DocuSeal: Self-Hosted E-Signatures That Never Leave Your Server

An open-source DocuSign alternative you run yourself, for the price of a small VPS

Every e-signature vendor charges you per envelope, and every one of them holds your signed documents on their cloud. For a solo or small firm, that means a monthly bill that scales with your caseload and a copy of every executed engagement letter, fee agreement, and settlement sitting on a server you do not control.

DocuSeal is the open-source fix. It is a document-signing platform you run yourself, in a Docker container, on hardware you own or a cheap VPS. Clients still sign from a browser link like they would with DocuSign. The difference is that the template, the signer's data, and the final signed PDF live on your box, and there is no per-signature fee.

I run it in production for firm fee agreements. Here is how to stand one up.

What it actually is

DocuSeal is a Ruby on Rails app that ships as a single Docker image. You give it a document, drop signature and date fields onto it in the browser, and it becomes a reusable template. Send that template to a client, they sign in their browser, and DocuSeal builds the executed PDF plus an audit trail (who signed, when, from what IP).

It is a straight DocuSign or HelloSign replacement for the 90 percent case: engagement letters, fee agreements, consents, releases. It does not try to be a full practice-management system, and that is a feature.

The 60-second version, to see it work

Before you wire up anything permanent, prove it runs. This uses a built-in SQLite database and is fine for kicking the tires, not for real client data.

```
docker run --name docuseal -p 3000:3000 \
  -v $(pwd)/docuseal:/data \
  docuseal/docuseal:latest
```

Open `http://YOUR-SERVER-IP:3000` in a browser. You will get a setup screen. Create your admin account, and you land on the Templates page. That is the whole product. Upload a PDF, add a signature field, done.

When you are satisfied it works, stop it (`docker rm -f docuseal`) and do the real install below.

The real install: Postgres and HTTPS

For anything holding client data you want two things the quick start skips: a proper Postgres database and HTTPS on your own domain. DocuSeal's own Compose file gives you both, and bundles Caddy as a reverse proxy that fetches a Let's Encrypt certificate automatically.

Grab the official Compose file and start it under your domain:

```
# Point an A record for sign.yourfirm.com at this server FIRST,
# so Caddy can issue the SSL cert on startup.
curl https://raw.githubusercontent.com/docusealco/docuseal/master/docker-
compose.yml \
  > docker-compose.yml

sudo HOST=sign.yourfirm.com docker compose up -d
```

That single command brings up three containers: DocuSeal, a Postgres database, and Caddy handling TLS. The `HOST` variable does double duty; it tells Caddy which domain to get a certificate for, and it tells DocuSeal to force HTTPS and build correct links in signing emails.

Two things to change in that Compose file before you trust it with real matters:

```
environment:
  - FORCE_SSL=${HOST}
  - DATABASE_URL=postgresql://postgres:CHANGE-THIS-PASSWORD@postgres:5432/
docuseal
  # ...and set the matching POSTGRES_PASSWORD in the postgres service to the
same value
```

The database password ships as a placeholder. Generate a real one (32+ characters, no shell-breaking symbols), set it in both the `DATABASE_URL` and the `POSTGRES_PASSWORD` line so they match, and never commit that file to a public repo. If you already run a Postgres instance, point `DATABASE_URL` at it and delete the bundled `postgres` service; DocuSeal creates its own schema on first boot.

First-run setup

Load `https://sign.yourfirm.com` and DocuSeal walks you through it:

1. **Create the admin account.** This is the firm's owner login. Use a real password manager entry.

2. **Set up email (SMTP).** Signing invitations go out as email, so DocuSeal needs a mail server. Add your SMTP settings as environment variables:

```
- SMTP_ADDRESS=smtp.yourprovider.com
- SMTP_PORT=587
- SMTP_USERNAME=notifications@yourfirm.com
- SMTP_PASSWORD=your-smtp-password
- SMTP_DOMAIN=yourfirm.com
```

1. **Build your first template.** Upload a PDF (say, your fee agreement), then drag signature, initials, date, and text fields onto it and assign each to a signer role. Save it. That template is now reusable for every client.

