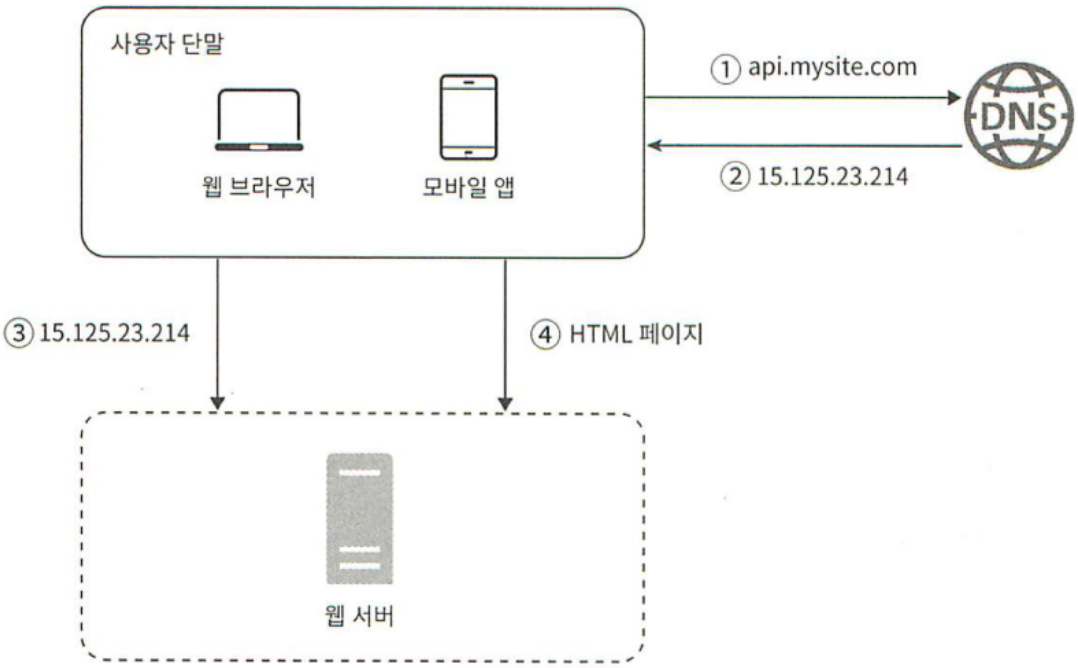
1장. 사용자 수에 따른 규모 확장성

그동안 내가 진행한 프로젝트에서는 대규모 사용자는 생각도 못하고 개발자 4~5명을 위한 단순한 어플만을 구현해왔다. 대규모 시스템에 대해 생각을 못 해본 것은 당연하고 기능 개발에만 초점을 두었던 것 같다. 하지만 최근 면접 준비를 하면서 질문으로 '대용량 트래픽이 들어왔을 때 어떻게 처리할 것인가?'와 관련된 다양한 질문들을 받았고 대답을 잘 하지 못했고 회고를 할 때도 답을 잘 알아내지 못했다. 그래서 특히 이번 장이 흥미롭게 다가왔고 완벽하게는 아니지만 이제 답변을 할 수 있을 것 같다.

단일 서버

- 모든 컴포넌트(웹, 앱, 데이터베이스, 캐시 등)가 단 한 대의 서버에서 실행되는 간단한 시스템부터 설계된다고 가정



1. 도메인 이름을 DNS를 통해 IP 주소로 변환하는 과정

2. DNS 조회 결과로 IP 주소가 반환 (웹 서버 주소)
3. 해당 IP주소로 HTTP 요청이 전달된다.
4. 요청을 받은 웹 서버는 HTML 페이지나 JSON 형태의 응답을 반환한다.

▼ 참고 (요청은 어디서 오는가?)

1. 웹 어플리케이션 : 비즈니스 로직, 데이터 저장 등을 처리하기 위해서는 서버 구현용 언어를 사용하고, 프레젠테이션용으로는 클라이언트 구현용 언어를 사용 (타임리프 같은 건가..)
2. 모바일 앱 : 모바일 앱과 웹 서버 간 통신을 위해 HTTP 프로토콜을 이용. HTTP 프로토콜을 통해서 반환될 응답 데이터의 포맷으로는 주로 JSON을 이용한다.

데이터베이스

사용자 수가 많아지면 단일 서버로는 충분하지 않기에 여러 서버를 두어야 합니다.

웹/모바일 트래픽 처리 서버(웹 계층)과 데이터 베이스 서버 (데이터 계층)을 분리하면 그 각각을 독립적으로 확장해 나갈 수 있다. ⇒ **계층을 분리하자.**

관계형 데이터베이스 vs 비 관계형 데이터베이스

관계형 데이터베이스 (RDBMS)

- 자료를 테이블과 열, 칼럼으로 표현한다.
- SQL을 사용하면 여러 테이블에 있는 데이터를 그 관계에 따라 조인하여 합칠 수 있다.

비 관계형 데이터베이스 (NoSQL)

- 4종류
  - 키-값 저장소
  - 그래프 저장소
  - 칼럼 저장소
  - 문서 저장소
- 조인 연산 지원 X

- 사용하여야 할 때
  - 아주 낮은 응답 지연시간이 요구됨(빨리 데이터를 반환해야함)
  - 다루는 데이터가 비정형이라 관계형 데이터가 아님
  - 데이터를 직렬화하거나 역직렬화 할 수 있기만 하면 됨
  - 아주 많은 양의 데이터를 저장할 필요가 있음

