8️⃣

# 분산 이메일 서비스

## 문제 이해 및 설계 범위 확정

**비기능 요구사항**

- 안정성
  - 이메일 데이터는 소실되어서는 안 됨
- 가용성
  - 이메일과 사용자 데이터를 여러 노드에 자동으로 복제하여 가용성을 보장해야 함
  - 부분적으로 장애가 발생해도 시스템은 계속 동작해야 함
- 확장성
  - 사용자 수가 늘어나도 감당할 수 있어야 함
  - 사용자나 이메일이 많아져도 시스템 성능은 저하되지 않아야 함
- 유연성과 확장성
  - 새 컴포넌트를 더하여 쉽게 기능을 추가하고 성능을 개선할 수 있어야 함

**개략적 규모 추정**

- 10억 명의 사용자
- 한 사람이 하루에 보내는 평균 이메일 수 = 10
- 이메일 전송 QPS = 10^9 * 10 / 10^5 = 100000
- 한 사람이 하루에 수신하는 이메일 수 = 평균 40
- 메일 하나의 메타데이터 = 평균 50KB

  첨부 파일을 포함하지 않음
- 1년간 메타데이터를 유지하기 위한 스토리지 요구사항

= 10억 명 사용자 * 하루 40건 이메일 * 365일 * 50KB = 730PB

- 첨부 파일을 포함하는 이메일의 비율 = 20%

  첨부 파일의 평균 크기 = 500KB

- 1년간 첨부 파일을 보관하는 데 필요한 저장 용량

  = 하루 40건 이메일 * 365일 * 20% * 500KB = 1460PB

⇒ 많은 데이터를 처리해야 함

⇒ 분산 데이터베이스 솔루션이 필요함

# 개략적 설계안 제시 및 동의 구하기

## 이메일 101

**이메일 프로토콜**

- SMTP
  - Simple Mail Transfer Protocol
  - 이메일을 한 서버에서 다른 서버로 보내는 표준 프로토콜
- POP
  - Post Office Protocol
  - 이메일을 가져오는 목적으로 가장 널리 사용되는 프로토콜 중 하나
  - 원격 메일 서버에서 이메일을 수신하고 다운로드하기 위해 사용하는 표준 프로토콜
  - 단말로 다운로드된 이메일은 서버에서 삭제됨
  - 한 대의 단말에서만 이메일을 읽을 수 있음
  - 아메일을 일부만 읽을 수 없어서 이메일을 확인하려면 전부 내려 받아야 함
- IMAP
  - 이메일을 가져오는 목적으로 가장 널리 사용되는 프로토콜 중 하나

    - 클릭하지 않으면 메시지는 다운로드 되지 않고, 메일 서버에서 지워지지도 않음
    - 여러 단말에서 이메일 읽기 가능
    - 개인 이메일 계정에서 가장 널리 사용되는 프로토콜
    - 이메일을 실제로 열기 전에는 헤더만 다운로드하기 때문에 인터넷 속도가 느려도 잘 동작함
- HTTPS
    - 기술적으로는 메일 전송 프로토콜은 아님
    - 웹 기반 이메일 시스템의 메일함 접속에 이용될 수 있음

**도메인 이름 서비스(DNS)**

- 수신자 도메인의 메일 교환기 레코드 검색에 이용됨

**첨부 파일**

- 일반적으로 Base64 인코딩을 사용함
- 일반적으로 첨부 파일에는 크기 제한이 있음
- 다목적 인터넷 메일 확장
    - Multi-purpose Internet Mail Extension
    - 인터넷을 통해 첨부 파일을 전송할 수 있도록 하는 표준 규격

