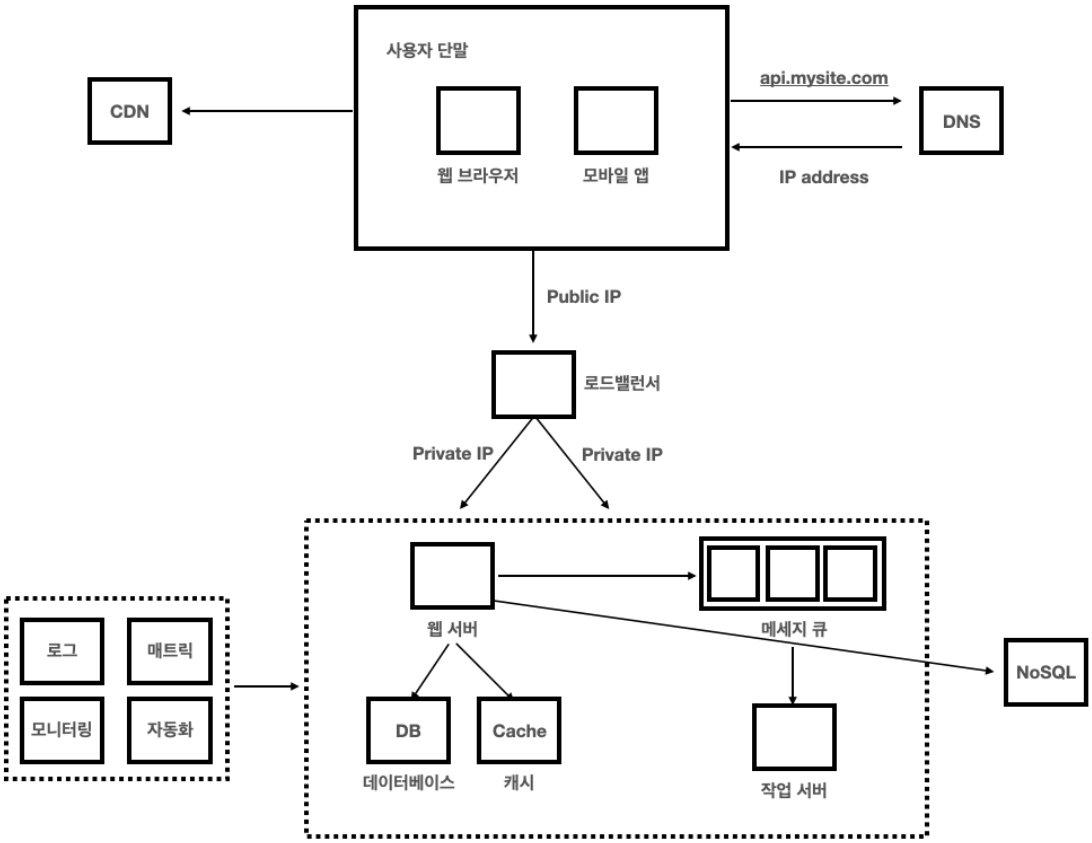
글을 쓰고 분리 → Pub/Sub

MSA에서 회원서비스 쿠폰을 발급하고 싶을 때 → 이벤트 드리븐

로그, 메트릭 그리고 자동화

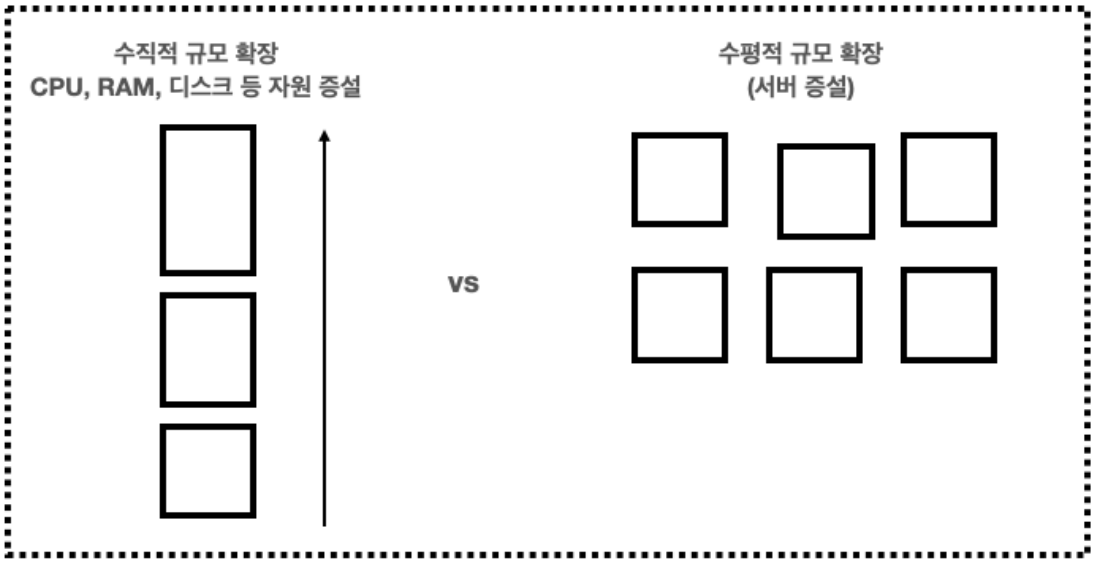
비즈니스의 규모가 커지면 로그, 매트릭, 자동화 도구에 필수적으로 투자해야한다.

- 로그: 에러 모니터링을 위해, 로그를 단일 서비스로 모아주는 도구를 사용할 수 있다.
- 메트릭: 유용한 정보, 시스템의 현 상태를 파악하기 위해 사용한다.