수직적 확장 VS 수평적 확장

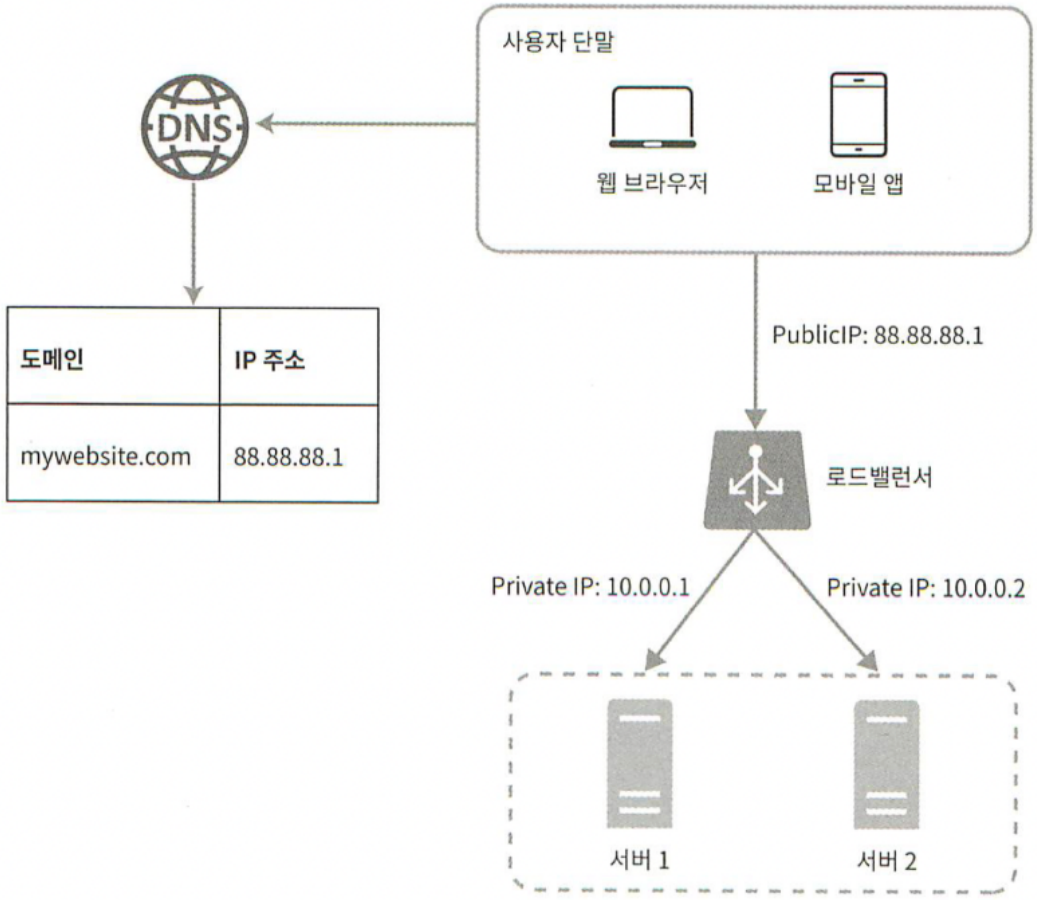
- 수직적 규모 확장 (Scale Up)
  - Upgrade!! 만약 현재 서버가 데이터 받기를 버거워 하면? ⇒ **UPGRADE!! (더 좋은 CPU, 더 많은 RAM)**
  - 단순한 구조라서 서버로 유입되는 트래픽의 양이 적을 때는 수직적 확장이 좋다. 하지만 심각한 단점 존재

    ⇒ CPU나 메모리를 무한대로 증설할 방법이 없음 / 장애시 자동 복구 방안이나 다중화 방안을 제시하지 않음 (즉, 서버에 장애가 발생하면 웹/앱은 완전히 중단)

- 수평적 규모 확장 (Scale Out)
  - 더 많은 서버를 추가하여 성능을 개선하는 것
  - 로드 밸런서를 도입해서 트래픽을 여러 대의 서버에 분산시킨다.

로드 밸런서

로드 밸런서는 부하 분산 집합에 속한 웹 서버들에게 트래픽을 고르게 분산하는 역할을 한다.



주요 키워드 : 공개 IP 주소, 사설 IP 주소

사용자는 로드밸런서의 공개 IP 주소로 접속한다. 따라서 웹 서버는 클라이언트의 접속을 직접 처리하지 않고 보안을 위해 서버간의 통신에는 사설 IP 주소를 이용한다.

사설 IP 주소는 같은 네트워크에 속한 (동일한 로드밸런서에서 공유되는 것) 서버 사이의 통신에만 쓰일 수 있는 주소로 인터넷 IP를 이용해서는 접속할 수 없습니다.

로드밸런서로 부하를 나누면 장애를 자동으로 복구하지 못하는 문제는 해소, 웹 계층의 가용성은 향상된다.

- 웹 서버 계층에서 하나의 서버에 장애가 발생하면 다른 서버가 모든 트래픽을 받는다. 즉, 웹 서버 전체가 다운되는 일이 방지되고 부하를 나누기 위해서 서버를 추가할 수도 있다.

- 웹사이트로 유입되는 트래픽이 가파르게 증가하면 두 대의 서버로 트래픽을 감당할 수 없는 시점이 오는데, 로드 밸런서를 이용하면 그냥 여러 서버를 추가하면 된다! 그러면 로드밸런서가 자동으로 트래픽을 분산하기 시작할 것이다.

⇒ 웹 계층은 어느정도 이해 완료

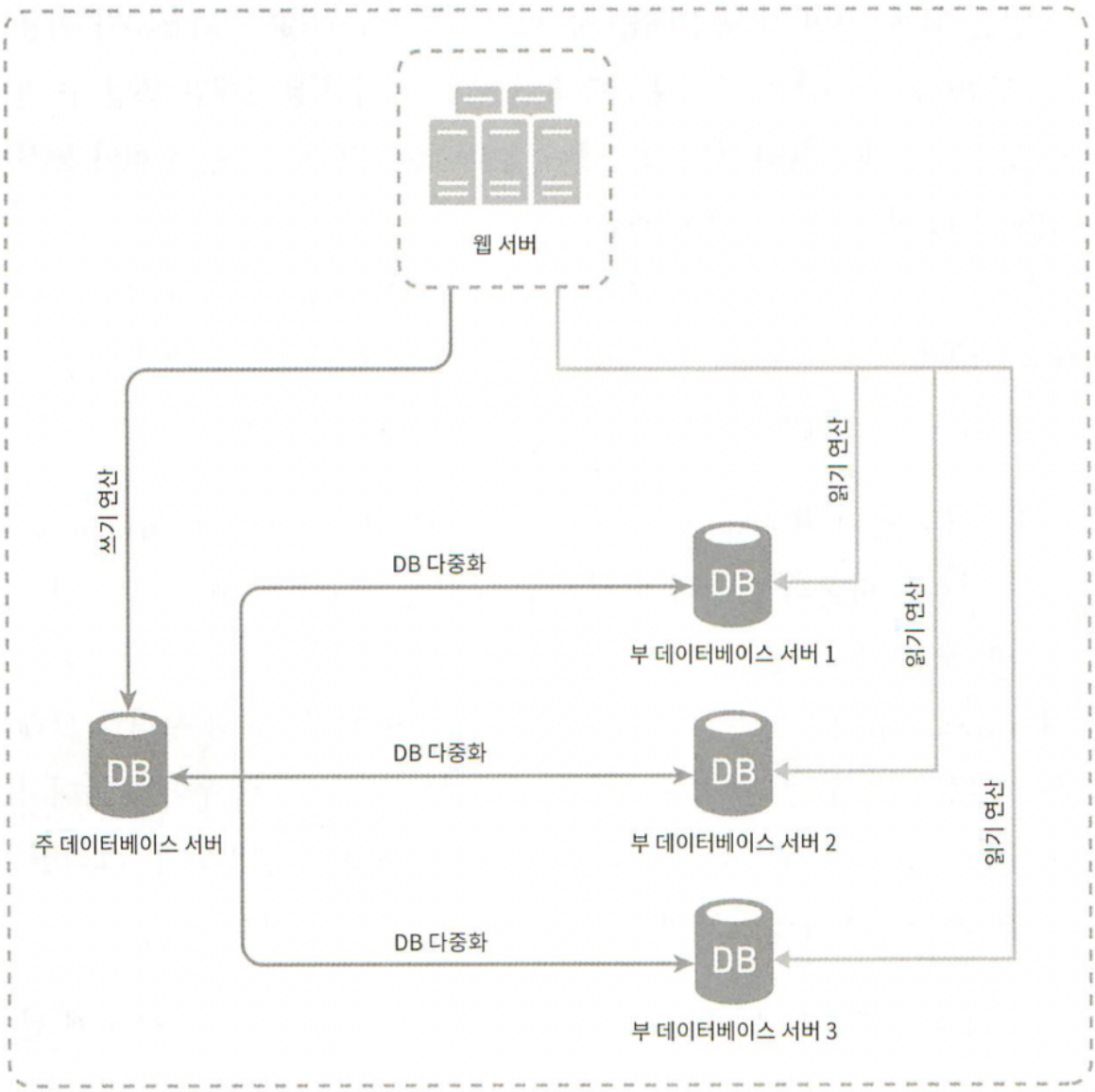
데이터베이스 다중화

많은 데이터베이스 관리 시스템이 다중화를 지원한다. 서버 사이의 주-부 관계를 설정하고

주 서버에는 데이터 원본을, 부 서버에는 사본을 저장한다.