# 전통적 메일 서버

**전통적 메일 서버 아키텍처**(송신자: 아웃룩 → 수신자: 지메일)

1. 송신자는 아웃룩 클라이언트에 로그인하여 이메일을 전송
2. 이메일은 아웃룩 메일 서버로 전송

   SMTP 사용
3. 아웃룩 메일 서버는 DNS 질의를 통해 수신자 SMTP 서버 주소를 찾음

4. 주소를 찾아서 해당 메일 서버로 이메일 전송
5. 지메일 서버는 이메일을 저장하고 수신자가 읽어갈 수 있도록 함
6. 수신자가 지메일에 로그인하면 지메일 클라이언트는 IMAP/POP 서버를 통해 새 이메일을 가져옴

**저장소**

- 이메일을 파일 시스템의 디렉터리에 저장함
- 각각의 이메일은 고유한 이름을 가진 별도 파일로 보관
- 각 사용자의 설정 데이터와 메일함은 사용자 디렉터리에 보관
- 파일과 디렉터리를 활용하는 방안은 사용자나 메일의 수가 많은 경우에는 부적합함
  - 디스크 I/O 병목 현상 발생
- 이메일을 서버의 파일 시스템에 보관하였기 때문에 가용성과 안정성 요구사항도 만족할 수 없음
  - 디스크 손상이나 서버 장애가 얼마든지 발생할 수 있음

⇒ 더 안정적인 분산 데이터 저장소 계층이 필요함

# 분산 메일 서버

**이메일 API**

- 이메일 API의 의미는 메일 클라이언트마다, 이메일 생명주기 단계마다 달라질 수 있음
  - 모바일 단말 클라이언트를 위한 SMTP/POP/IMAP API
  - 송신 측 메일 서버와 수신측 메일 서버 간의 SMTP 통신
  - 대화형 웹 기반 이메일 애플리케이션을 위한 HTTP 기반 RESTful API

⇒ 책에서는 가장 중요한 API만 다룰 것

| PATH | 내용 |
| --- | --- |
| [POST] /v1/messages | To, Tc, Bcc 헤더에 명시된 수신자에게 메시지를 전송 |

| PATH | 내용 |
| --- | --- |
| [GET] /v1/folders | 주어진 이메일 계정에 존재하는 모든 폴더를 반환 |
| [GET] /v1/folders/{:folder_id}/messages | 주어진 폴더 아래의 모든 메시지를 반환 |
| [GET] /v1/messages/{:message_id} | 주어진 특성 메시지에 대한 모든 정보를 반환 |

**분산 메일 서버 아키텍처**

- 웹메일
  - 웹브라우저를 사용해 메일을 주고 받음
- 웹서버
  - 사용자가 이용하는 요청/응답 서비스
  - 로그인, 가입, 사용자 프로파일 등에 대한 관리 기능 담당
  - 본 설계안에서는 위에 언급한 이메일 API를 웹서버를 통해 통신
- 실시간 서버
  - 새로운 이메일 내역을 클라이언트에 실시간으로 전달하는 역할
- 메타데이터 데이터베이스
- 첨부 파일 저장소
  - 객체 저장소를 사용할 것
  - 이미지나 동영상 등의 대용량 파일을 저장하는 데 적합한 S3를 사용할 것
  - 카산드라는 적합하지 않음
    - 실질적으로 1MB 이상의 파일을 지원하지 못함
    - 레코드 캐시를 사용하기 어려움
    - 첨부 파일이 너무 많은 메모리를 잡아먹을 것
- 분산 캐시
  - 자주 읽을 가능성이 높은 이메일을 메모리에 캐시함
  - 레디스를 사용할 것
    - 리스트 같은 다양한 기능을 제공

