# [3장] 시스템 설계 면접 공략법

가상 면접 사례로 배우는 대규모 시스템 설계 기초

이민석 / unchaptered

https://inblog.ai/monthly-cs
https://github.com/monthly-cs/2024-03-system-design-interview-1

# 시스템 설계 면접(면접자)

1. 설계 기술의 시연
2. 의사결정에 대한 **방어 능력** | 의사결정에 대한 당위성을 설득
3. 피드백에 대한 **추가 조치 능력**

# 시스템 설계 면접(면접관)

1. **기술**적 측면에 대한 평가
2. **협력**에 적합한 사람인지 평가
3. **압박**에 대한 내성 평가
4. **모호한 질문**을 해결할 수 있는 능력을 평가
5. 설계의 순수성(Purity)과 타협적 결정(Tradeoff)의 불균형 평가

# 효과적 면접을 위한 4단계 접근법

1단계 | **요구사항 분석** | 문제 이해 및 설계 범위 확정
- 1단계 - 1 | 기능 요구사항 탐색
- 1단계 - 2 | 개략적인 규모 추정을 위한 수치 추정 탐색
- 1단계 - 3 | 추가적인 가용성, 재해복구에 대한 요구사항
- 1단계 - 4 | 정리

2단계 | **초기 설계** | 개략적인 설계안 제시 및 동의 구하기

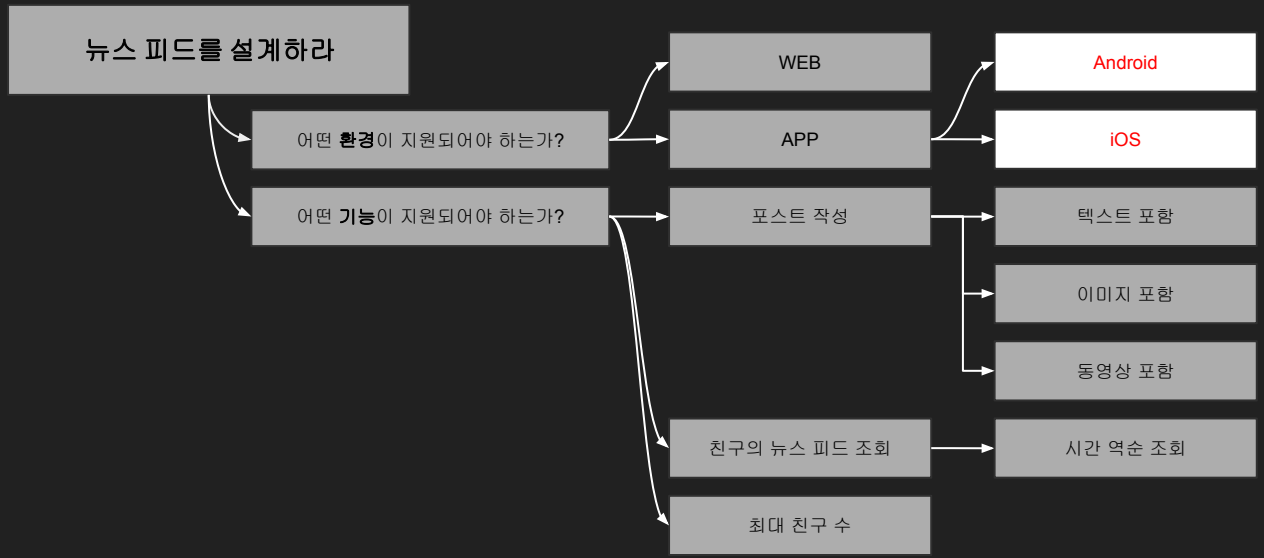
3단계 | **상세 설계**

4단계 | **마무리**
- 4단계 - 1 | 요약
- 4단계 - 2 | 구간 별 장애 발생 시의 상황
- 4단계 - 3 | 운영 측면
- 4단계 - 4 | 백만 사용자가 된다면 일어날 법한 일들

