Caddy in Front of FastAPI: HTTPS, HTTP/3, and Zero Downtime

Tera Byte 26 ⋮⋮ 10/8/2025

Write A Catalyst and Build it into Existence.

Member-only story

Generated by sora.chatgpt.com

Why pair Caddy with FastAPI

FastAPI shines at application logic and data validation, while Caddy is brilliant at serving the public edge with automatic certificates, quick configuration, and modern transport features. Put Caddy in front and your FastAPI service gains effortless HTTPS, clean routing, and HTTP 3 without extra plugins. This split of concerns keeps the Python app simple and lets the web server handle TLS, compression, caching hints, and graceful restarts. The result is a nimble stack that feels fast to users and friendly to operate.

The high level layout

A common layout keeps Caddy as the only internet facing container, proxying to one or more FastAPI processes that listen on an internal network. Clients connect via HTTPS on port 443, Caddy terminates TLS and forwards to the app at something like http colon slash slash app colon 8000. Health checks and metrics live on distinct paths so that operations tooling can monitor without touching the public routes. Static media can be served directly by Caddy for maximum throughput and fewer Python cycles.

Building the app image

Use a small base image to keep startup quick. If you like modern Python tooling, the uv project from Astral can install and run dependencies quickly with a lockfile for repeatability. For example, copy pyproject and uv lock, run uv pip install to a virtual environment, then copy your FastAPI code. Keep the runtime slice thin by avoiding build tools and caches. The app command can be uv run uvicorn myapp colon app with workers and bind address tuned for your environment.

Choosing a process manager

For simple services uvicorn alone is fine. For heavier traffic consider gunicorn as a supervisor that spawns multiple uvicorn workers. Gunicorn adds preloading, graceful worker recycling, and tight control over concurrency. A balanced starting point is one worker per CPU core with uvicorn workers set to asyncio and keepalive tuned to match Caddy upstream settings. Test under load, then adjust worker counts and timeouts until tail latencies look healthy.

Writing the Caddyfile

Caddy configuration reads almost like prose. A minimal site block declares your domain, enables automatic TLS, and defines a reverse proxy to the FastAPI upstream. Add header directives for HSTS, content type sniffing protection, and compression. If you serve WebSocket endpoints, Caddy handles them automatically without extra flags. For static directories, mount a route that serves files from a volume. Keep logs in JSON so observability tools can parse them, and tag entries with the service name to make cross service searches easier.

Enabling HTTPS and HTTP 3

Caddy obtains and renews certificates from trusted authorities with zero ceremony. Point the A record to your server and it will fetch certificates on first request, then renew them before expiry. To enable HTTP 3 you only need to open UDP port 443 and ensure your platform allows it. Caddy will negotiate QUIC transparently, which reduces handshake latency on mobile networks and improves resilience in packet loss. Your app code does not change at all; the proxy translates protocols while preserving headers like X Forwarded For and X Forwarded Proto.

Containers and networking

In Docker compose, place Caddy and the app in the same user defined network so the proxy can reach app colon 8000 by name. Expose only Caddy ports to the host. Mount a volume for Caddy data so certificate state survives restarts. Keep the app container stateless, then scale replicas using the compose scale flag or an orchestrator such as Swarm or Kubernetes. Each replica registers as an upstream, and Caddy can round robin by default. For sticky sessions, route based on a cookie or shift state to Redis so any replica can serve any request.

Zero downtime deploys

Aim for graceful handoffs at both layers. Configure gunicorn to use graceful timeouts so it finishes in flight requests before exiting. On deploy, start new containers, wait for their health checks to pass, then remove old ones. Caddy detects that upstreams are healthy and begins routing to them without tearing existing connections. If you run Caddy as a container, reload the config rather than restarting the process. Version your Caddyfile alongside app code so each release has the right routes and headers. With this flow, users never see a blip, even during schema migrations that are backwards compatible for a brief window.

Health checks and readiness

Expose a lightweight route such as slash health that returns quickly without touching databases. Add a deeper slash ready route that verifies downstream connectivity and migrations are applied. Wire the compose or orchestrator health checks to slash health for liveness and to slash ready for readiness. Caddy can use active health checks for upstreams, removing a replica from rotation if it fails, then re adding it when the route recovers. This keeps spikes and cold starts from leaking into user traffic.

CI CD that respects the edge

In a CI CD pipeline, build the app image, run tests, and publish with a unique tag. Scan for vulnerabilities, then push a signed image. Next, validate the Caddyfile using caddy validate to catch mistakes before rollout. For staging, enable a separate site block with a staging domain and a synthetic origin header so the app can point to staging databases. Promotion becomes a config switch and a rolling update, not a rebuild. Keep environment secrets in your platform store, inject them at runtime, and never bake them into images.

Observability from day one

Collect logs, metrics, and traces that tell a coherent story. Caddy can emit access logs that include request id, upstream duration, and status code. FastAPI can instrument with OpenTelemetry, exporting traces to a collector. Add Prometheus metrics for request counts and latency buckets, plus process metrics like CPU and memory. With these pieces connected, a spike in p95 latency in the app can be correlated with longer upstream times in Caddy, and the trace will show where time was spent, whether in Python code, database calls, or network waits.

Hard won tips

Keep timeouts consistent between layers so requests do not die early in one place and linger elsewhere. Use proxy protocol or real IP headers only if needed, and confirm that your framework reads them correctly. Validate that HTTP 3 is actually negotiated by checking response alt svc headers and your browser dev tools. For static files, let Caddy set caching headers and ETags so clients cache aggressively while still updating when content changes. When in doubt, load test with a mix of short and long requests to see how concurrency settings behave under pressure.

Closing thoughts

Caddy and FastAPI form a compact and modern stack that delivers secure transport, speedy negotiation, and clean deployments without drama. With docker for packaging, uv or pip for dependable builds, gunicorn for worker management, and a thoughtful CI CD path, you can ship quickly and keep shipping without interruptions. Add observability from the start, practice safe migrations, and let Caddy handle the web so your FastAPI app can focus on the product. The outcome feels simple to maintain and delightful to use, which is exactly what a production service should be.

Want the poster version of this guide? Click the image to view it full size and use it as a quick checklist at your desk.