쓰기 연산 (insert, update, delete): 주 서버만 지원

읽기 연산 (select): 주 서버, 부 서버 모두 가능 ⇒ 대부분의 어플리케이션은 읽기 연산 비중이 훨씬 높다. 그래서 부 데이터베이스 수 > 주 데이터베이스 수



다중화를 했을 때의 장점

- 더 나은 성능 : 병렬로 처리될 수 있는 질의 수가 늘어나 성능이 좋아짐
- 안정성 : 여러 장소에 다중화하여 데이터베이스 서버 일부가 파괴되어도 데이터는 보존될 것
- 가용성 : 데이터를 여러 지역에 복제해 한 곳에서 장애가 발생해도 다른 서버에 있는 데이터를 가져와 계속 서비스 할 수 있다.

Q. 데이터베이스 서버 가운데 하나가 다운되면 무슨 일이 벌어지는가?

- 부 서버가 한 대 뿐인데 부 서버가 다운된 경우

  읽기 연산은 한시적으로 모두 주 데이터베이스로 전달될 것이며 새로운 부 데이터베이스 서버가 장애 서버를 대체할 것이다.

- 부 서버가 여러대이고 부 서버가 다운된 경우

  나머지 부 데이터베이스 서버들로 읽기 연산이 분산될 것이고 새로운 부 데이터베이스가 장애 서버를 대체할 것 이다.

- 주 데이터베이스 서버가 다운된 경우

  한 대의 부 데이터베이스만 있는 경우 해당 부 데이터베이스 서버가 새로운 주 서버의 역할을 할 것이다. 그리고 새로운 주 서버상에서 수행할 것이다.

  프로덕션 환경에서 벌어지는 일은 이것보다 더욱 복잡하다. 부 서버에 보관된 데이터가 최신 상태가 아닐 수 있기 때문이다.

  Q. 의문… ⇒ 주에서는 쓰고, 부에서는 읽어, 근데 주에는 쓰여있는데 부에 동기화가 안되서 없는 값으로 인식하면 위험한 것 아닌가..? ⇒ 둘이 동기화를 어떻게 시켜주지..?? 항상 같은 상태를 유지해야하는 것 아닌가? 근데 또 쓰기할때마다 갱신시켜주면 이것도 비효율적일 것 같은데..

** 프로덕션(production) 환경 : 실제 서비스를 위한 운영 환경

이제까지 웹 서버 계층과 데이터 계층에 대한 내용이였습니다. 다음으로는 응답시간을 개선해 볼 순서입니다. 응답 시간은 캐시를 붙이고 정적 콘텐츠를 콘텐츠 전송 네트워크 (CDN)으로 옮기면 개선할 수 있다.

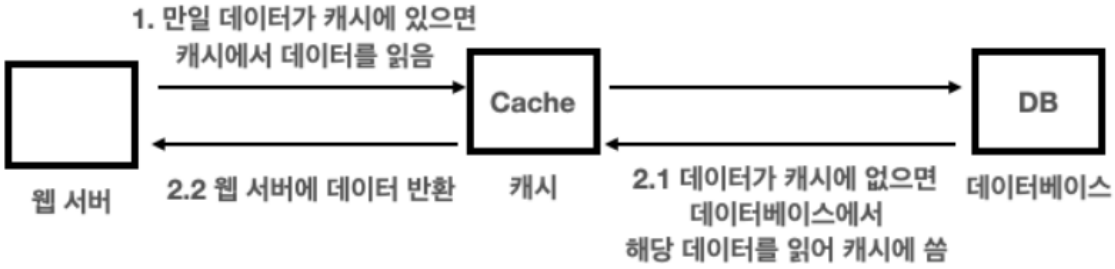
캐시

캐시는 값비싼 연산 결과 또는 자주 참조되는 데이터를 **메모리 안에** 두고 다음 요청을 더 빨리 처리할 수 있게 하는 저장소입니다.

캐시가 없다면 웹 페이지를 새로고침할 때마다 표시할 데이터를 가져오기 위해 한 번 이상의 데이터베이스 호출이 발생하게 된다.

캐시 계층

데이터가 잠시 보관되는 곳으로 데이터베이스보다 훨씬 빠르다. 성능도 빨라지며 부하를 줄일 수 있고 캐시 계층의 규모를 독립적으로 확장시키는 것도 가능해진다.



요청을 받은 웹 서버는 캐시에 응답이 저장되어 있는지 확인한다.

읽기 주도형 캐시 전략

- 저장되어 있다면 캐시에서 값을 반환한다
- 만약 없다면 질의를 통해 DB에 접근한 후 캐시에 저장한 뒤 클라이언트에 반환한다.

캐시를 사용할 때 유의할 점

- 캐시는 데이터 갱신이 거의 일어나지 않고 참조가 빈번하게 일어나는 경우 쓰기 유용하다.
- 캐시는 데이터를 휘발성 메모리에 두므로 영속적으로 보관할 데이터를 캐시에 두는 것은 바람직하지 않습니다. 중요 데이터는 지속적 저장소에 저장해야 합니다. **(캐시는 당연히 휘발성이고 임시적으로 자주 쓰이는 데이터를 모아두는 곳 아닌가? 이게 주 데이터베이스보다는 데이터베이스의 사본 느낌이지 않나? 근데 이런 말이 왜 언급되는 거지??)**
- 만료 시간(expored time)을 지정해두는 것이 좋습니다. 만료 정책이 없다면 데이터는 캐시에 계속 남게됩니다. 만료 기간은 너무 짧으면 데이터베이스를 너무 자주 읽어야해서 곤란하고 너무 길면 원본과 차이가 날 수 있어서 곤란하다.
- 일관성 이슈 : 데이터 저장소의 원본과 케시 내의 사본이 같은지에 대한 여부
    - 저장소의 원본을 갱신하는 연산과 캐시를 갱신하는 연산이 단일 트랜잭션으로 처리되지 않는 경우 이 일관성은 깨질 수 있습니다.
- 캐시서버를 한 대만 두는 경우 SPOF (Single Point Of Failure, 어떤 지점에서 오류가 생겼을 때 전체 시스템의 동작을 중단시켜버릴 수 있는 경우)이 발생할 수 있다. 즉 이를 피하려면 여러 지역에 걸쳐 캐시 서버를 분산시켜야 함

- 캐시 메모리를 과할당(overprovision)하자. **(Q. 과할당,,?? 그냥 무작정 크게만 잡으면 되는가? 캐시면 직접적으로 메모리에 할당되는 건데 이는 메모리 낭비아닌가?)**

  캐시 메모리가 너무 작으면 액세스 패턴에 따라 데이터가 너무 자주 캐시에서 밀려나버리고 성능이 저하된다.

