분산 이메일 서비스

▼ 1️⃣ 설계 범위 확정

질문 목록

- 회원 수 10억 명
- 중요한 기능
  - 이메일 수발신
  - 모든 이메일 조회
  - 읽음 여부에 따른 이메일 필터링
  - 검색(제목, 발신인, 내용)
  - 스팸 및 바이러스 방지
- 메일 서버 연결 방식
  - SMTP, POP, IMAP, HTTP
- 첨부파일 지원 여부

비기능 요구사항

- **안정성**: 이메일 데이터는 소실되면 안됨
- **가용성**: 이메일, 사용자 데이터를 여러 노드에 자동 복제하여 가용성 보장
- **확장성**: 사용자 수가 늘어나도 감당 가능해야 함
- **유연성 / 확장성**: 새 컴포넌트를 더해 쉽게 기능 추가 또는 성능 개선

규모 추정

- 10억 명 사용자
- 하루 1인 평균 이메일 발송 건수 = `10건`
  - 이메일 전송 QPS = `100,000`
- 하루 1인 평균 이메일 수신 건수 = `40건`
- 1개 이메일 메타데이터 평균 `50KB` (첨부파일 미포함)

    - 메타데이터는 DB에 저장한다고 가정
    - 1년간 유지하기 위한 스토리지 요구사항 = 10억명 * 하루 40건 * 365일 * 50KB = `730PB`
- 첨부파일을 포함하는 이메일의 비율 = `20%`, 첨부 파일 평균 크기 `500KB`
    - 1년간 파일 보관에 필요한 저장 용량 = 10억명 * 하루 40건 * 365일 * 20% * 500KB = `1460PB`

▼ 2️⃣ 개략적 설계

이메일 서버에 대해 알아야 할 사항과 이메일 서버의 진화 과정을 다뤄보자

이메일 101

이메일 프로토콜

SMTP

이메일을 다른 서버로 보내는 표준 프로토콜

POP

이메일 클라이언트가 원격 메일 서버에서 이메일을 받는 표준 프로토콜

다운로드된 이메일은 서버에서 삭제된다. → 한 대 단말에서만 읽기 가능

이메일을 확인하기 위해 전부 내려받아야 한다. → 용량이 큰 메일은 시간이 오래 걸림

IMAP

이메일 클라이언트가 원격 메일 서버에서 이메일을 받는 표준 프로토콜

클릭 전까지 다운로드도, 서버에서 삭제되지도 않는다. → 여러 단말에서 읽기 가능

가장 널리 사용되는 프로토콜

인터넷 속도가 느려도 잘 동작한다 → 이메일 열기 전까지 헤더만 다운로드

HTTPS

웹 기반 이메일 시스템의 메일함 접속에 이용

도메인 이름 서비스

- DNS 서버는 수신자 도메인의 메일 교환기(MX) 레코드 검색에 이용된다.
- DNS 레코드 검색 시 MX 레코드가 표시된다.

- MX 레코드는 우선순위와 메일 서버 목록으로 이루어진다.
- 우선순위가 높을수록(값이 작을수록) 최우선으로 쓰인다.

첨부 파일

- 일반적으로 Base64 인코딩을 사용한다.
- 일반적으로 크기 제한이 있다.
- **MIME(다목적 인터넷 메일 확장)**: 인터넷을 통해 첨부 파일을 전송할 수 있도록 하는 표준 규격

전통적 메일 서버

보통 서버 한 대로 운용되는, 사용자가 많지 않을 때 잘 동작하는 시스템

프로세스

1. A는 아웃룩 클라이언트에 로그인하여 이메일을 보낸다.
2. 이메일은 아웃룩 메일 서버로 전송된다. (아웃룩 클라이언트와 메일 서버 통신 간 SMTP를 사용한다.)
3. 아웃룩 메일 서버는 DNS 질의를 통해 수신사 SMTP 서버 주소를 찾고 해당 메일 서버로 이메일을 보낸다.
4. 수신자 서버(지메일)는 이메일을 저장하고 수신자인 B가 읽어갈 수 있도록 한다.
5. 수신자가 지메일에 로그인하면 지메일 클라이언트는 IMAP/POP 서버를 통해 새 이메일을 가져온다.