        - 규모 확장도 용이
- 검색 저장소
    - 분산 문서 저장소
    - 고속 텍스트 검색을 지원하는 역 인덱스를 자료 구조로 사용할 것

**이메일 전송 절차**

1. 사용자가 웹메일 환경에서 메일 작성 후 로드밸런서로 전송
2. 로드밸런서는 처리율 제한 한도를 넘지 않는 선에서 요청을 웹서버로 전달
3. 웹서버
    - 기본적인 이메일 검증
        - 이메일 크기 한도 등 사전에 미리 정의된 규칙을 사용하여 수신된 이메일 검사
    - 수신자 이메일 주소 도메인이 송신자 이메일 주소 도메인과 같은지 검사
        - 같다면 스팸 여부와 바이러스 감염 여부를 검사하고, 송수신자 각각의 '보낸 편지함'과 '받은 편지함'에 저장
        - 수신인 측 클라이언트는 RESTful API를 사용하여 이메일을 바로 가져올 수 있음
        - 4단계 이후는 수행할 필요가 없음
4. 메시지큐
    - 기본적인 검증을 통과한 이메일은 외부 전송 큐로 전달하고, 첨부 파일이 너무 크다면 첨부 파일은 객체 저장소에 따로 저장
    - 기본적인 검증을 통과하지 못한 이메일은 에러 큐에 보관
5. 외부 전송 담당 SMTP 작업 프로세스는 외부 전송 큐에서 메시지를 꺼내어 이메일의 스팸 및 바이러스 감염 여부를 확인
6. 검증 절차를 통과한 이메일은 저장소 계층 내의 '보낸 편지함'에 저장
7. 외부 전송 담당 SMTP 작업 프로세스가 수신자의 메일 서버로 메일을 전송

**메일이 처리되지 않고 외부 전송 큐에 오랫동안 남아 있는 경우**

- 수신자 측 메일 서버에 장애 발생
  - 나중에 메일을 다시 전송
  - 지수적 백오프가 좋은 전략일 수도
- 이메일을 보낼 큐의 소비자 수자 불충분
  - 더 많은 소비자를 추가하여 처리 시간을 단축

**이메일 수신 절차**

1. 이메일이 SMTP 로드밸런서에 도착
2. 로드밸런서는 트래픽을 여러 SMTP 서버로 분산
3. 첨부 파일이 큐에 들어가기 너무 큰 경우에는 첨부 파일 저장소에 보관
4. 이메일을 수신 이메일 큐에 넣음

   메일 처리 작업 프로세스와 SMTP 서버 간의 결합도를 낮춤

   수신되는 이메일의 양이 폭증하는 경우 버퍼 역할도 함
5. 스팸 메일을 걸러내고 바이러스를 차단함
6. 이메일을 메일 저장소, 캐시, 객체 저장소 등에 보관
7. 수신자가 온라인 상태인 경우에는 이메일을 실시간 서버로 전달
8. 수신자가 오프라인 상태인 경우에는 저장소 계층에 보관하고, 온라인 상태가 되면 웹메일 클라이언트가 웹 서버에 RESTful API를 통해 연결
9. 웹 서버는 새로운 이메일을 저장소 계층에서 가져와 클라이언트에 반환

# 상세 설계

## 메타데이터 데이터베이스

**이메일 메타데이터의 특성**

- 이메일의 헤더는 일반적으로 작고, 빈번하게 이용됨
- 이메일 본문의 크기는 작은 것부터 큰 것까지 다양하지만 사용 빈도는 낮음
- 일반적으로 사용자는 이메일을 한 번만 읽음
- 이메일 관련 작업(이메일 가져오기, 읽은 메일로 표시, 검색)은 사용자별로 격리 수행되어야 함
    - 해당 사용자만 읽을 수 있어야 함
    - 이메일에 대한 작업도 해당 사용자만 수행할 수 있어야 함
- 데이터의 신선도는 데이터 사용 패턴에 영향을 미침
    - 사용자는 보통 최근 메일만 읽음
    - 만들어진 지 16일 이하 데이터에 발생하는 읽기 질의 비율은 전체 질의의 82%
- 데이터의 높은 안정성이 보장되어야 함
    - 데이터의 손실을 용납되지 않음