- 자동화: 생산성과 안정성을 높일 수 있다.



데이터베이스의 규모 확장

저장할 데이터가 많아지면 데이터베이스에 대한 부하도 증가한다. 이 때도 애플리케이션 서버처럼 스케일 업, 스케일 아웃 두 가지 방법이 존재한다.



데이터베이스에도 수직적 확장보다는 수평적 확장이 더 적절하다.

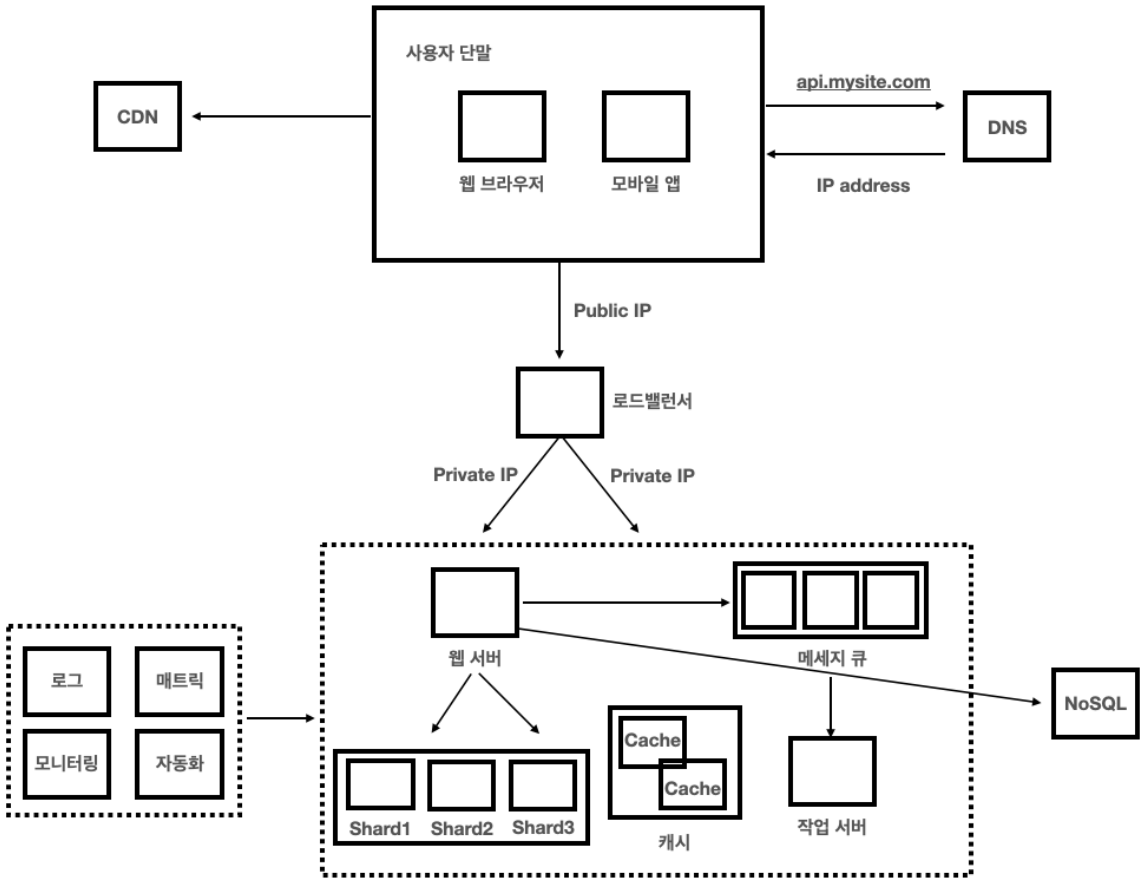
수평적 확장에 대표적으로 **샤딩**이 있다. 샤딩은 대규모 데이터베이스를 샤드라고 부르는 작은 단위로 분할하는 기술을 말한다.