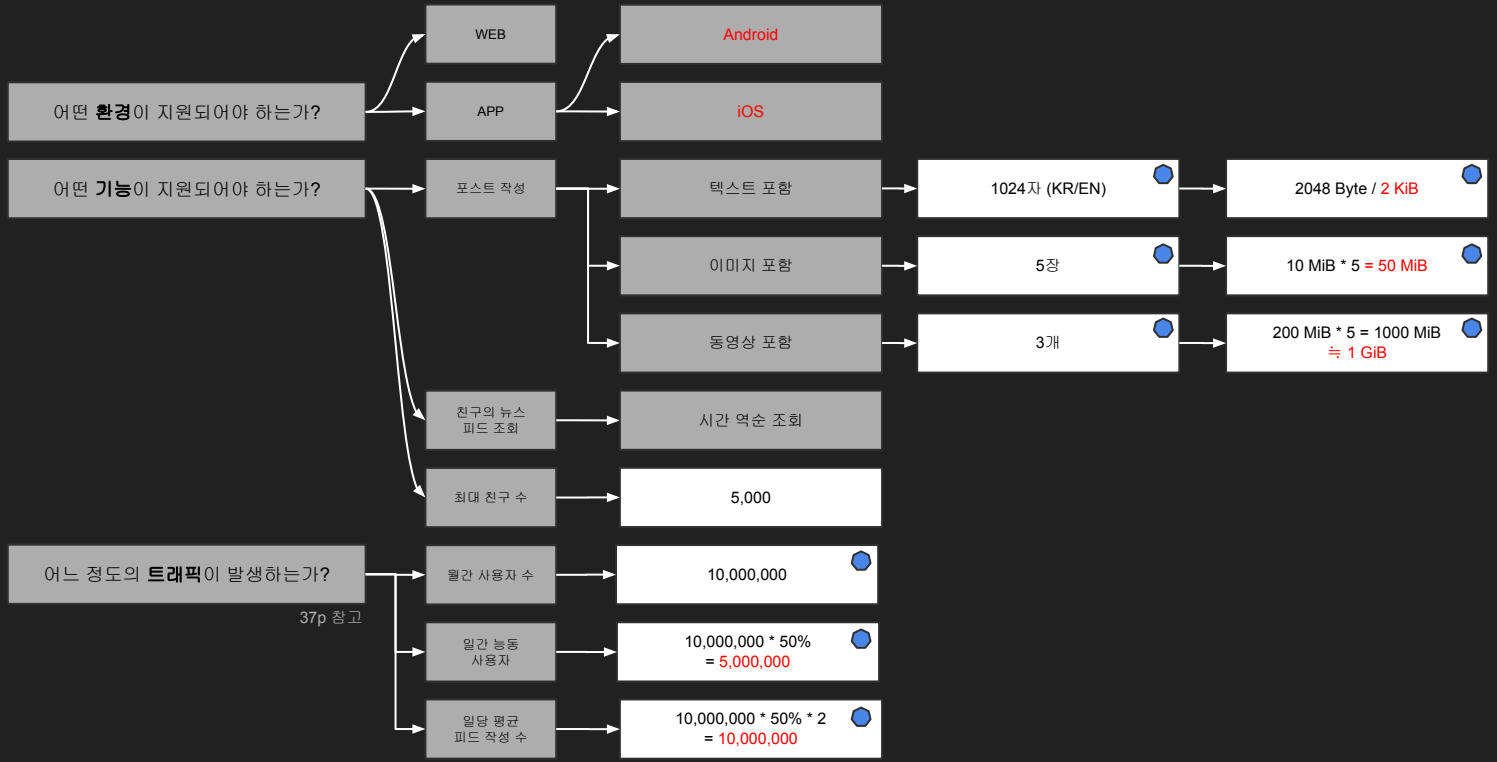
# 1단계 | 요구사항 분석 | 문제 이해 및 설계 범위 확정

1. 시스템 설계를 위한 **기능 요구사항** 탐색
2. 개략적인 규모 추정을 위한 **수치 추정** 탐색
3. 추가적인 **가용성, 재해복구**에 대한 요구사항
4. **정리**

# 1단계 - 1 | 기능 요구사항 탐색



# 1단계 - 2 | 개략적인 규모 추정을 위한 수치 추정 탐색



# 1단계 - 3 | 개략적인 규모 추정을 위한 수치 추정 탐색


QPS = 일간 능동사용자 * 피드 작성 수 / 24시간 / 3600초
= 5,000,000 * 2 / 24/ 3600
≒ 115
QPS, 텍스트 크기 = 115 * 2KiB ≒ **200KiB**
QPS, 이미지 크기 = 115 * 10MiB ≒ **1GiB**
QPS, 동영상 크기 = 115 * 200MiB = 23,000 MiB ≒ **23 GiB**