**올바른 데이터베이스의 선정**

- 관계형 데이터베이스
    - 이메일을 효율적으로 검색할 수 있음
        - 헤더와 본문에 대한 인덱스 걸기
    - 하지만 데이터 크기가 작을 때 적합
        - BLOB을 사용할 수도 있지만 검색 질의 성능이 좋지 않음
        - BLOB 컬럼의 데이터를 접근할 때마다 디스크 I/O가 발생함
- 분산 객체 저장소
    - 이메일의 원시 데이터를 그대로 객체 저장소에 저장
    - 백업 데이터를 보관하기에는 좋지만 다른 기능들을 구현하기에는 좋지 않음
        - 이메일 읽음 표시
        - 키워드 검색
        - 이메일 타래

- NoSQL 데이터베이스
  - 지메일은 구글 빅테이블을 저장소로 사용하고 있음 = 실현 가능한 방안
    - 빅테이블은 오픈소스가 아님
    - 어떻게 구현했는지 모름
  - 카산드라가 좋은 대안이 될 수도 있지만 대형 이메일 서비스 제공 업체 가운데 카산드라를 사용하는 곳은 아직 없음

⇒ 본 설계안이 필요로 하는 기능을 완벽히 지원하는 데이터베이스는 없음

⇒ 대형 이메일 서비스 업체는 대부분 독자적인 데이터베이스 시스템을 만들어 사용

⇒ 면접에서는 불가능하니 특정 조건을 충족해야 한다는 것을 설명하기

- 어떤 단일 컬럼의 크기는 한 자릿수 MB 정도일 수 있음
- 강력한 데이터 일관성이 보장되어야 함
- 디스크 I/O가 최소화되도록 설계되어야 함
- 가용성이 아주 높아야 함
- 일부 장애를 감내할 수 있어야 함
- 증분 백업이 쉬워야 함

**데이터 모델**

데이터를 저장하는 한 가지 방법은 user_id를 파티션 키로 사용하여 특정한 사용자의 데이터는 항상 같은 샤드에 보관하는 것인데 메시지를 여러 사용자와 공유할 수 없다는 문제가 있지만 본 면접의 요구사항과는 관계 없음

- 파티션 키
  - 데이터를 여러 노드에 분산하는 구실을 함
  - 일반적으로 통용되는 규직은 데이터가 모든 노드에 균등하게 분산되도록 하는 파티션 키를 골라야 함
- 클러스터 키
  - 같은 파티션에 속한 데이터를 정렬하는 구실을 함

1. 특정 사용자의 모든 폴더 질의
    - user_id를 파티션 키로 사용
    - 어떤 사용자의 모든 폴더는 같은 파티션 안에 있음

| **folders_by_user** | | |
|---|---|---|
| user_id | UUID | K |
| folder_id | UUID | |
| folder_name | TEXT | |

2. 특정 폴더에 속한 모든 이메일 표시
    - 사용자가 자기 메일 폴더를 열면 이메일은 가장 최근 이메일부터 오래된 것 순으로 정렬되어 표시
    - <user_id, folder_id> 형태의 복합 파티션 키 사용
    - email_id는 이메일을 시간순으로 정렬할 때 사용되는 클러스터 키

| **emails_by_folders** | | |
|---|---|---|
| user_id | UUID | K |
| folder_id | UUID | K |
| email_id | TIMEUUID | C ↓ |
| from | TEXT | |
| subject | TEXT | |
| preview | TEXT | |
| is_read | BOOLEAN | |

3. 이메일 생성/삭제/수신
    - 이메일 상세 정보를 가져오는 방법

```
SELECT * FROM emails_by_user WHERE email_id = 123
```

    - 한 이메일에는 여러 첨부 파일이 있을 수 있음
        - email_id와 file_name 필드를 같이 사용하면 모든 첨부 파일을 질의할 수 있음

| **emails_by_user** | | |
|---|---|---|
| user_id | UUID | K |
| email_id | TIMEUUID | C ↓ |
| from | TEXT | |
| to | LIST<TEXT> | |
| subject | TEXT | |
| body | TEXT | |
| attachments | LIST<filename\|size> | |

| **attachments** | | |
|---|---|---|
| email_id | TIMEUUID | C |
| file_name | TEXT | K |
| url | TEXT | |