샤딩 전략을 구현할 때 중요한 것은 샤딩 키 이다. 샤딩 키에 따라서 한 곳으로만 부하가 집중되거나, 여러 곳으로 적절히 분산되기 때문이다.

샤딩을 위해 고려해야 할 조건

- 데이터의 재 샤딩
  - 데이터가 너무 많아져서 하나의 샤드로는 더 이상 감당하기 어려울 때
  - 샤드 간 데이터 분포가 균등하지 못하여 어떤 샤드에 할당된 공간 소모가 다른 샤드에 비해 빨리 진행될 때
- 유명 인사: 핫스팟 키 문제라고도 부르는데, 유명인 정보가 저장된 특정 샤드에 쿼리가 집중되어 서버에 과부하가 걸리는 문제
- 조인과 비정규화

  일단 하나의 데이터베이스를 여러 샤드 서버로 쪼개고 나면, 여러 샤드에 걸친 데이터를 조인하기 힘들어진다. 이를 해결하기 위해 데이터베이스를 비정규화하여 하나의 테이블에서 쿼리가 수행될 수 있도록 할 수 있다.



▼ 샤딩과 파티셔닝의 차이점

파티셔닝이란 퍼포먼스, 가용성 또는 정비용이성을 목적으로 논리적인 데이터 엘리먼트들을 다수의 엔티티(table)로 쪼개는 행위를 뜻하는 용어이다.

샤딩은 **수평 파티셔닝(horizontal partitioning)**과 동일하다. 데이터베이스를 샤딩하게 되면 기존에 하나로 구성될 스키마를 다수의 복제본으로 구성하고 각각의 샤드에 어떤 데이터가 저장될지를 샤드 키를 기준으로 분리한다.

예를 들어, 고객의 데이터베이스를 CustomerId를 샤딩 키로 사용하여 샤딩하기로 하였다. 0 ~ 10000 번 고객의 정보는 하나의 샤드에 저장하고 10001 ~ 20000 번 고객의 정보는 다른 샤드에 저장하기로 하였다. DBA는 데이터 엑세스 패턴과 저장 공간 이슈(로드의 적절한 분산 , 데이터의 균등한 저장)를 고려하여 적절한 샤드키를 결정하게 된다.

수직 파티셔닝(vertical partitioning)은 하나의 엔티티에 저장된 데이터들을 다수의 엔티티들로 분리하는 것을 말한다.

예를 들어, 한 고객은 하나의 청구 주소를 가지고 있을 수 있다. 그러나, 데이터의 유연성을 위해 다른 데이터베이스로 정보를 이동하거나 보안 이슈 등을 이유로 CustomerId를 참조하도록 하고 청구 주소 정보를 다른 테이블로 분리할 수 있다.

요약하면 파티셔닝은 퍼포먼스, 가용성, 정비용이성등의 목적을 위해 논리적인 엔티티들을 다른 물리적인 엔티티들로 나누는것을 의미하는 일반적인 용어이다. 수평 파티셔닝 또는 샤딩은 스키마 복제 후 샤드키를 기준으로 데이터를 나누는것을 말한다. 수직 파티셔닝은 스키마를 나누고 데이터가 따라 옮겨가는것을 말한다.

참고링크

https://velog.io/@haron/가상-면접-사례로-배우는-대규모-시스템-설계-기초-1장-사용자-수에-따른-규모-확장성

https://baebalja.tistory.com/584