일간 텍스트 생성량(최댓값) = 일간 능동사용자 * 피드 작성 수 * 2KiB
= 5,000,000 * 2 * 2KiB
= 20,000,000 KiB
≒ 20,000 MiB
≒ 20 GiB
월간 텍스트 생성량(최댓값) = 일간 텍스트 생성량 * 30
≒ 600 GiB
연간 텍스트 생성량(최댓값) = 일간 텍스트 생성량 * 365
= 7300 GiB
≒ 7 TiB
일간 이미지 생성량(최댓값) = 5,000,000 * 5 * 10MiB
= 250,000,000 MiB
≒ 0.25 PiB
월간 이미지 생성량(최댓값) = 일간 이미지 생성량 * 30
≒ 7.5 PiB
연간 이미지 생성량(최댓값) = 일간 이미지 생성량 * 365
≒ 91.25 PiB

일간 동영상 생성량(최댓값) = 5,000,000 * 3 * 200 MiB
= 3,000,000,000 MiB
≒ 3.0 PiB
월간 동영상 생성량(최댓값) = 일간 이미지 생성량 * 30
≒ 90.0 PiB
연간 동영상 생성량(최댓값) = 일간 이미지 생성량 * 365
≒ 1095 PiB

# 1단계 - 3 | 추가적인 가용성, 재해복구에 대한 요구사항

**[가용성]**

최초에는 Multi-AZ로 배포를 하지만 추후에는 Multi-Region으로 서비스 이전이 필요할 수 있음

**[재해복구]**

시스템은 분단위(minutes) 내로 복구가 되어야함

# 1단계 - 4 | 정리

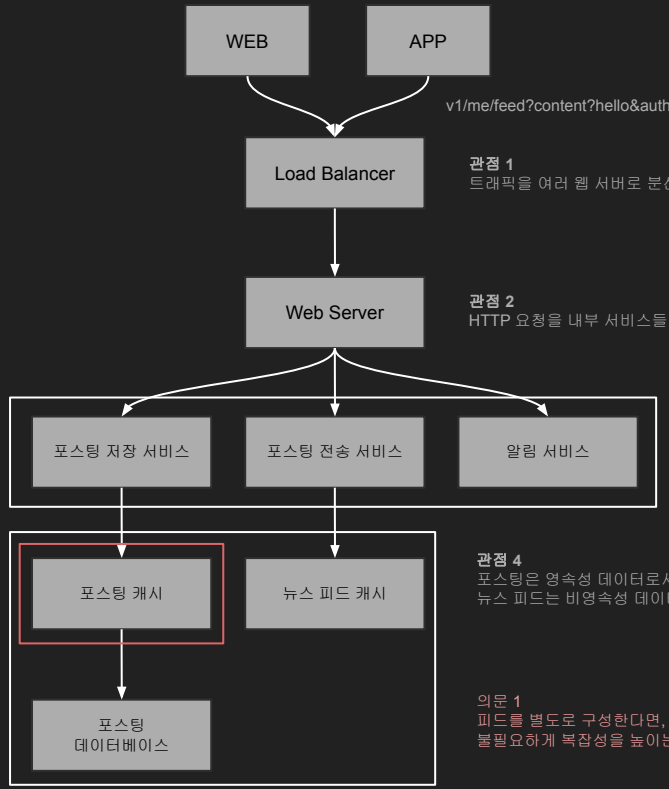
**[트래픽]**
예측 불가능한 시간대에 트래픽이 몰릴 가능성이 존재함
1개의 쓰기 작업에 대해 최대 5,000배의 조회작업이 발생할 수 있음
글, 이미지, 동영상이 각각 7 TiB / 90 PiB / 1000 PiB씩 쓰일 수 있음