4. 읽은 또는 읽지 않은 모든 메일
    - 관계형 데이터베이스로 도메인 모델을 구현하는 경우

```
SELECT * FROM emails_by_folder
WHERE user_id = <user_id> and folder_id = <folder_id> and
ORDER BY email_id;

-- is_true 값에 따라 읽음 또는 읽지 않음 여부를 변경할 수 있음
-- true = 읽음 / false = 읽지 않음
```

    - 본 설계안은 NoSQL 데이터베이스로 도메인을 구현함
        - 파티션 키와 클러스터 키에 대한 질의만 허용함 = is_read 필드 사용 불가능
        - 주어진 폴더에 속한 모든 메시지를 가져온 다음에 애플리케이션 단에서 필터링을 수행할 수도 있음
            - 대규모 서비스에는 적합하지 않음
        - 비정규화를 진행하는 것이 적합

- emails_by_folder 테이블을 read_emails, unread_emails 두 테이블로 분할
- 읽지 않은 메일을 읽은 메일로 변경할 경우 이메일을 unread_emails 테이블에서 삭제한 다음 read_emails 테이블로 옮기기
- 특정 폴더 안의 읽지 않은 모든 메일을 가져오는 질의

```
SELECT * FROM unread_emails
WHERE user_id = <user_id> and folder_id = <folder_id
ORDER BY email_id;
```

| **read_emails**<br>**unread_emails** | | |
|---|---|---|
| user_id | UUID | K |
| folder_id | UUID | K |
| email_id | TIMEUUID | C ↓ |
| from | TEXT | |
| subject | TEXT | |
| preview | TEXT | |

**이메일 타래 가져오기**

- 모든 답장을 최초 메시지에 타래로 엮어 보여주는 기능
- JWZ 같은 알고리즘을 통해 구현
  - 이메일 헤더에는 Message-ID, In-Reply-To, References 세 가지 필드로 구현됨
  - 이메일 클라이언트는 타래 내의 모든 메시지가 사전에 메모리로 로드되어 있는 경우 전체 대화 타래를 재구성해 낼 수 있게 됨

**일관성 문제**

- 높은 가용성을 달성하기 위해 다중화에 의존하는 분산 데이터베이스는 일관성과 가용성 사이에서 타협적인 결정을 내림

- 이메일 시스템의 경우, 데이터의 정확성이 아주 중요함
  - 반드시 하나의 주 사본을 통해 서비스된다고 가정해야 함
  - 장애가 발생하면 클라이언트는 다른 사본을 통해 주 사본이 복원될 때까지 동기화/갱신 작업을 완료할 수 없음

⇒ 데이터 일관성을 위해 가용성을 희생

## 이메일 전송 가능성

- 전용 IP
  - 이메일을 보낼 때는 전용 IP 주소를 사용
  - 대부분의 이메일 서비스 사업자는 아무 이력이 없는 IP, 새로운 IP 주소에서 온 메일을 무시함
- 범주화
  - 범주가 다른 이메일은 다른 IP 주소를 사용
  - ex) 마케팅 목적의 이메일은 중요한 이메일과 같은 서버에서 발송하지 않음
    - ISP가 모든 이메일을 판촉 메일로 분류할 수 있음
- 발신인 평판
  - 새로운 이메일 서버의 IP 주소는 사용 빈도를 서서히 올리는 것이 좋음
    - 그래야 좋은 평판이 쌓임
    - 오피스365, 지메일 등의 대형 사업자가 해당 IP 주소에서 발송되는 메일을 스팸으로 분류할 가능성이 낮아짐
    - 아마존 SES에 따르면, 새로운 IP 주소를 메일 발송에 아무 문제 없이 쓸 수 있게 되는 데 대략 2~6주가 걸린다고 함
- 스팸 발송자의 신속한 차단
  - 스팸을 뿌리는 사용자는 서버 평판을 심각하게 훼손하기 전에 시스템에서 신속히 차단해야 함
- 피드팩 처리