저장소

- 이메일은 파일 시스템의 디렉터리에 저장한다.
- 각각의 이메일은 고유 이름을 가진 별도 파일로 보관한다.
- 각 사용자의 설정 데이터와 메일함은 사용자 디렉터리에 보관한다. (Maildir)

⊖ 이메일 검색, 백업 목적으로는 사용이 어려웠다.

⊖ 이메일 양이 많아지고 파일 구조가 복잡해질수록 디스크 I/O 병목이 발생한다.

⊖ 가용성, 안정성이 떨어진다.

분산 메일 서버

전통적인 방식에서의 문제를 해결하기 위해 현대적 사용 패턴을 지원

이메일 API

POST /v1/messages

To, Tc, Bcc 헤더에 명시된 수신자에게 메시지를 전송

GET /v1/folders

주어진 이메일 계정에 존재하는 모든 폴더를 반환

```
[{
    id: string              # 고유한 폴더 식별자
    name: string            # 폴더 이름
    user_id: string         # 계정 소유자 ID
}]
```

GET /v1/folders/:folder_id/messages

주어진 폴더 아래의 모든 메시지 반환

GET /v1/messages/:message_id

주어진 특정 메시지에 대한 모든 정보 반환

분산 이메일 서버 아키텍처

- **웹메일**: 사용자는 웹브라우저를 사용해 메일을 수발신한다.
- **웹서버**: 사용자가 이용하는 서비스에 대해 관리한다. 모든 이메일 API 요청은 웹서버를 통한다.
- **실시간 서버**: 새로운 이메일 내역을 실시간으로 클라이언트에 전달한다. (stateful)
  - 웹소켓에 브라우저 호환성 문제 발생 가능성이 있어 웹소켓 + 롱 폴링(백업)을 사용한다.
- **메타데이터 데이터베이스**: 이메일 제목, 본문, 발신인, 수신인 목록 등을 저장한다.
- **첨부파일 저장소**: 아마존 S3 같은 객체 저장소를 사용해 대용량 파일을 저장한다. (~25MB)
  - 🤔 카산드라를 사용하지 않는 이유?
    - 카산드라 BLOB이 지원하는 최대 크기가 2GB지만 실질적으로 1MB 이상은 지원하지 못함
    - 첨부파일이 너무 많은 메모리를 잡아 먹어 레코드 캐시를 사용하기 어려움
- **분산 캐시**: 최근에 수신된 이메일은 자주 읽는다 → 레디스를 사용해 캐시

- **검색 저장소**: 분산 문서 저장소로 고속 텍스트 검색을 지원하는 역 인덱스를 자료 구조로 사용한다.

이메일 발신 절차

1. 사용자가 웹메일 환경에서 메일을 전송한다.
2. 요청은 로드 밸런서로 전송된다.
3. 로드밸런서는 처리율 제한 한도를 넘지 않는 선에서 요청을 웹 서버로 전달한다.
4. 웹 서버는 다음 역할을 담당한다.
    a. **기본적인 이메일 검증**: 크기 한도 등 미리 정의된 규칙을 사용해 검사한다.
    b. **도메인 검사**: 수신자 이메일 주소 도메인이 송신자 이메일 주소 도메인과 같다면 스팸 여부, 바이러스 감염 여부를 검사한다. 모든 검사를 통과하면 송신인의 보낸 편지함, 수신인의 받은 편지함에 저장된다. 이 때 이후 단계는 수행하지 않아도 된다.
5. 메시지 큐 (여기서부터 이메일 주소 도메인이 서로 다른 경우)
    a. 기본적인 검증을 통과한 이메일은 외부 전송 큐로 전달, 실패하면 에러 큐에 보관돈다.
    b. 큐에 넣기에 크기가 큰 이메일은 첨부 파일만 객체 저장소에 따로 저장하고 참조 정보를 보관한다.
6. 외부 전송 담당 SMTP 작업 프로세스는 외부 전송 큐에서 메시지를 꺼내 스팸, 바이러스 여부를 검사한다.
7. 검증 절차를 통과하면 이메일 저장소 계층 내 보낸 편지함에 저장된다.
8. 외부 전송 담당 SMTP 작업 프로세스가 수신자의 메일 서버로 메일을 전송한다.