- 데이터 방출 정책(캐시가 꽉 찼고 추가로 데이터를 넣어야하는 경우 기전 데이터를 내보내는 방식)을 정하자.
  - LRU : 마지막으로 사용된 시점이 가장 오래된 데이터를 내보내는 정책
  - LFU : 사용된 빈도가 가장 낮은 데이터를 내보내는 정책
  - FIFO : 가장 먼저 캐시에 들어온 데이터를 가장 먼저 내보내는 정책

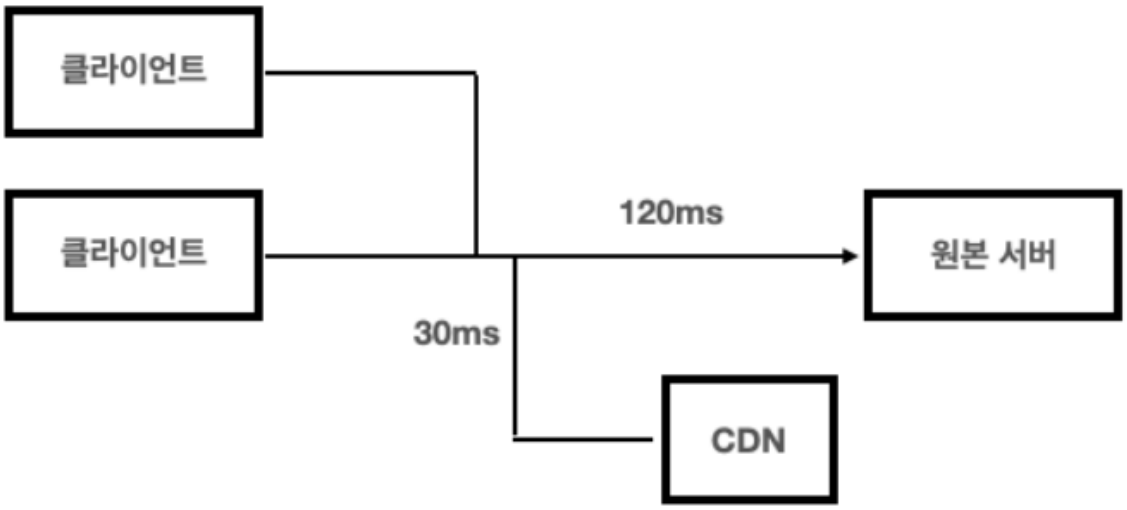
콘텐츠 전송 네트워크(CDN, content delivery network)

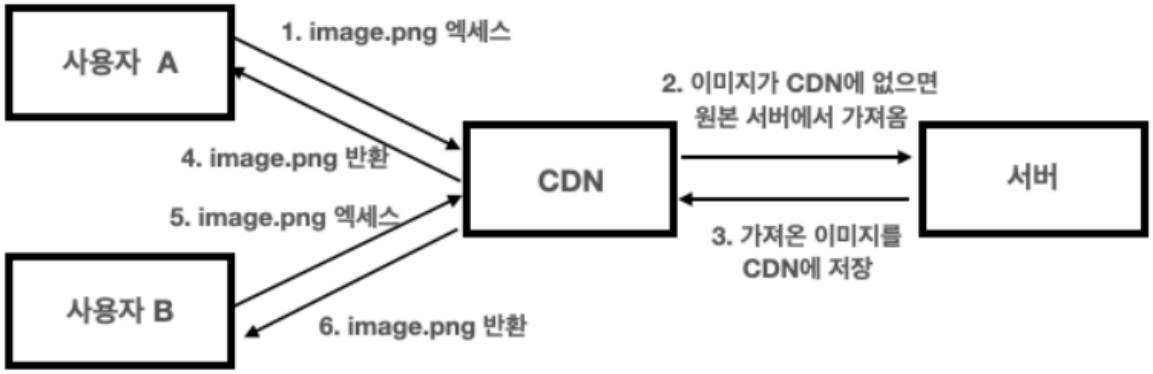
CDN은 정적인 콘텐츠(이미지, 비디오, CSS, JS)를 전송하는 데 쓰이는 지리적으로 분산된 서버의 네트워크이다.

▼ 참고 (동적 콘텐츠 캐싱은 어떻게?)

요청경로, 질의 문자열, 쿠키, 요청헤더 등의 정보에 기반하여 HTML 페이지를 캐시하는 것

사용자에게 가장 가까운 CDN 서버가 정적 콘텐츠를 전달하게 됩니다. 사용자가 CDN 서버로부터 멀수록 웹사이트는 천천히 로드된다.





