Engineer

macOS

Apple's desktop operating system

What it is

macOS is the operating system Apple ships on every Mac: the desktop and the Finder, the bundled apps, and underneath them a Unix system with a terminal, a shell and the familiar command-line tools. Apple builds it for its own hardware and sells the two together, so in practice choosing macOS means choosing a Mac.

The core is Darwin, whose source Apple publishes. Its kernel, XNU, is a hybrid that joins the Mach kernel from Carnegie Mellon University with code taken from FreeBSD and with IOKit, a C++ framework for drivers; the name stands for “X is Not Unix”. Much of the userland comes from FreeBSD too, and the manual pages for `sed` and `date` on a Mac describe FreeBSD’s versions of those tools. Since macOS 10.15 Catalina the default shell has been zsh.

The hardware changed under it recently. In late 2020 Apple began moving the Mac from Intel processors to its own chips, Apple silicon, and every Mac line from the MacBook Air to the Mac Pro now has Apple silicon models. Those chips are arm64, which matters the moment software built on a Mac has to run on an x86-64 server.

On laptops and desktops macOS is weighed against Windows and Linux.

Use the concept in a decision

Put “What is macOS?” to work in a real decision. We can help relate the concept to the system you are building or buying. The laptop, the printer, the login and the mailbox, sorted quickly and in plain language, without anyone feeling foolish.

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. Established a Technical Support department — 10,000 clients served by a support department built from scratch

Established a Technical Support department Get in touch →

Why it matters

For a company that writes software, macOS plays two parts that are easy to blur: it is the machine many developers work on, and it is a platform customers run software on.

As a workstation, a Mac differs from the Linux servers it deploys to in three places. Its command-line tools come from FreeBSD rather than GNU, so the same flag can mean different things in a script. Its processor is arm64 where the server may be x86-64. And it has no Linux kernel, while Linux containers share the kernel of their host, so tools running them on a Mac start a Linux virtual machine first; Apple's own `container` tool, which needs Apple silicon and macOS 26, gives each container a lightweight virtual machine of its own. None of this rules a Mac out; it means testing on the system you deploy to rather than trusting a green run on a laptop.

As a platform, macOS raises the question of how to build for it. A cross-platform shell, one codebase with an embedded browser engine shipped for Windows, Linux and the Mac, keeps several systems in step cheaply, but on the Mac it carries a browser runtime a native application has no use for and draws its own controls instead of the system's. A native application in Swift and SwiftUI uses the system's controls and materials, and follows the user's settings for appearance, accent colour, contrast, text size and reduced motion without extra code. The price is a separate client and people who know the platform. Native pays off when Mac users are the customers.

Shipping to Macs also passes through a gate. Software distributed outside the Mac App Store is signed with a Developer ID certificate and sent to Apple's notary service, an automated scan for malicious content and code-signing problems that is separate from App Review. Since macOS 10.15, all software built after 1 June 2019 and distributed with Developer ID must be notarised. Apple returns a ticket to staple to the software and publishes the same ticket online, where Gatekeeper looks for it when the user first opens the app. The certificate comes with the Apple Developer Program, at $99 a year. Plan the membership and the signing step into the first release.

What it looks like in practice

The desktop experience of one messaging product on macOS was a cross-platform shell that neither looked nor behaved like a Mac application. Its well-tested Rust core exposes a C interface, and Swift can call C, so a native client in Swift and SwiftUI was built directly on that interface. The core is linked in as a static archive from a pinned upstream revision, which keeps the build reproducible, and a thin wrapper owns every pointer that crosses between the languages. The client starts as a Mac application, uses the platform's own materials and controls, and carries no embedded browser. The page is frank about the gap: there is no notarised distribution and no update channel, so it is a working application rather than a product other people run. The notary step above is what stands between the two.