    - 불만 신고가 접수되는 비율을 낮추고 스팸 계정을 신속히 차단하기 위해서는 ISP 측에서의 피드백을 쉽게 받아 처리할 수 있는 경로를 만드는 것이 중요
    - 이메일이 전달되지 못하거나 가용자로부터 불만 신고가 접수된 후 발생하는 일
        - 경성 반송: 수신인의 이메일 주소가 올바르지 않아 ISP가 전달을 거부한 경우
        - 연성 반송: ISP 측의 이메일 처리 자원 부족 등의 이유로 실시적으로 전달할 수 없었던 경우
        - 불만 신고: 수신인이 '스팸으로 신고' 버튼을 누른 경우

      ⇒ 세 가지 경우 각각에 대해 별도의 큐를 유지하여 별도로 관리할 수 있도록 함
- 이메일 인증
    - 피싱이나 프리텍스팅이 전체 유출 사고에서 차지하는 비중은 93%
    - 피싱에 대응하는 보편적인 전략으로는 SPF, DKIM, DMARC, ...

이메일이 목적지에 성공적으로 도착하도록 하기는 매우 어려움

도메인 지식도 필요하고, ISP와 좋은 관계를 유지해야 함

## 검색

- 기본적인 이메일 검색은 보통 이메일 제목이나 본문에 특정 키워드 포함 여부
- 고급 기능에는 발신인, 제목, 읽지 않음과 같이 메일 속성에 따른 필터링
- 이메일이 전송, 수신, 삭제될 때마다 색인 작업을 수행해야 함

⇒ 이메일 시스템의 검색 기능에서는 쓰기 연산이 읽기 연산보다 훨씬 많이 발생함

**일래스틱서치**

- 질의가 대부분 사용자의 이메일 서버에서 실행되므로 user_id를 파티션 키로 사용하여 같은 사용자의 이메일은 같은 노드에 묶어 높음
- 사용자는 검색 버튼을 누르고 결과가 수신될 때까지 대기
    - 동기 방식으로 처리

- 이메일 전송, 이메일 수신, 이메일 삭제와 같은 이벤트는 처리 결과를 사용자에게 전달할 필요가 없음
- 필요한 것은 색인 작업
  - background 작업 형태로 처리 가능
- 2021년 6월을 기준으로 가장 널리 사용되고 있은 검색 엔진 데이터베이스
  - 이메일 검색에 필요한 텍스트 기반 검색을 잘 지원함
  - 주 이메일 저장소와 동기화를 맞추는 부분이 까다롭기는 함

**맞춤형 검색 솔루션**

- 대규모 이메일 서비스 사업자는 보통 자기 제품에 고유한 요구사항을 만족시키기 위해 검색 엔진을 자체적으로 개발해 사용하는데 이는 아주 복잡한 작업이기 때문에 이번 장에서 다루지 않을 것
- 검색 솔루션을 구현할 때 마주하게 될 주요 과제인 디스크 I/O 병목 문제만 간단하게 다룰 것
- 매일 저장소에 추가되는 메타데이터와 첨부 파일의 양을 PB 수준이고, 하나의 이메일 계정에 오십 만개가 넘는 이메일이 저장되는 것도 드문 일이 아니기 때문에 메일 색인 서버의 주된 병목은 보통 디스크 I/O
- 색인을 구축하는 프로세스는 다량의 쓰기 연산을 발생시킬 수밖에 없으므로 Log-Structured Merge 트리를 사용하여 디스크에 저장되는 색인을 구조화하는 것이 바람직한 전략
  - 빅테이블, 카산드라, RocksDB, ...의 핵심 자료 구조
  - 쓰기 경로는 순차적 쓰기 연산만 수행하도록 최적화되어 있음
  - 새로운 이메일이 도착하면 우선 메모리 캐시로 구현되는 0번 계층에 추가되고, 메모리에 보관된 데이터의 양이 사전에 정의된 임계치를 넘으면 데이터는 다음 계층에 보합됨
  - 자주 바뀌는 데이터를 그렇지 않은 데이터와 분리할 수 있음
    - 데이터를 두 개 파트로 나누고, 어떤 요청이 폴더 변경에 관한 것이면 폴더 정보만 바꾸고 이메일 데이터는 내버려 둠