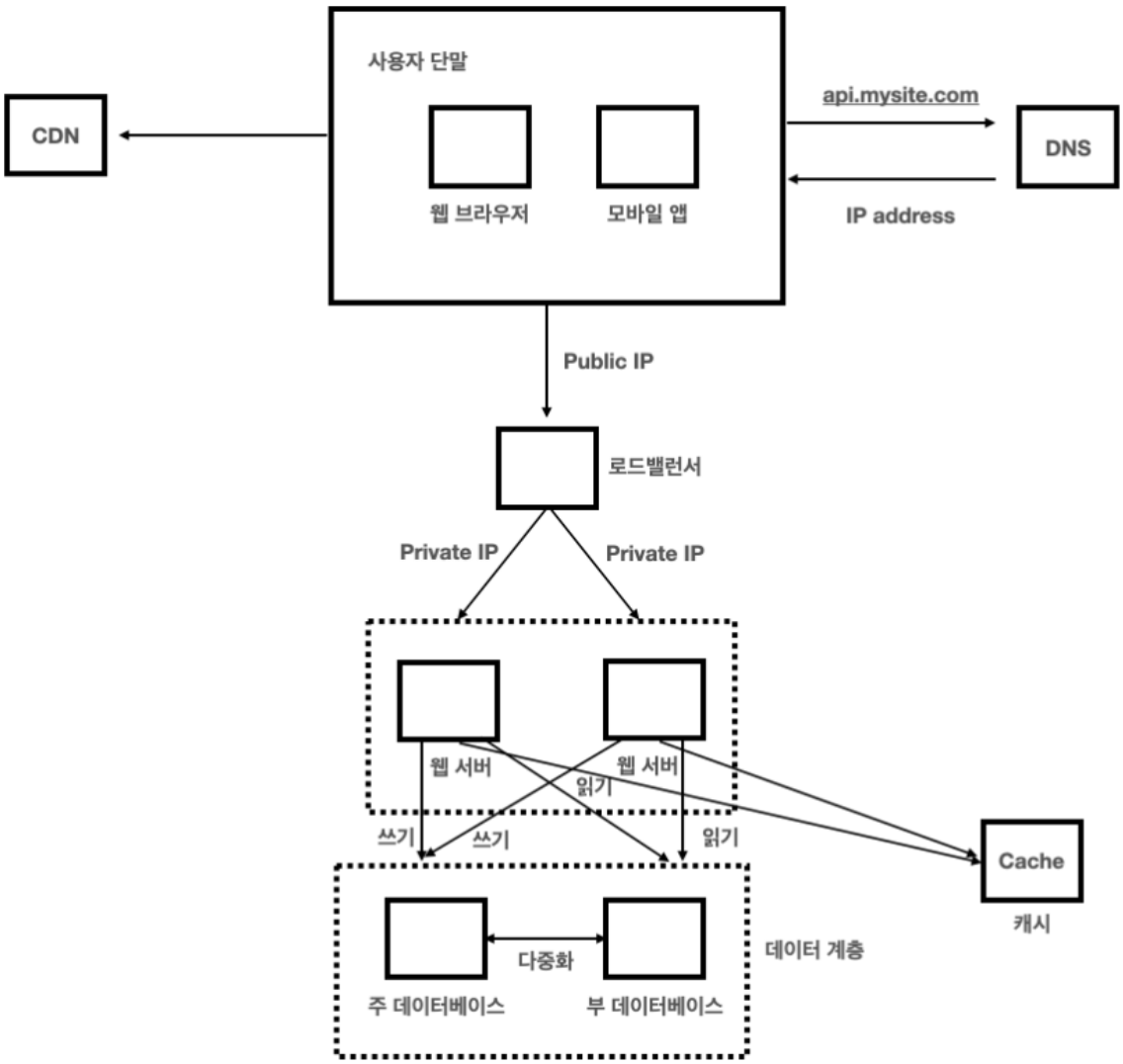
CDN 동작과정

1. 사용자 A가 이미지 URL을 이용해 image.png에 접근
2. CDN에 해당 이미지가 없으면 원본 서버에서 가져온다.
3. 원본 서버가 파일을 CDN에 반환
4. CDN 서버는 파일을 캐시하고 사용자 A에게 반환 (파일은 TTL(time to live) 만큼 캐시된다)
5. 사용자 B가 같은 사진을 요청한다.

6. TTL이 만료되지 않은 경우 해당 요청을 CDN에 캐시되어 있는 정보로 반환한다.

CDN 사용 시 고려할 점

- 비용
- 적절한 만료 기한 설정
- CDN 장애에 대한 대처 방안 (모든 요청이 무조건 캐시로 바로 접근하는 것이 아니라 원본 데이터로도 접근할 수 있게 처리)
- 콘텐츠 무효화



CDN과 캐시가 추가된 설계

1. 정적 콘텐츠는 더 이상 웹 서버를 통해 서비스하지 않으며, CDN을 통해 제공하여 더 나은 성능 보장
2. 캐시가 데이터베이스 부하를 줄여줌

무상태(Stateless) 웹 계층

이 부분은 **웹 계층을 수평적을 확장하는 방법**에 대한 고민이다. 이를 위해서 상태 정보를 웹 계층에서 제거해야합니다. ⇒ 바람직한 전략은 상태 정보를 관계형 데이터베이스나 NoSQL 같은 **지속성 저장소에 보관하고, 필요할 때 가져오도록 하는 것**이다. 이런 웹 계층을 무상태 웹 계층이라 부른다.

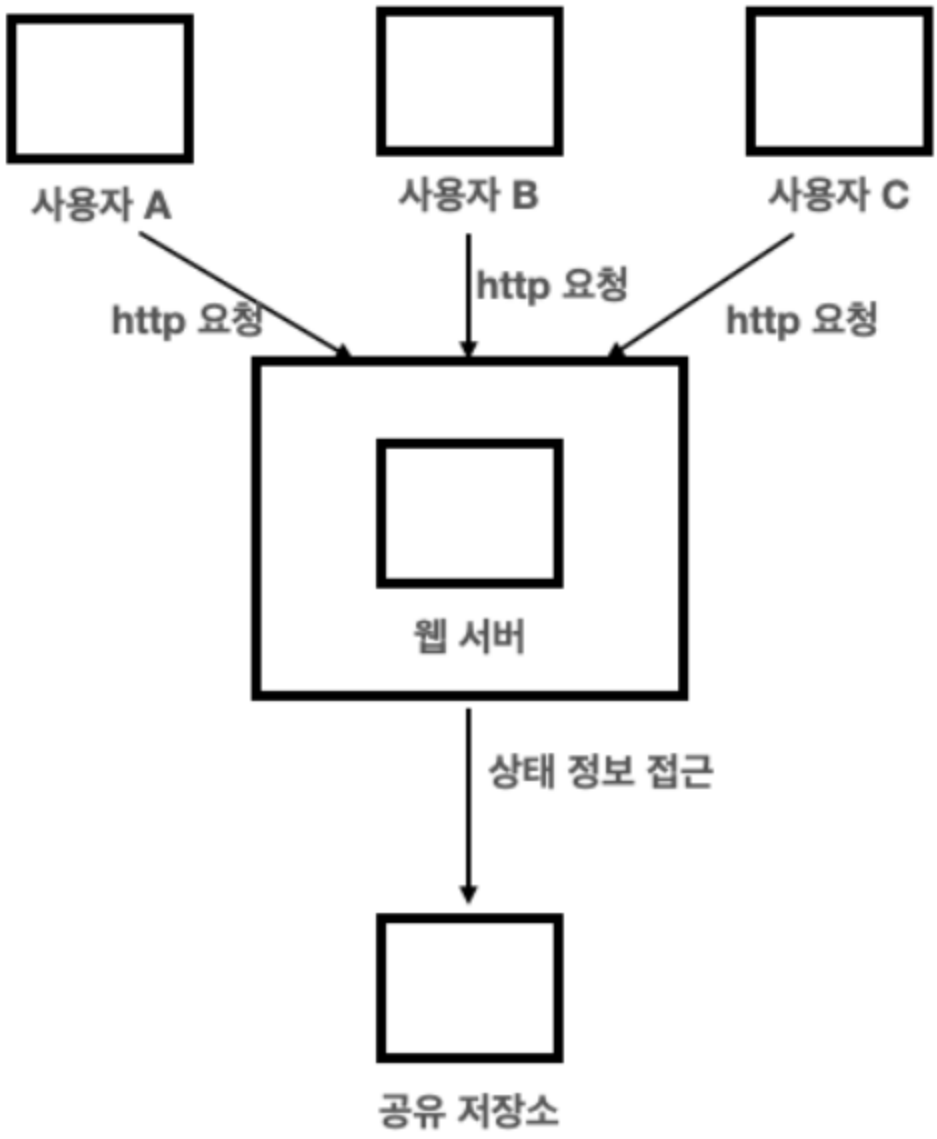
상태 정보 의존적인 아키텍쳐

상태 정보를 보관하는 서버는 클라이언트 정보, 즉 상태를 유지하여 요청들 사이에 공유되도록 한다.