> ⚠ **외부 전송 큐 운영 시 주의점**
>
> 메일이 처리되지 않고 큐에 오래 남아있으면 그 이유를 분석해야 한다.
> -
> **수신자 측 메일 서버에 장애**: 나중에 다시 전송해야 한다. (지수적 백오프 전략 추천)
> -
> **이메일을 보낼 큐의 소비자 수가 불충분**: 더 많은 소비자를 추가하여 처리 시간을 단축한다.

이메일 수신 절차

1. 이메일이 SMTP 로드밸런서에 도착한다.
2. 로드밸런서는 트래픽을 여러 SMTP 서버로 분산한다.
    a. SMTP 연결은 이메일 수락 정책을 구성해서 유효하지 않은 이메일을 반송할 수 있다.
3. 이메일의 첨부 파일이 너무 크면 첨부 파일 저장소(S3)에 보관한다.
4. 이메일을 수신 이메일 큐에 넣는다.
    a. 메일 처리 작업 프로세스 - SMTP 서버 간 결합도를 낮추어 독립적 규모 확장이 가능하다.
5. 메일 처리 작업 프로세스는 스팸 메일을 걸러 내고 바이러스를 차단한다.
6. 이메일을 메일 저장소, 캐시, 객체 저장소 등에 보관한다.
7. 수신자가 온라인 상태면 이메일을 실시간 서버에 전달한다.
8. 실시간 서버는 수신자 클라이언트가 새 이메일을 실시간으로 받을 수 있게 하는 웹소켓 서버다.
9. 오프라인 상태 사용자의 이메일은 저장소 계층에 보관한다.
    a. 온라인 상태가 되면 웹메일 클라이언트가 웹 서버에 RESTful API를 통해 연결한다.
10. 웹 서버는 새로운 이메일을 저장소 계층에서 가져와 클라이언트에 반환한다.

▼ 3️⃣ 상세 설계

메타데이터 데이터베이스

이메일 메타데이터의 특성을 알아보고 올바른 데이터베이스와 데이터 모델을 고르자

이메일 메타데이터의 특성

- 헤더는 일반적으로 작고, 빈번하게 이용된다.
- 본문의 크기는 다양하지만 사용 빈도는 낮다. (일반적으로 한 번만 읽는다.)
- 이메일 가져오기, 읽은 메일 표시, 검색 등 작업은 사용자 별로 격리 수행돼야 한다.
    - 한 사용자의 이메일은 그 사용자만 읽을 수 있어야한다.
    - 한 사용자의 이메일에 대한 작업은 그 사용자만 할 수 있어야 한다.

- 데이터의 신선도는 데이터 사용 패턴에 영향을 미친다. (보통 최근 메일만 읽는다.)
- 데이터의 높은 안정성이 보장돼야 한다. 손실 절대 불가

데이터베이스 선정

관계형 데이터베이스

- 이메일을 효율적으로 검색할 수 있다.
  - 헤더, 본문에 대한 인덱스를 만들자
- 데이터 크기가 작을 때 적합하기 때문에 질의 성능이 좋지 않다.
  - 비정형 BLOB을 사용할 시 큰 이메일 처리는 가능하나 많은 디스크 I/O가 발생한다.

분산 객체 저장소

- 백업 데이터를 보관하기 좋다.
- 이메일 기능(읽음 표시, 키워드 검색, 이메일 타래)를 구현하기 좋지 않다.

