24
Distributed Email Service

Email is one of the most complex and widely used internet services. In this chapter, we design a scalable email service that supports millions of users and billions of messages per day. We focus on the core flows, storage, and key trade-offs that make such a system reliable and secure.

Step 1 - Understand the Problem and Establish Design Scope

Email systems can mean many things to different people. For this chapter, we focus on a system that supports the core flows of sending, receiving, storing, and retrieving messages for millions of users.

Functional requirements

Sending: Users can send emails to one or more recipients, including CC and BCC.

Receiving: Users receive emails from other users and from external mail servers.

Storage: Emails must be stored reliably and durably, with support for folders or labels, metadata, and attachments.

Retrieval: Users can fetch their mail from multiple clients and devices, with consistent mailbox state.

Scale: The system must support millions of users and billions of messages per day.

Delivery guarantees: Messages should be delivered reliably with retries and eventual consistency. Duplicate delivery is acceptable if handled gracefully.

Non-functional requirements

Latency: Inbox loads and message retrieval should be fast enough for a responsive user experience.

Availability: The system should remain available even during partial failures, with graceful degradation rather than total outage.

Security and privacy: Support TLS for transport, authentication, spam and abuse protection, and access control. Protect user data at rest and in transit.

Back-of-the-envelope estimation

Assume 1 billion users, 10 billion messages per day, average message size of 50 KB, average attachment size of 500 KB, and 30 days of retention.

- Ingress: 10 billion messages × 50 KB ≈ 500 TB per day
- Attachment storage: 10 billion × 500 KB ≈ 5 PB per day
- Retention: 30 days ≈ 150 PB
- Metadata: 10 billion × 1 KB ≈ 10 TB per day
- Peak SMTP connections: 100 million concurrent connections
- Peak API requests: 100 million requests per second

Step 2 - Propose High-Level Design and Get Buy-In

Email is a store-and-forward system. Mail servers accept messages, queue them, retry when needed, and deliver them to the recipient's mailbox. We'll start with the protocols users and servers actually use, then sketch the core components that move and store mail.

Email protocols

SMTP: Simple Mail Transfer Protocol is the standard protocol for sending mail between servers and from clients to servers. It handles the envelope, recipients, and message transfer.

IMAP: Internet Message Access Protocol lets clients read and manage mail stored on a server. It supports folders, flags, and partial sync across devices.

POP3: Post Office Protocol version 3 is a simpler protocol that downloads mail to a single client. It's less flexible but still widely supported.

Traditional mail servers

Traditional mail servers store messages in local mailbox formats and deliver them directly between organizations. This worked well for small networks but scales poorly when you need high availability, spam filtering, and compliance.

Modern email architecture

Modern email systems separate ingress, storage, and retrieval into distinct services. This allows each component to scale independently and fail without taking the entire system down.

Core components

- Ingress: Accepts incoming mail from other servers and from clients, applies spam and policy checks, and queues for delivery.
- Storage: Persists mailboxes, folders, metadata, and attachments. Supports multi-region replication.
- Retrieval: Serves IMAP and API requests so clients can read, search, and sync mail.

Ingress

Ingress is the entry point for all incoming mail. It must accept connections from the internet, validate senders, filter spam and abuse, and queue messages for delivery. Ingress should be stateless and horizontally scalable so it can handle connection floods and DDoS attempts.

Key responsibilities

- Accept SMTP connections from other mail servers.
- Validate recipients and domains.
- Run spam, malware, and policy checks.
- Queue messages for durable delivery.
- Rate limit by sender, IP, and domain.

Storage

Storage must persist mailboxes, folders, metadata, and attachments reliably and durably. Most systems use a combination of a metadata store and blob storage, with multi-region replication for disaster recovery.

Key responsibilities

- Persist mailbox state and message metadata.
- Store attachments as blobs with deduplication.
- Replicate across regions for durability.
- Support quotas and retention policies.

Retrieval

Retrieval serves IMAP and API requests so clients can read, search, and sync mail. It must support multi-device access with consistent mailbox state.