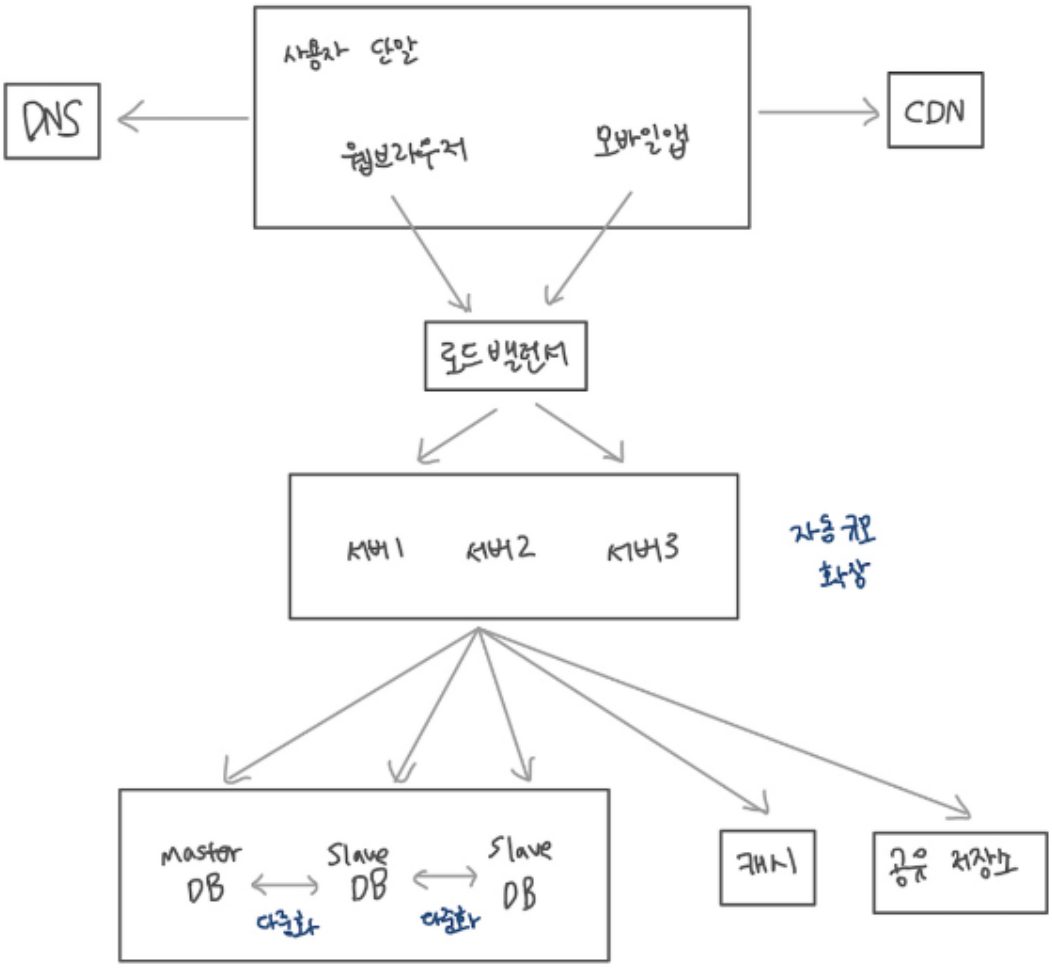
문제점은 같은 클라이언트로부터의 요청은 항상 같은 서버로 전송되어야 하는 것

⇒ 대부분의 로드밸런서는 이를 지원하기 위해 고정 세션이라는 기능을 제공하는데 이는 큰 부담을 주게된다.

무상태 아키텍쳐



웹 서버는 상태 정보가 필요한 경우 공유 저장소로부터 데이터를 가져옵니다. 상태 정보는 웹 서버로부터 물리적으로 분리되어 있다. 이 구조는 단순하고, 안정적이며느 규모 확장이 쉽다.



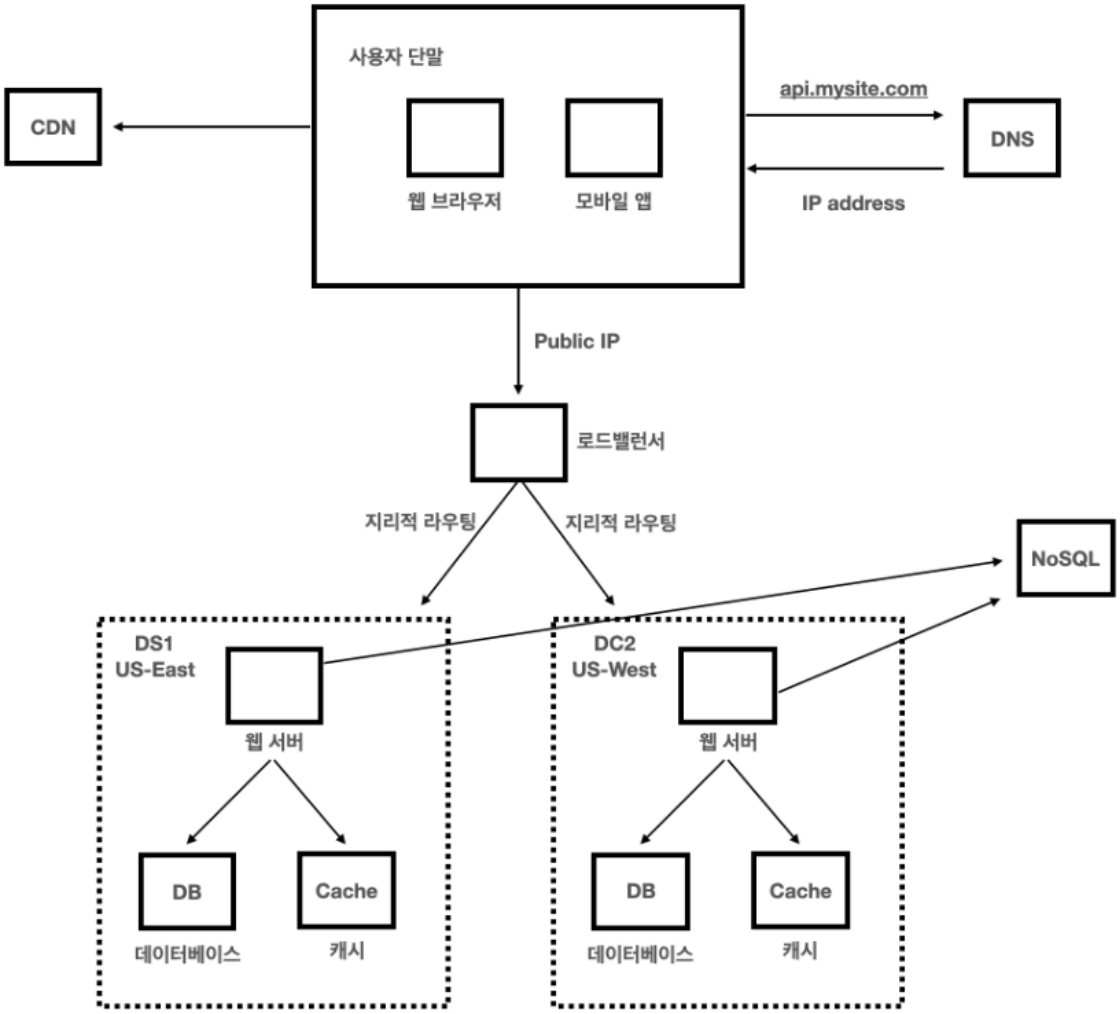
Q. page 19쪽 그림 1-14(위 그림)에서 데이터베이스로 실선과 접선으로 구분.. 이게 무슨 의미일까..?