**[미디어 파일]**
대량의 미디어 파일 업로드/다운로드로 인한 병목이 발생할 수 있음
업로드 후, 피드 생성에 실패한 경우 정크 동영상이 남을 수 있음

**[가용성]**
최초에는 Multi-AZ로 배포를 하지만 추후에는 Multi-Region으로 서비스 이전이 필요할 수 있음

**[재해복구]**
시스템은 분단위(minutes) 내로 복구가 되어야함

# 2단계 | 초기 설계 | 개략적인 설계안 제시 및 동의 구하기




**관점 1**
트래픽을 여러 웹 서버로 분산

**관점 2**
HTTP 요청을 내부 서비스들로 중계하는 역할을 진행

**관점 3**
가용성을 극대화하기 위해서 **핵심 기능 별로 서비스를 구성**

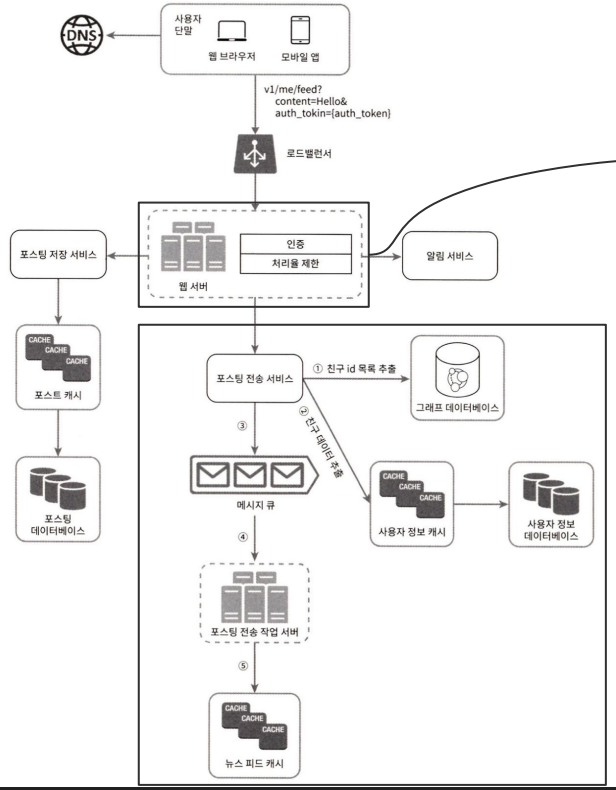
**관점 4**
포스팅은 영속성 데이터로서 캐시 + 데이터베이스에 저장
뉴스 피드는 비영속성 데이터로서 캐시에만 저장

의문 1
피드를 별도로 구성한다면, 포스팅 캐시의 역할은 무엇인가?
불필요하게 복잡성을 높이는 부분이 아닐까?

AS-IS (포스팅 캐시가 없을 경우)
- 피드 최초 구성 시, "시간 역순"을 충족하는 대상을 데이터베이스에서 모두 SELECT해야 한다

TO-BE (포스팅 캐시가 있을 경우)
- 피드 최초 구성 시, 포스팅 캐시에 적재된 캐시들이 "시간 역순"을 충족하게만 저장을 해두면 로직이 간결해진다.

# 3단계 | 상세 설계




**관점 1**
웹 서버에서 처리율 제한 기능을 수행

> **처리율 제한?**
> 서버가 동시에 처리할 수 있는 **동시 처리량** 제한

**관점 2**
포스팅 전송 서비스는 **Fanout Pattern**을 사용하여 아래의 목표를 달성 가능
1. 실패한 작업의 재실행 보장(N회 한정)
2. 작업의 순서 보장(FIFO Queue인 경우)

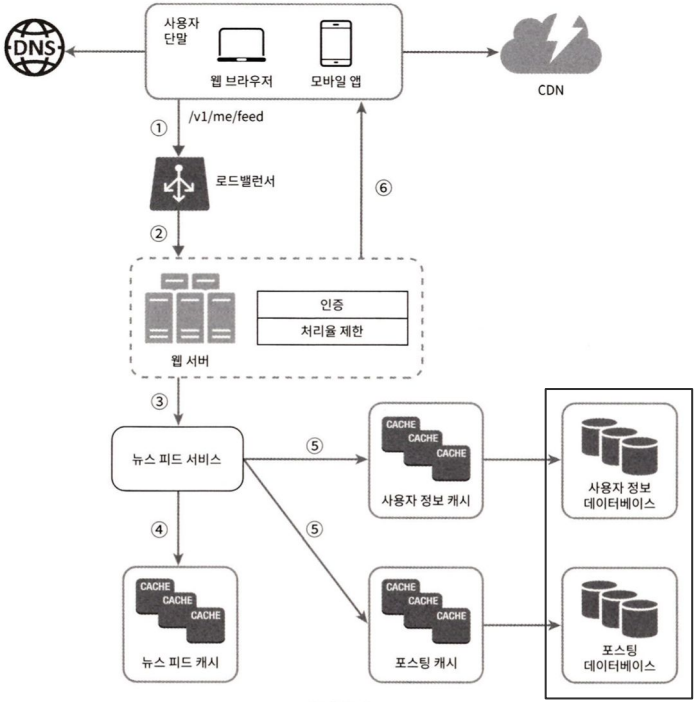
읽기에 유리한 Fanout-on-write와 쓰기에 유리한 Fanout-on-demand 방식이 있다.
여기서는 일반 사용자는 Fanout-on-write를 사용하고 친구가 많은 사용자는 Fanout-on-demand 방식을 사용해서 시스템 부하를 줄일 것입니다