### 일래스틱서치 vs 맞춤형 검색 솔루션

| | 일래스틱서치 | 맞춤형 검색 솔루션 |
|---|---|---|
| **규모 확장성** | 어느 정도까지 확장 가능 | 이메일 사용 패턴에 다라 시스템을 최적화할 수 있음<br>규모 확장이 용이 |
| **시스템 복잡도** | 데이터 저장소와 일래스틱서치<br>두 가지 상이한 시스템을 동시에 유지해야 함 | 하나의 시스템 |
| **데이터 일관성** | 한 데이터의 두 사본이 존재<br>메타데이터 저장소와 일래스틱서치 유지하기 까다로움 | 메타데이터 저장소에 하나의 사본만이 유지 |
| **데이터 손실 가능성** | 없음<br>색인이 손상되면 주 저장소의 데이터를 사용해 복구 | 없음 |
| **개발 비용** | 통합하기 쉬운 편<br>대규모의 이메일 검색이 필요한 경우에는 일래스틱서치를 전담하는 팀이 필요 | 맞춤형 검색 솔루션 구현이 필요<br>굉장히 많은 엔지니어링 노력이 필요 |

소규모의 이메일 시스템을 구축하는 경우에는 일래스틱서치가 좋은 선택지

대규모 시스템을 구축하는 경우에는 데이터베이스에 내장된 전용 검색 솔루션을 사용하는 것이 바람직

## 규모 확장성 및 가용성

### 규모 확장성

각 사용자의 데이터 접근 패턴은 다른 사용자와 무관

⇒ 시스템의 대부분 컴포넌트는 수평적으로 규모 확장이 가능

### 가용성

- 데이터를 여러 데이터센터에 다중화하는 것이 필요
- 사용자는 네트워크 토폴로지 측면에서 보았을 때 자신과 물리적으로 가까운 메일 서버와 통신

- 장애 때문에 통신이 불가능한 네트워크 영역이 생기면 다른 데이터센터에 보관된 메세지를 이용

# 마무리

**추가로 논의해 볼 만한 주제**

- 결함 내성
  - 시스템의 많은 부분에 장애가 발생할 수 있음
  - 노드 장애, 네트워크 문제, 이벤트 전달 지연 등의 문제를 어떻게 대처할 것인가
- 규정 준수
  - 이메일 서비스는 전 세계 다양한 시스템과 연동해야 함
  - 각 나라에는 준수해야 할 법규가 있음
  - 합법적 감청은 이 분야의 대표적 특징
- 보안
  - 민감한 정보가 포함되기 때문에 보안이 중요함
  - 지메일이 제공하는 기능
    - 피싱이나 멀웨어 공격을 방지하는 피싱 방지
    - 안전하지 않은 사이트를 경고하는 사전 경고
    - 의심스러운 로그인 시도를 차단하는 계정 안전
    - 송신자가 메시지에 대한 보안 정책을 설정할 수 있는 기밀 모드
    - 타인이 이메일 내용을 엿보지 못하도록 하는 이메일 암호화
- 최적화
  - 같은 이메일이 여러 수신자에게 전송되는 경우도 있기 때문에 같은 첨부 파일이 그룹 이메일 객체 저장소에 여러 번 저장되는 경우가 있음

- 저장하기 전에 저장소에 이미 동일한 첨부 파일이 있는지 확인하면 저장 연산 실행 비용을 최적화할 수 있음