위 공유 저장소는 관계형 데이터베이스일 수도 있고, Memcahed/Redis 같은 캐시 시스템일 수도 있고 NoSQL일 수도 있다.

데이터센터

장애가 없는 경우 **지리적 라우팅을 통해 가장 가까운 데이터 센터로 안내**됩니다.

지리적 라우팅에서의 geoDNS는 사용자의 위치에 따라 도메인 이름을 어떤 IP 주소로 변환할지 결정할 수 있도록 해주는 DNS 서비스입니다.

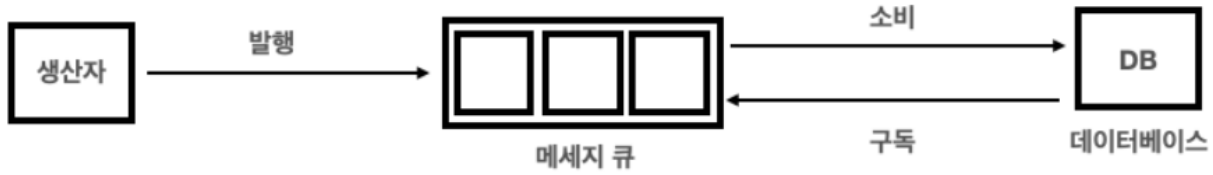


만약 데이터 센터 중 한곳에 심각한 장애가 발생하면 모든 트래픽은 장애가 없는 데이터 센터로 전송되고 **다중 데이터센터 아키텍쳐를 만들려면 몇가지 난제를 해결**해야 한다.

- 트래픽 우회 : 올바른 데이터 센터로 트래픽을 보내는 효과적인 방법을 찾아야 한다. GeoDNS는 사용자에게서 가장 가까운 데이터센터로 트래픽을 보낼 수 있도록 해준다.
- 데이터 동기화 : 데이터 센터마다 별도의 데이터베이스를 사용하고 있다면 장애 발생시 찾는 데이터가 없을 수도 있다. 이런 상황을 막는 보편적 전략은 **데이터를 여러 데이터 센터에 걸쳐 다중화하는 것**이다.
- 테스트와 배포 : 여러 데이터 센터를 사용하도록 구성되어 있다면 서비스를 여러 위치에서 테스트해보는 것이 중요하다. 자동화된 배포 도구는 모든 데이터 센터에 동일한 서비스가 설치되는데 중요한 역할을 한다.

메세지 큐

- 메세지의 무손실 보장 : 메세지 큐에 보관된 메세지는 소비자가 꺼낼때까지 안전하게 보관
- 비동기 통신(순서없이 그냥 무작위로)을 지원하는 컴포넌트
- 메세지의 버퍼 역할
- 서버간의 결합이 느슨해진다 (낮은 결합도)
- 생산자와 소비자는 독립적으로 확장될 수 있다. (서비스 확장성에 좋음)



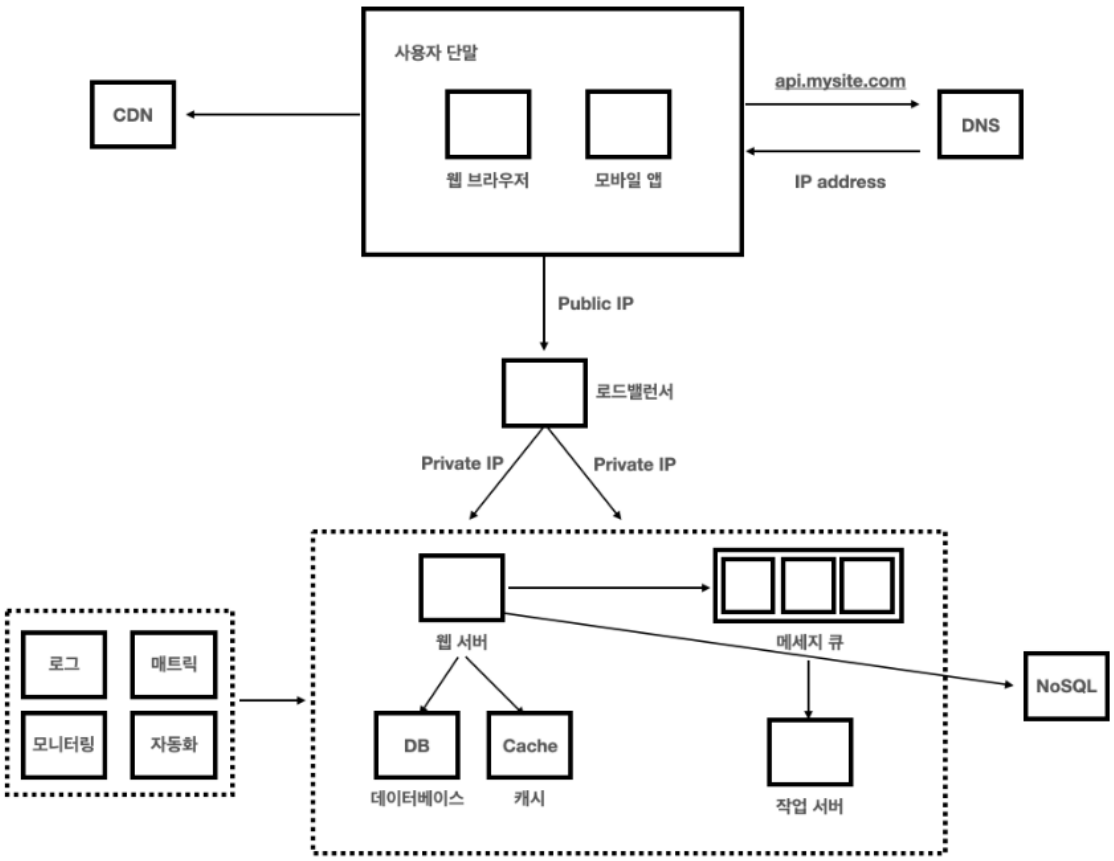
생산자(producer/publisher) : 입력 서비스가 메세지를 만들어 메세지 큐에 발행(publish)

소비자(consumer/subscriber) : 메세지를 받아 그에 맞는 동작을 수행함

큐의 크기가 커지면 많은 작업 프로세스를 추가해야 처리 시간을 줄일 수 있다. 큐가 항상 비어 있는 상태라면 작업 프로세스의 수를 줄일 수 있다.