NoSQL 데이터베이스

- 실현 가능한 방안이지만 참고할 사례가 부족하다.

→ 결론) 본 설계안이 필요로 하는 기능을 완벽 지원하는 데이터베이스는 없다.

- 대형 이메일 서비스 업체는 대체로 독자적인 DBMS를 만들어 사용한다.

🤔 우리가 알고 있어야 할 것은?

- 어떤 단일 컬럼 크기는 한 자릿수MB 정도일 수 있다.
- 강력한 데이터 일관성이 보장돼야 한다.
- 디스크 I/O가 최소화되어야 한다.
- 가용성이 아주 높고 일부 장애를 감내할 수 있어야 한다.
- 증분 백업이 쉬워야 한다.

데이터 모델

`user_id`를 파티션 키로 특정 사용자의 데이터는 항상 같은 샤드에 보관하자

단점: 메시지를 여러 사용자와 공유할 수 없다. → 우리 요구사항과 관계 없음

- 파티션 키: 데이터를 여러 노드에 분산시키는 역할, 균등 분산되도록 하는 파티션 키를 골라야 함

- 클러스터 키: 같은 파티션에 속한 데이터를 정렬하는 역할

질의 1) 특정 사용자의 모든 폴더 질의

파티션 키가 `user_id`이기 때문에 사용자의 모든 폴더는 같은 파티션에 있다.

folders_by_user
user_id UUID folder_id UUID folder_name TEXT

질의 2) 특정 폴더에 속한 모든 이메일 질의

같은 폴더에 속한 모든 이메일이 같은 파티션에 속하려면 `<user_id, folder_id>` 형태의 복합 파티션 키를 사용해야 한다.

`email_id`를 시간순으로 정렬 가능한 클러스터 키로 활용할 수 있다.

emails_by_folder
user_id UUID **folder_id UUID**   *email_id TIMEUUID* from TEXT subject TEXT preview TEXT is_read BOOLEAN

질의 3) 이메일 생성/삭제/수신

```
// 특정 이메일의 상세 정보
SELECT * FROM emails_by_user WHERE email_id = 123;
```

`email_id`와 `filename` 필드를 같이 사용하면 한 메일에 있는 모든 첨부파일을 질의할 수 있다.

질의 4) 읽은/읽지 않은 메일

```
// 읽은 메일 목록
SELECT * FROM emails_by_folder
WHERE user_id = '123'
AND folder_id = '234'
```

```
AND is_read = TRUE
ORDER BY email_id
```

NoSQL 데이터베이스는 파티션 키와 클러스터 키에 대한 질의만 허용하므로 **is_read는 질의할 수 없다.**

- 모든 메시지를 가져온 다음에 애플리케이션 단에서 필터링할 수 있다. (대규모 서비스에 부적합)
- 테이블을 비정규화하여 해결할 수 있다. (읽은 상태의 모든 이메일, 안 읽은 상태의 모든 이메일)

질의 보너스) 이메일 타래 가져오기

보통 JWZ 같은 알고리즘을 통해 구현한다. 기본적인 아이디어를 정리한다.

이메일 헤더에는 보통 세가지 필드가 있다.

- Message-Id: 메시지 식별자. 메시지를 보내는 클라이언트가 생성
- In-Reply_To: 이 메시지가 어떤 메시지에 대한 답신인지 나타내는 식별자
- References: 타래에 관계된 메시지 식별자 목록

이 필드들이 있으면 이메일 클라이언트는 타래 내 모든 메시지가 사전에 메모리에 있는 경우 전체 타래를 재구성할 수 있다.

일관성 문제

분산 데이터베이스는 `일관성`과 `가용성` 사이의 타협을 해야 한다.

이메일은 정확성이 매우 중요하므로 모든 메일함은 반드시 하나의 사본을 통해 서비스한다고 가정한다.

장애가 발생하면 클라이언트는 주 사본이 복원될 때까지 동기화/갱신을 완료할 수 없다.