> **왜 Fanout-on-write를 선택하는가?**
> 요구사항 분석 단계에서 발행된 피드는 최대 5,000명의 친구들에게 일괄적으로 보여지게 됨을 알게 되었습니다. 따라서 대량의 읽기 작업이 발생할 수 있고 **시스템 부하**를 줄이기 위해서 읽기 캐시를 쓰는 것이 합리적입니다.

> **왜 Fanout-on-demand를 선택하는가?**
> 친구가 5,000명에 가까운 사용자의 활동을 모두 써버리면 Hotkey 이슈가 발생합니다. Fanout-on-demand로 이를 방지합니다.

https://velog.io/@haron/%EA%B0%80%EC%83%81-%EB%A9%B4%EC%A0%91-%EC%82%AC%EB%A1%80%EB%A1%9C-%EB%B0%B0%EC%9A%B0%EB%8A%94-%EB%8C%80%EA%B7%9C%EB%AA%A8-%EC%8B%9C%EC%8A%A4%ED%85%9C-%EC%84%A4%EA%B3%84-%EA%B8%B0%EC%B4%88-11%EC%9E%A5-%EB%89%B4%EC%8A%A4-%ED%94%BC%EB%93%9C-%EC%8B%9C%EC%8A%A4%ED%85%9C-%EC%84%A4%EA%B3%84-i6re3cjt

# 3단계 | 상세 설계




**관점 1**
이미지, 동영상은 CDN을 사용해서 공급

**개인적 관점 1**
이미지와 동영상은 서로 다른 CDN을 사용하는 것이 좋음
- 일반적으로 동영상의 캐시 적중률이 낮기 때문에, 서로 **다른 캐시 정책(TTL)을 적용**할 필요성이 있습니다.

**의문 1**
왜 사용자 데이터베이스와 포스팅 데이터베이스를 분리하였는가?

↓

**AS-IS (분리하지 않을 경우)**
- 대량의 쓰기, 조회 작업에서 DB 부하가 걸릴 시, 데이터베이스의 장애로 인해서 MSA형태로 분할된 서비스 전체가 사용 불가능해짐. → 가용성 악화

**TO-BE (분리한 경우)**
- 신규 친구 정보가 되지 않더라도, 기존의 포스팅과 뉴스 피드가 조회 가능함. → 가용성 강화
- 대신 최저 비용은 더 많이 나올 것

https://velog.io/@haron/%EA%B0%80%EC%83%81-%EB%A9%B4%EC%A0%91-%EC%82%AC%EB%A1%80%EB%A1%9C-%EB%B0%B0%EC%9A%B0%EB%8A%94-%EB%8C%80%EA%B7%9C%EB%AA%A8-%EC%8B%9C%EC%8A%A4%ED%85%9C-%EC%84%A4%EA%B3%84-%EA%B8%B0%EC%B4%88-11%EC%9E%A5-%EB%89%B4%EC%8A%A4-%ED%94%BC%EB%93%9C-%EC%8B%9C%EC%8A%A4%ED%85%9C-%EC%84%A4%EA%B3%84-i6re3cjt