The platform's look can be enforced rather than imitated. A design system of 48 named tokens in six groups builds colour and material on the platform's semantic colours and glass material instead of fixed values, so the interface follows the system appearance, the accent colour and the contrast settings with no code watching for them, and its text styles scale with the user's size preference. A linter rejects a raw number where a token belongs, a hardcoded font size, and an animation that ignores the reduce-motion preference. The page names its limit too: tokens make an interface consistent, which is different from making it good.

Compressed memory and swap let a macOS app grow far past the machine's physical memory. One incident record documents diagnostic reports of 111 GB of compressed pages on a 36 GB machine on 2026-09-01, and a separate footprint sample showing growth of about 41 MB per second during a connection retry loop. The event path retained core log events in an unbounded main stream and copied them to subscribers that did not use those events. Buffers were bounded, subscriptions filtered by event type and logging given a rate budget. A full test-run comparison recorded 610,996 core log lines falling to 1,411; these are test-run measurements, not a measured post-fix memory figure or a production benchmark. The trade is explicit: repetitive diagnostic events may be dropped, and affected state can be refreshed instead of retaining unlimited events. Separate shutdown timeouts needed a separate fix.

Where it goes wrong

Activity Monitor shows almost no memory to spare on a Mac that runs perfectly well. macOS caches recently used files in otherwise unused memory and, as memory fills, compresses inactive apps to make room for active ones, so

Memory Used close to Physical Memory and a large Compressed figure are normal. Apple's own guidance is that free memory does not necessarily make a Mac faster. Read the Memory Pressure graph instead: green means the RAM is used efficiently, yellow that the Mac might eventually need more, red that it needs more. Swap is the plainest signal:

```
sysctl vm.swapusage
vm_stat | grep -E '^Swap(ins|outs)'
```

The `vm_stat` counters run from boot, so compare two readings a minute apart; swapouts that keep climbing mean the work has outgrown the RAM.

A script that works on the Mac stops with an odd error on the Linux server, or the other way round. The FreeBSD-derived `sed` on a Mac treats the word after `-i` as a backup suffix, so the GNU habit `sed -i 's/a/b/' file` takes the expression as the suffix and the file name as the script. The Mac habit `sed -i '' …` makes GNU `sed` read the expression as a file name. A suffix attached to the flag works on both:

```
sed -i.bak 's/^port=80$/port=8080/' app.conf && rm app.conf.bak
```

The lasting fix is to run scripts in CI on the system they will meet in production.

A build shared outside the team is blocked by Gatekeeper when the recipient opens it. The app was signed for development, or signed but never notarised. Sign it with a Developer ID certificate, with the hardened runtime enabled and a secure timestamp, send it to the notary service, and staple the ticket so Gatekeeper finds it even offline. A ZIP archive can be notarised but cannot hold a ticket, so staple the app inside it:

```
ditto -c -k --keepParent App.app App.zip
xcrun notarytool submit App.zip --keychain-profile notary --wait
xcrun stapler staple App.app
```

A container image runs on the developer's Mac and dies on the server with `exec format error` **.** Docker builds for the platform of the BuildKit daemon by default, and on an Apple silicon Mac that daemon runs in an arm64 virtual machine. A container shares the host's kernel, so without emulation an x86-64 host runs x86-64 code only. Check the server with `uname -m` , then build for it with `docker build --platform linux/amd64` , or build in CI on a machine with the server's CPU type.

Further reading

- Mac User Guide — Apple's own guide to macOS, kept for each current release.
- Check if your Mac needs more RAM — the Memory Pressure graph and what its three colours mean.
- Notarizing macOS software before distribution — Developer ID, the hardened runtime and how Gatekeeper uses the ticket.
- Multi-platform builds — why arm64 and amd64 hosts need different images, and how one build makes both.
- container — Apple's tool for running Linux containers on a Mac, each in a lightweight virtual machine.
- XNU — the macOS kernel's source, with a short account of how Mach, FreeBSD and IOKit fit together.



Glossary

Choose for your actual constraints

Describe the device or service, the observed symptoms and the data that must be preserved. We can establish the safest useful diagnostic step.

See what it costs Get in touch →

Make the terminology practical

Ask how “What is macOS?” 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/macos/