일관성을 위해 가용성을 희생한다.

전송 가능성

- **전용 IP**: 대부분의 이메일 서비스 사업자는 아무 이력이 없는 새로운 IP 주소에서 온 메일을 무시한다.
- **범주화**: 범주가 다른 이메일은 다른 IP 주소를 통해 보내라. (광고는 다른 서버에서 ...)
- **발신인 평판**: 새로운 이메일 서버의 IP 주소는 사용 빈도를 서서히 올리는 것이 좋다. (스팸 분류 가능성)

- **스팸 발송자의 신속한 차단**: 스팸을 뿌리는 사용자는 서버 평판을 훼손하기 전 차단돼야 한다.
- **피드백 처리**: ISP 피드백을 쉽게 받아 처리하는 경로를 만들어야 한다.
  - 경성 반송: 수신인의 이메일 주소가 올바르지 않아 ISP가 전달을 거부
  - 연성 반송: ISP 측의 이메일 처리 자원 부족 등의 이유로 일시적 전달 불가
  - 불만 신고: 수신인이 스팸 신고 버튼을 누른 경우
- **이메일 인증**: SPF, DKIM, DMARC 등 피싱에 대응하는 보편적 전략이 있다.

검색

ElasticSearch

`user_id`를 파티션 키로 같은 사용자의 이메일은 같은 노드에 묶어 놓는다.

카프카를 활용하여 색인 작업을 시작하는 서비스와 실제로 색인을 수행할 서비스 사이의 결합도를 낮춘다.

주 이메일 저장소와 동기화를 맞추는 문제가 까다롭다.

맞춤형 검색 솔루션

자체적으로 검색 솔루션을 구현하는 경우 마주하게 될 주요 과제인 디스크 I/O 병목 문제를 다룬다.

- 매일 저장소에 추가되는 메타데이터와 첨부파일의 양은 페타바이트 수준이다.
- 색인을 구축하는 프로세스는 다량의 쓰기 연산을 발생시킬 수 밖에 없으므로 `LSM` 트리를 사용하여 디스크에 저장되는 색인을 구조화하는 것이 바람직한 전략이다.
- 쓰기 경로는 순차적 쓰기 연산만 수행하도록 최적화되어 있다.
- 자주 바뀌는 데이터를 그렇지 않은 데이터와 분리할 수 있다.

비교

소규모의 이메일 시스템을 구축한다면 ElasticSearch가 좋다.

그렇지 않으면 데이터베이스에 내장된 전용 검색 솔루션을 사용하는 것이 바람직할 수 있다.

규모 확장성

- 각 사용자의 데이터 접근 패턴은 다른 사용자와 무관하다.

→ 시스템의 대부분 컴포넌트는 수평적 규모 확장이 가능하다.

- 데이터를 여러 데이터센터에 다중화하여 가용성을 향상시켜야 한다.

  → 사용자는 네트워크 토폴로지 측면에서 자신과 물리적으로 가까운 메일 서버와 통신한다.

  → 장애 발생 시 사용자는 다른 데이터센터에 보관된 메시지를 이용한다.

▼ 4️⃣ 마무리

- 전통적인 이메일 서버는 어떻게 설계되어 있을까?
  - 왜 이 서버들은 현대적 사용 패턴을 충족하기 어려운가?
- 이메일 API
- 이메일을 보내고 받는 프로세스
- 메타데이터 데이터베이스
- 이메일 전송 가능성 문제
- 검색
- 규모 확장성 문제

더 논의하고 싶다면?

- 결함 내성: 장애가 발생하면 어떻게 할 것인가?
- 규정 준수: 각 나라에서 준수해야 할 법규가 있는 경우
- 보안: 민감한 정보가 포함된 이메일을 어떻게 제공할 것인가?
- 최적화: 같은 메일이 여러 수신자에게 전송되어 저장소에 동일한 데이터가 쌓이는 경우 저장 비용 최적화