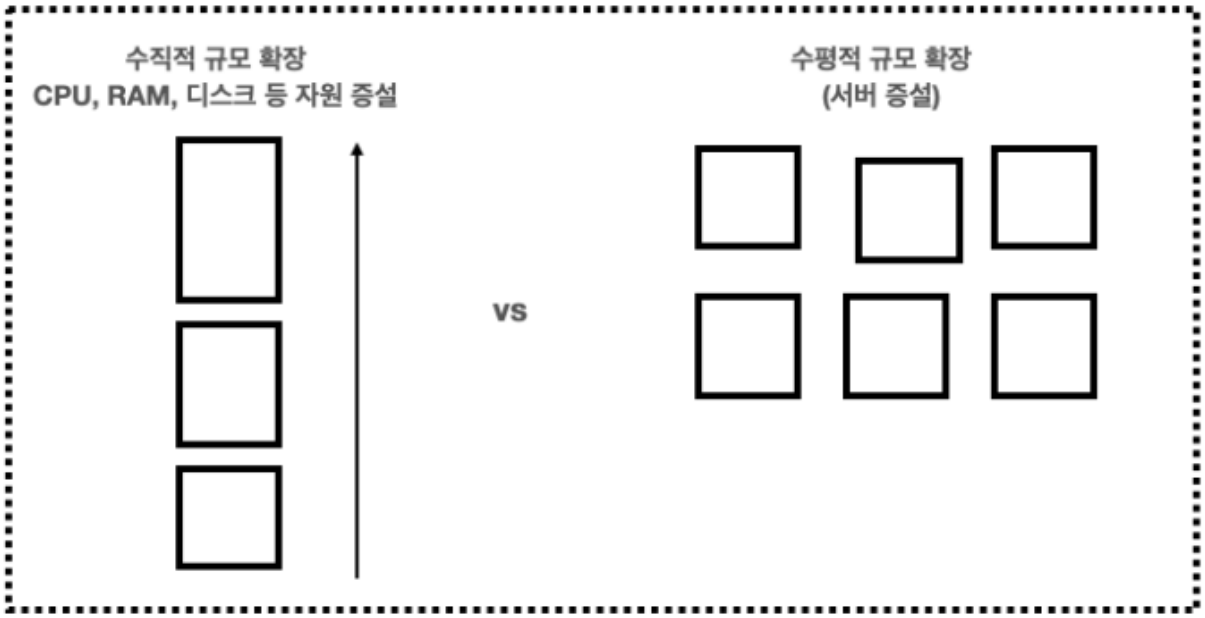
로그, 메트릭 그리고 자동화

- 로그 : 에러 모니터링 하는 것. 로그를 단일 서비스로 모아주는 도구를 사용하면 더 효율적
- 메트릭 : 유용한 정보, 시스템 현 상태를 파악하기 위해
- 자동화 : 생산성과 안정성을 높이기 위해



- 메세지 큐는 각 컴포넌트가 보다 느슨히 결합될 수 있도록 한다.
- 로그, 모니터링, 메트릭, 자동화 등을 위한 장치가 추가되었다.

데이터베이스의 규모 확장



수직적 확장 (Scale Up)

기존 서버에 더 많은, 고 성능의 자원을 증설하는 방법

- CPU, RAM 등을 무한 증설할 주 없음. 사용자가 계속 늘어나면 한 대 결국 감당하기 어렵게 될 것
- SPOF (Single Point Of Failure)로 인한 위험성이 크다.
- 비용이 든다. 고성능 서버로 갈수록 가격이 올라가게 된다.

수평적 확장 (샤딩)

더 많은 서버를 추가하면서 성능을 추가시키는 것

샤딩은 대규모 데이터베이스를 샤드(Shard)라고 부르는 작은 단위로 분할하는 기술을 말한다. 샤드에 보관되는 **데이터사이에는 중복이 없**습니다.

샤딩 전략을 구현할 때 중요한 것은 샤딩키이다. 샤딩 키에 따라서 한 곳으로만 부하가 집중될 수 있고 여러 곳으로 적절히 잘 분산될 수 있다.

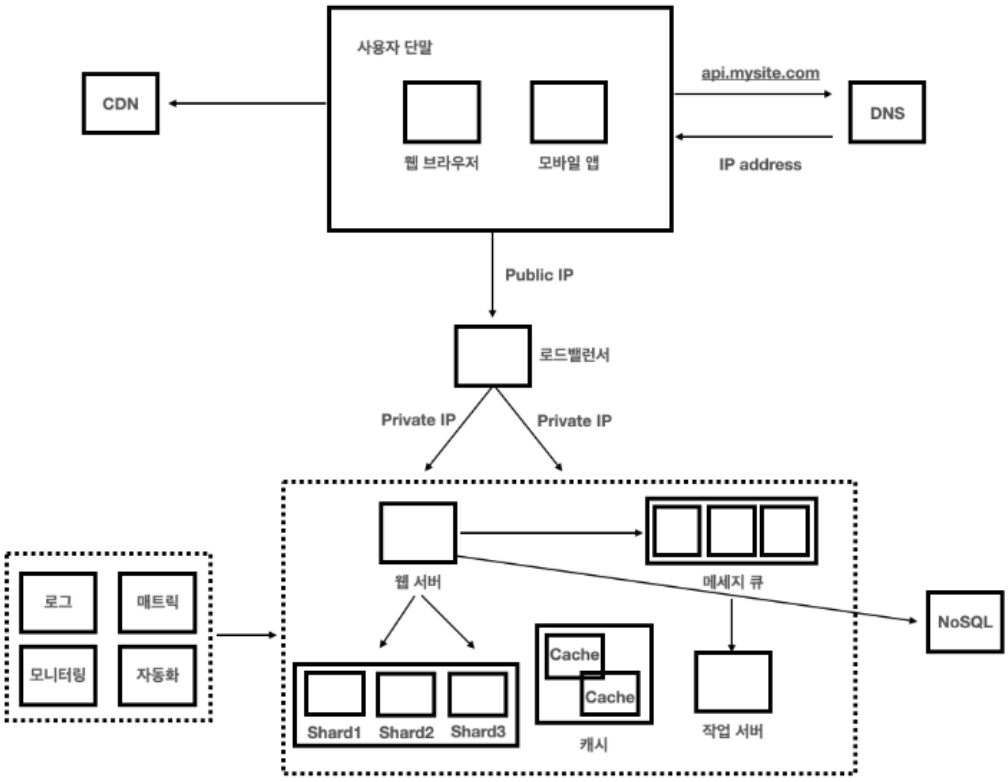
샤딩을 위해서 고려해야 할 조건

- 데이터의 재 샤딩 (reshading)
    - 데이터가 너무 많아져서 하나의 샤드로는 더 이상 감당하기 어려울 때

    - 샤드 간 데이터 분포가 균동하지 못하면 어떤 샤드에 할당된 공간 소모가 다른 샤드에 비해 빨리 진행될 때
- 유명인사 문제 (celebrity)

  특정 샤드에 질ㄹ의가 집중되어 서버에 과부하가 걸리는 문제
- 조인과 비정규화 (join and de-normalization)

  하나의 데이터베이스를 여러 샤드 서버로 쪼개고 나면, 여러 샤드에 걸친 데이터를 조인하기 힘들어진다.



정리 (시스템 규묘 확장을 위한 기법)

- 웹 계층은 무상태 계층으로
- 모든 계층에 다중화 도입
- 가능한 많은 데이터를 캐시할 것

- 여러 데이터 센터를 지원할 것
- 정적 콘텐츠는 CDN을 통해 서비스할 것
- 데이터 계층은 샤딩을 통해 그 규모를 확장할 것
- 각 계층은 독립적 서비스로 분환할 것
- 시스템을 지속적으로 모니터링하고, 자동화 도구들을 활용할 것