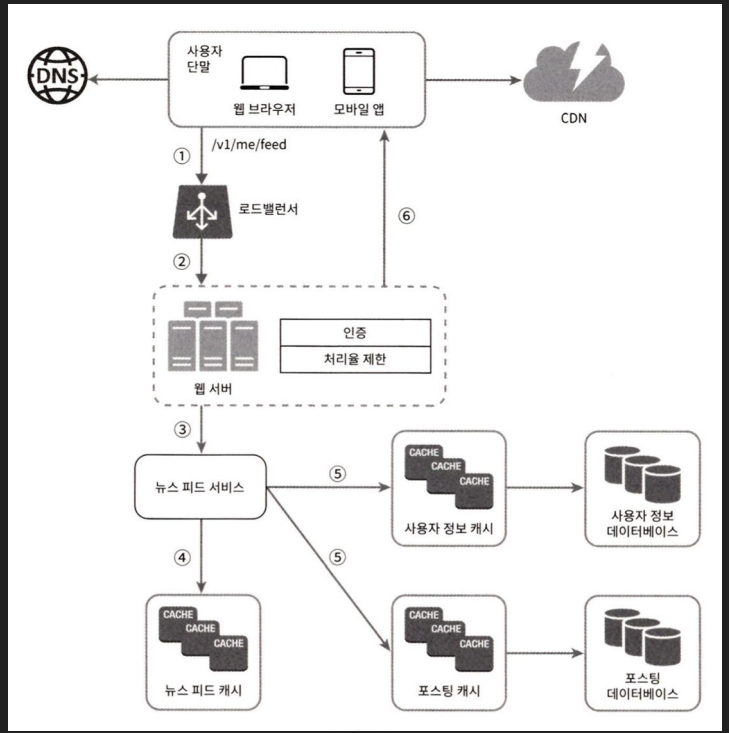
# 4단계 - 1 | 요약

하나의 서비스를 여러 개의 서비스로 분할하여 배포

각 서비스의 연결고리는 Fanout Pattern을 사용하여 추상화

서비스에 맞게 캐시, 데이터베이스를 적절히 사용하여 병목 지점을 예방

# 4단계 - 2 | 구간 별 장애 발생 시의 상황




## 사용자 정보 캐시, 포스팅 캐시가 다운될 경우

- 대다수의 메모리 기반 데이터베이스(In-mem DB)는 데이터가 휘발되기 때문에, 일시적으로 성능 저하가 발생할 수 있다.

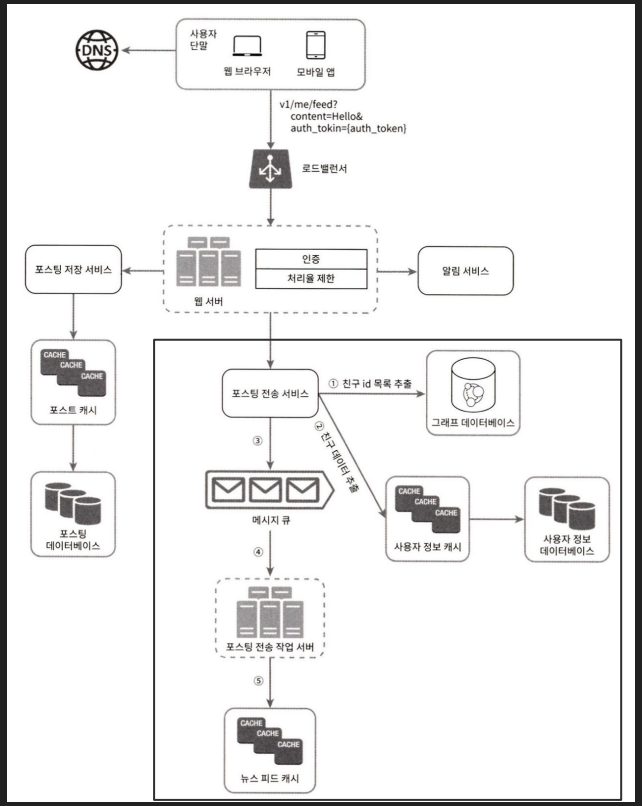
## 사용자 정보 캐시, 포스킹 캐시가 오버플로우 되는 경우

- 가장 오래된 정보부터 지워지도록 구성하더라도 캐시가 적정량 보다 못미칠 경우, 반복적인 메모리 정리 작업이 진행될 수 있다. 따라서 스케일 업이나 아웃을 통해서 이를 방지해야 한다.