To send one: open the template, click to create a submission, type the client's name and email, and DocuSeal emails them a signing link. When they finish, you get the executed PDF back with its audit trail attached.

Yes, there is an API

The web interface is enough to run a firm by hand. The reason to self-host, though, is that DocuSeal exposes a full REST API, so your intake system or matter manager can fire off a signing request without anyone clicking through the DocuSeal console.

Grab an API key under Settings, then create a submission from a template with one call:

```
curl https://sign.yourfirm.com/api/submissions \
  -H "X-Auth-Token: YOUR-API-KEY" \
  -H "Content-Type: application/json" \
  -d '{
    "template_id": 1,
    "send_email": true,
    "submitters": [
      { "role": "Client", "email": "client@example.com", "name": "Jane
Client" }
    ]
  }'
```

That prefills the template with the client's details and emails them the link, all triggered from your own software.

The other half is the **webhook**. In Settings, point DocuSeal's webhook at a URL your system controls. When the client finishes signing, DocuSeal POSTs a `form.completed` event to that URL with the submission data and a link to the signed PDF. Your side catches it, marks the matter's fee agreement as executed, and files the PDF automatically. Create-by-API, confirm-by-webhook is the pattern that turns a signing tool into an actual pipeline.

> **Using AI to wire this up:** Hand Claude your intake app's stack and a sample DocuSeal webhook payload, then ask it to write the endpoint that verifies the event, pulls the signed PDF, and files it against the right matter. Describe your matter model and let it draft the handler; you review and deploy.

The deeper wiring (verifying the webhook, storing the signed PDF against the right matter, handling declines and reminders) is in the Founders walkthrough.

The part the vendor pitch skips: privilege and the audit trail

Two lawyer-specific notes the generic tutorials will not give you.

First, self-hosting is a confidentiality posture, not just a cost play. When you run DocuSeal yourself, the executed documents and every scrap of client PII stay on infrastructure you control and can lock down. That is a meaningfully different answer to give a client, or a bar examiner, than "a third-party vendor has it."

Second, a signature graphic is not the point; the audit trail is. A signed contract without the record of who signed it, when, and from where is a Word document with a picture pasted on it. DocuSeal generates that trail automatically. Back it up as carefully as the PDF itself, because in a fee dispute the trail is the evidence.

Standard caveat: this is general technical guidance, not a security audit of your specific setup. If you are storing privileged client documents, treat the server like any other place that data lives and lock it down accordingly.

Back it up, or you do not have it

Two things must be backed up together: the Postgres database and the data volume. Miss either and you have half a system.

```
# 1. Database dump
docker exec docuseal-postgres-1 pg_dump -U postgres docuseal \
  | gzip > /backup/docuseal/db-$(date +%F).sql.gz

# 2. The data volume (uploaded files, signed PDFs)
rsync -a --delete ./docuseal/ /backup/docuseal/data/
```

Then pull that backup off-site. A backup that lives on the same server is a copy, not a backup. Once a quarter, restore it to a throwaway box and confirm a signed PDF opens with its audit trail intact. An untested backup is a guess.

The playbook

```
# Stand it up (DNS pointed first)
curl https://raw.githubusercontent.com/docusealco/docuseal/master/docker-
compose.yml > docker-compose.yml
# set a real DB password in both places, then:
sudo HOST=sign.yourfirm.com docker compose up -d

# Then, in the browser at https://sign.yourfirm.com
#   create admin  ->  add SMTP  ->  build a template  ->  send a submission
```

Stand up the container, put it behind your own domain with HTTPS, wire the API and webhook into your intake, and back up the database and files together. That is a firm's entire e-signature stack, running on a server you own, with no per-signature invoice.