Engineer

Email

mail sent between mailboxes across the internet

What it is

Email is the system that carries a written message to an address of the form `name@domain` and leaves it in a mailbox until its owner reads it. It is also spelled e-mail, and the standards that define it mostly just say mail. Nothing about it requires the reader to be online when the message is sent: servers accept the message, hold it and pass it on, which is why it is described as store-and-forward rather than as a conversation.

Two documents do most of the defining. RFC 5322 says what a message is: a block of header fields, one blank line, then the body. RFC 5321 describes SMTP, the protocol a sending server uses to hand that message to the next server on the way. To find the next server, a sender looks up the domain’s mail exchanger records in DNS, which name the machines willing to accept mail for it.

A complete message can be very short:

```
From: Ana Holm <ana@example.org>
To: orders@example.com
Date: Fri, 16 Oct 2026 09:14:00 +0200
Subject: Invoice 114 has the wrong VAT number
Message-ID: <20261016.0914.ana@example.org>

Hello, could you send a corrected copy?
```

Only two of those header fields are compulsory: the date and the originator’s address. Everything else is optional by the letter of the standard, though a message without a subject or a message identifier looks careless to the programs that sort mail. Lines may run to 998 characters at most and should stay within 78, because plenty of mail programs still cut off or wrap anything longer.

There is a second, hidden layer. SMTP moves the message inside an *envelope*: a return address, where reports of failure go, and a list of recipients. The envelope and the `From:` and `To:` lines the reader sees are separate and often differ. A blind copy works because the recipient is on the envelope and absent from the headers; a mailing list works because the envelope's return address points at the list, so a dead subscriber's bounce reaches the list software instead of the person who wrote the post.

Use the concept in a decision

Put "What is email?" to work in a real decision. We can help relate the concept to the system you are building or buying. A site that is quick, findable and editable by your own staff, without a bill for every change of wording.

Explore our services Get in touch →

Test the assumptions behind the term

Use the explanation and linked work to frame questions for your project. We can examine the assumptions with you before a tool is chosen. Developed 100 web applications using HTML/HTML5 — 100 web applications built

Developed 100 web applications using HTML/HTML5 Get in touch →

Why it matters

An email address is the identity most of the web falls back on. Sign-up confirmations, password resets, receipts, invoices and the alert that a payment has bounced all travel by mail, so a product whose mail goes missing has users who are locked out without anyone noticing. The person who never received their confirmation link rarely writes to say so. They leave.

Email also has no owner. Any domain can run its own servers, and any server can talk to any other, and that openness is why it has outlasted almost every messaging service built since. The price is trust. The original design lets a sender write anything in the `From:` line, so receiving servers now decide for themselves whether to believe a message, and they decide from evidence the sending domain publishes in DNS: which hosts may send for it, which keys sign its mail, and what to do with a message that claims the domain and carries neither.

For a business the useful split is between two kinds of traffic. Transactional mail goes to one person because of something that person just did: a code, a receipt, a reset link. Bulk mail goes to many people because the sender chose to send it. They earn reputations separately, draw complaints at different rates and are often sent from different domains, so mixing them lets a newsletter that annoys people slow down the password reset.

Finally there is where mail lives. Running a mail server means storage, spam filtering, patching and a sending reputation that has to be earned and kept. A hosted mailbox for reading and a sending provider for automated mail is the usual answer, and it is a choice worth making on purpose rather than by default.

What it looks like in practice

On one platform mail carried verification, notifications, digests and campaigns, and a single provider made every outage total. The answer was to put three providers behind one interface with automatic failover: SendGrid first, then SMTP2GO and Azure Communication Services. Inbound webhooks report what each provider did with a message, and both sides land in a send log and a delivery event table, so the provider-reported delivery status can be queried instead of guessed. Campaigns to segments of users run on a separate broadcast schema of 5 tables and 24 functions, built apart from transactional mail because it is a different problem. A startup check sends a real message through the stack behind a flag; in development a failure logs a warning, and in staging and production it stops the process. The page names the cost too: three integrations to keep working, and delivery logs that grow and have to be pruned.

The company's own mail starts in DNS. Its zone holds about twenty records, among them mail routing through a hosted mailbox provider, a transactional sending domain and the authentication records for both, and the whole zone is kept as data in the repository and reconciled against the provider's API. An audit entry point reports the difference between the declaration and what is being served without changing anything, and it is the part used most.

Where the company's own mail lives was settled by arithmetic. Its public services run on one host with 512 MB of memory, and mail is kept off that machine on purpose: a mail server needs headroom the host has already spent, so it is written as an install scaffold waiting for a host of its own. Meanwhile letters to the company are read in its own mailbox, hosted with Apple iCloud, and there is no mailing list behind it.

Where it goes wrong

Mail starts landing in spam the week a provider changes. Nothing in the application changed, yet receivers grow suspicious. The domain's published records still name the old sender, so mail from the new one arrives unsigned or

from a host the domain never authorised. Add the new sender to the records before the switch, keep the old one until its last message has gone, and, if the domain asks for them, read the aggregate reports receivers send back.

The application boots, and nobody receives anything. An expired key or a suspended sending account looks exactly like a working configuration until a real person waits for a code. Send a real message at startup wherever silence is dangerous, and record what the provider says happened to each message, so that a missing mail can be traced rather than argued about.

Bounces pile up at an address nobody reads. The envelope's return address points at a `no-reply` box, so permanent failures arrive and stay unopened. The sender keeps writing to mailboxes that closed a year ago, and receivers read that as the habit of a spammer. Point the return address at something that processes reports, and stop mailing an address after a permanent failure.

A temporary refusal gets treated as final. SMTP replies starting with 4 mean try again later; replies starting with 5 mean stop. A sender that gives up on the first 4xx loses mail that a busy or cautious server would have accepted half an hour on. RFC 5321 asks for retries at least 30 minutes apart and a give-up time of at least four to five days; transactional mail usually wants quicker retries with backoff, and a clear record when it finally stops trying.

Further reading

- RFC 5322: Internet Message Format — the header fields, the blank line, the line-length limits and the two fields every message must carry.
- RFC 5321: Simple Mail Transfer Protocol — the envelope, the reply codes, mail exchanger lookup and the advice on retrying.
- RFC 7208: Sender Policy Framework (SPF) — how a domain lists the hosts allowed to send mail in its name.
- RFC 6376: DomainKeys Identified Mail (DKIM) Signatures — signing headers and body with a key the receiver fetches from DNS.
- RFC 7489: DMARC — tying both checks to the visible `From:` domain, with a policy and reports; an informational document rather than a standard.

Glossary

Choose for your actual constraints

Describe the workflow, the existing interfaces and the users who need the change. We can define a testable delivery and a handover that lets your team maintain it.

See what it costs Get in touch →

Make the terminology practical

Ask how “What is email?” applies to your organisation. A concrete example from your current system is enough to start the discussion.

Explore our services

welcome@engineer.company +45 50 10 61 84 Copenhagen, Denmark

Take this page with you

Contact card

Carefully Crafted With Love by D.

CVR 45969703 · Tagensvej 105C, 3. th, 2200 København N

Copyright © 2025 – present · Engineer ApS · All rights reserved.

Printed from https://engineer.company/glossary/email/