## 뉴스 피드 캐시가 다운될 경우

- 뉴스 피드 캐시가 다운되면, 피드를 보여줄 방법이 존재하지 않아보인다. 따라서 뉴스 피드 캐시는 고가용성(HA)이 확보될 수 있게 여러 지역에 배포를 해야 한다. 또한 장애로 휘발된 데이터가 복구되거나, 여러 지역의 메모리가 동기화될 수 있는 설정 또한 필요하다.

# 4단계 - 3 | 운영 측면




**Fanout Pattern은 디버깅이 어려워서**, 로그 및 모니터링이 반드시 필요하다.

- 높은 가용성(HA)과 확장성을 보장하는 Grafana + Prometheus, Pinpoint, Datadog과 같은 서비스들을 적절히 사용할 필요가 있다.

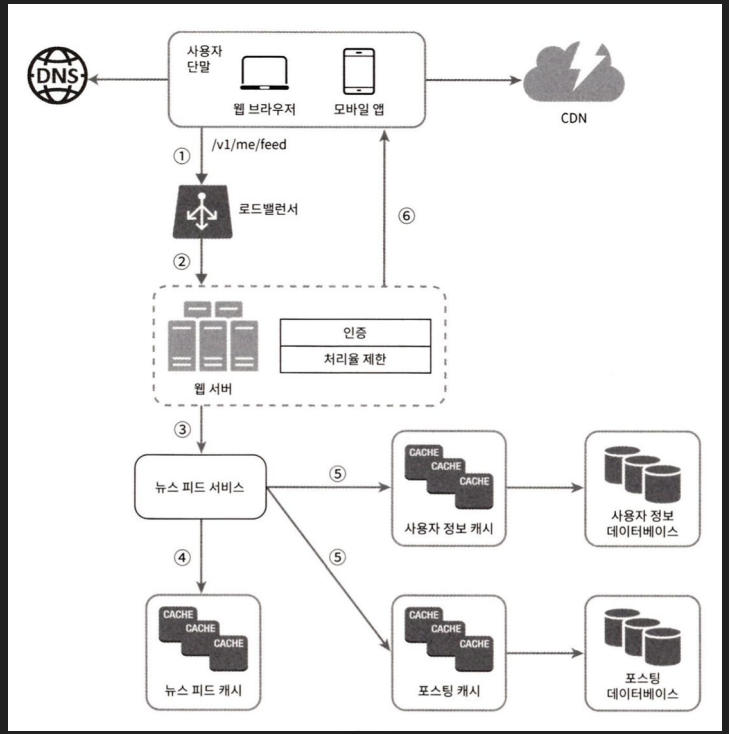
## 개별 서비스에 대한 알림 설정 필요

서비스의 일부/전체가 다운될 경우에 재해복구 알림이 반드시 필요하다.
나아가서 알림을 통해서 자동 복구(Failover)가 되는 전략을 마련해둘 필요가 있다.

캐시, 데이터베이스 등은 5.2.에서 말한 바와 같이 조치할 필요가 있다.
특히 데이터베이스는 Multi-Region에 배포하여 더 높은 가용성을 보장할 수 있다.

그 외우 경우, 목표 RTO에 맞는 Active-Passive 혹은 Active-Active 전략이 필요하다.

# 4단계 - 4 | 백만 사용자가 된다면, 일어날 법한 일들




추가 할당 가능한 서버 자원이 없을 경우, 어떻게할 것인가?

→ 서버 상한선의 70~80% 가까이 되었을 때의 알림이 필요하다.

→ 또한 90%가 넘어가면 별도의 계정을 사용할 준비가 필요하다.

→ 이에 따라 별도 계정에 VPC, Subnet, Peering(각 VPC 들을 연결) 해놔야 한다.

사용 중인 서버들에서 낭비되고 있는 자원들은 어떻게 할 것인가?

→ 1개 컴퓨터에 1개 서비스를 띄우는 것이 아니라 가상화 기술을 이용해서 자원을 효율적으로 활용할 수 있을 것이다.

로드 밸런서에 과한 부하가 걸릴 경우에 어떻게 할 것인가?

→ 규모가 큰 서비스들은 별도의 로드밸런서를 써야하는가?

→ 혹은 L7 로드밸런서에서 L4 로드밸런서로 변경해야 하는가?

할당 가능한 Private IP가 없을 경우에는 어떻게 할 것인가?

→ IPAM 등을 이용해서 VPC, Subnet 내의 활성 Private IP를 체크 가능

감사합니다.