Key responsibilities

- Serve IMAP connections for mail clients.
- Provide API access for web and mobile clients.
- Support search and filtering.
- Manage mailbox state across devices.

Step 3 - Deep Dive

Ingress: SMTP at scale

Ingress must accept connections from the internet, validate senders, and queue messages for delivery. Spam and abuse are the biggest challenges here. Most systems use layered checks before accepting mail.

Key mechanisms

- Connection limits per IP and per domain.
- Rate limits by sender and domain.
- Reputation checks against DNS blocklists.
- Spam and malware scanning.
- Policy checks for attachment types and sizes.
- Queue with retry and exponential backoff.

Storage: metadata and blobs

Most systems split storage into a metadata store and blob storage. Metadata holds mailboxes, folders, and message pointers. Blobs hold the actual message bodies and attachments.

Key design choices

- Metadata: Relational database or key-value store for mailbox state and message pointers.
- Blobs: Object storage for raw message bodies and attachments.
- Deduplication: Hash attachments to avoid storing duplicates.
- Quotas: Track per-user storage and enforce limits.
- Retention: Support TTL and legal hold policies.

Retrieval: IMAP and API

Retrieval serves mail clients that read, search, and sync mail. IMAP is the standard protocol for mail clients, and APIs support web and mobile apps.

Key operations

- Fetch mailbox state and message lists.
- Search by sender, recipient, subject, and date.
- Mark read/unread, flag, move, and delete.
- Push notifications for new mail.

Spam and abuse

Spam and abuse are the biggest challenges for any email system. Layered checks at ingress reject most bad mail before it enters the system.

Key mechanisms

- Connection limits per IP and per domain.
- Rate limits by sender and domain.
- Reputation checks against DNS blocklists.
- Spam and malware scanning.
- Policy checks for attachment types and sizes.
- Queue with retry and exponential backoff.

Delivery guarantees

Email delivery is store-and-forward with retries. Messages may be delayed but should not be lost. Duplicate delivery is acceptable if handled gracefully.

Key mechanisms

- Queue with durable storage.
- Retry with exponential backoff.
- Dead letter queue for permanent failures.
- Idempotent delivery to avoid duplicates.

Step 4 - Wrap Up

In this chapter, we proposed a scalable email service that supports millions of users and billions of messages per day. We covered the core protocols, ingress for accepting mail, storage for persisting messages, and retrieval for serving mail clients. We also discussed spam and abuse, delivery guarantees, and key trade-offs.

If time allows, we can discuss:

- Full-text search across mailboxes.
- Multi-region active-active deployment.
- Encryption at rest and in transit.
- Compliance and legal hold.
- Monitoring and alerting for delivery failures.

Chapter summary

References Materials

- RFC 5321 - Simple Mail Transfer Protocol
- RFC 5322 - Internet Message Format
- RFC 3501 - Internet Message Access Protocol
- RFC 1939 - Post Office Protocol - Version 3
- RFC 8314 - Cleartext Considerations and Implicit TLS for Email
- RFC 7208 - Sender Policy Framework (SPF)
- RFC 6376 - DomainKeys Identified Mail (DKIM) Signatures
- RFC 7489 - Domain-based Message Authentication, Reporting, and Conformance (DMARC)
- RFC 8461 - SMTP MTA Strict Transport Security (MTA-STS)
- RFC 8460 - SMTP TLS Reporting
- Amazon SES Developer Guide
- Google Workspace Admin Help - Email delivery basics
- Microsoft Exchange Server Architecture
- Postfix Architecture Overview
- Dovecot Architecture
- Stripe Engineering - How we scale our email infrastructure
- Cloudflare Blog - How we secure email at scale
- Mailgun Engineering - Lessons learned from building a mail platform
- Twilio SendGrid Engineering - Deliverability at scale
- Google Cloud Pub/Sub Documentation
- Amazon S3 Developer Guide
- Apache Kafka Documentation
- Redis Streams Documentation
- Prometheus Monitoring Documentation
- Grafana Dashboards Documentation