GUIDE_HOSTING.md

This guide explains how to deploy **Workforce-agent** end-to-end:

- **Frontend**: `frontend/` on **Vercel**
- **Backend**: `backend/` on **AWS EC2** (Uvicorn behind Nginx)
- **Database**: **Supabase Postgres** (with `pgvector`)
- **Domains** (recommended):
  - Frontend: `https://${FRONTEND_DOMAIN}`
  - Backend API: `https://${API_DOMAIN}`

This repo expects the **backend to load env vars from the repo root `.env`** (see `backend/core/config.py`).

Last reviewed / expanded: **Dec 2025**.

Deployment variables (set once, then copy/paste commands)

This guide avoids vague placeholders inside command blocks. Instead:

1. Set these variables in your terminal.
2. Copy/paste the commands exactly.

On your local machine (and also on EC2 when needed), run:

```
export DOMAIN="yourdomain.com"
export FRONTEND_DOMAIN="www.${DOMAIN}"
export API_DOMAIN="api.${DOMAIN}"

# EC2 access (greenfield)
export SSH_KEY_PATH=""
export EC2_PUBLIC_IP=""

# Git repo to deploy (set once)
export REPO_URL=""

# On EC2 we deploy the repo here:
export APP_USER="workforce"
export APP_HOME="/home/${APP_USER}"
export APP_DIR="${APP_HOME}/Agent-Hosting"

# Used for streaming verification later
export PROJECT_ID=""

# Used for rollback later (set it to a commit hash from `git log`)
export KNOWN_GOOD_COMMIT=""
```

Why:

- Your commands become deterministic.
- You only change values in one place.

Compatibility matrix (known-good + recommended)

This repo currently pins runtime dependencies in:

- `backend/requirements.txt`
- `frontend/package.json`

The below is a *practical* compatibility baseline for production hosting.

Backend host OS (recommended)

- **Amazon Linux 2023**
  - Use this guide's Amazon Linux steps end-to-end.
  - Certbot is installed via `/opt/certbot` virtualenv.
  - SELinux may require enabling `httpd_can_network_connect` for Nginx proxying.

Backend host OS (alternate)

- **Ubuntu Server 24.04 LTS (Noble)**
  - Default Python: **3.12**
  - Certbot via snap

Node / frontend build

- Vercel can build Vite apps directly.
- If you want consistent builds, set Vercel Project → Settings → General → **Node.js Version** to an LTS line.
  - As of late 2025, Node **v22** is an LTS line (Node.js release blog).

TLS / certificates

- Amazon Linux 2023: install `certbot` via a dedicated virtualenv at `/opt/certbot`.
- Ubuntu 24.04: use **Let's Encrypt** via **Certbot Snap**.

Table of contents

- 0) Repository layout
- 1) Target domains (recommended)
- 2) Supabase (Database) setup
- 3) Backend on AWS EC2 (or any Linux VPS)

- 4) Nginx reverse proxy (HTTPS + streaming + websockets)
- 5) Frontend on Vercel
- 6) DNS (connect domains)
- 7) End-to-end verification checklist
- 8) Troubleshooting
- 9) Operations (updates, logs, backups, rotations)
- 10) Cost control (AWS) and teardown
- 11) Security hardening checklist

---

0) Repository layout

- `frontend/` (Vite + React)
- `backend/` (FastAPI + SQLAlchemy + RAG)
- Root `.env` (used by backend)

---

1) Target domains (recommended)

Use one domain with a dedicated subdomain per service:

- **Frontend**: `${FRONTEND_DOMAIN}` (Vercel)
- **Backend API**: `${API_DOMAIN}` (EC2 + Nginx)

Why:

- Cookies can be shared across subdomains with `COOKIE_DOMAIN=.${DOMAIN}`.
- OAuth redirect URIs are stable.
- CORS config stays simple.

---

2) Supabase (Database) setup

2.1 Create Supabase project

- Create a Supabase project.
- In the Supabase dashboard, copy the **Postgres connection string**.
- You will paste that value into `DATABASE_URL` on EC2.

Notes:

- Prefer `sslmode=require` if your driver + environment supports it. If you run into SSL verification issues, explicitly configure CA certs rather than weakening SSL.
- If you use Supabase connection pooling (PgBouncer), confirm SQLAlchemy + your pool mode. Transaction pooling can break features that require session state.

Critical compatibility note (Supabase + SQLAlchemy, Oct 2025 Supabase guidance):

- For **long-running VMs/containers** (like EC2), Supabase recommends using the **direct connection string**.
- Supabase notes the direct connection hostname may resolve to **IPv6**, which will fail in IPv4-only environments.

Verify your server has IPv6:

```
curl -6 https://ifconfig.co/ip
```

If IPv6 is not supported in your deployment environment, consider:

- Using Supabase pooler in **session** mode, or
- Enabling Supabase's IPv4 add-on (plan-dependent)

If you do use the **transaction pooler** (recommended for serverless, not typical VMs), Supabase recommends SQLAlchemy `NullPool`.

2.2 Enable pgvector

In Supabase SQL editor:

```
create extension if not exists vector;
```

Verification:

```
select extname, extversion from pg_extension where extname = 'vector';
```

Enable via dashboard (Supabase docs):

- Database → Extensions → enable `vector`

2.3 Schema/migrations

This backend initializes tables on startup (see logs like "Initializing database schema").

If you use Alembic migrations in your workflow, run them in the backend environment before starting the service.

If schema is created dynamically by the app:

- Ensure the database user has privileges to create tables/indexes.
- Watch startup logs carefully on first boot.

---

3) Backend on AWS EC2

This section is written as a production runbook. If you follow it end-to-end, you should have:

- A hardened server with a stable public IP
- A systemd-managed backend process
- Nginx reverse proxy with correct streaming + WebSocket support
- Valid TLS via Let's Encrypt

3.1 EC2 instance + security group

Recommended for this repo:

- **Instance type**: `m7i-flex.large`
  - AWS naming uses a dot: `m7i-flex.large` (not `m7i-flex-large`).
  - Specs per AWS instance type page: **2 vCPU / 8 GiB RAM**
  - EBS-only
- **Disk**: 40–80 GB gp3

Security group inbound:

- **22**: SSH (your IP only)
- **80**: HTTP (0.0.0.0/0)
- **443**: HTTPS (0.0.0.0/0)

Strong recommendations:

- Use an **Elastic IP** (static) so DNS never breaks on reboot.
- Use Amazon Linux 2023 unless you have a strong reason to use another distro.
- Enable instance-level monitoring/alerts (CPU, memory, disk) before going live.

3.1.1 Required AWS setup (what you must configure)

You cannot fully automate instance creation without AWS CLI + IAM setup, so do this in the AWS Console:

- **EC2 Instance**:
  - OS: Amazon Linux 2023
  - Security Group inbound:
    - 22/tcp from your IP
    - 80/tcp from 0.0.0.0/0
    - 443/tcp from 0.0.0.0/0
- **Elastic IP**:
  - Allocate
  - Associate to your instance

Why:

- Elastic IP prevents outages when the instance reboots.
- Only 80/443 are public. Uvicorn stays private on 127.0.0.1.

3.1.2 First SSH login (greenfield)

On first connect you will typically SSH as `ec2-user`, then switch to the dedicated `workforce` user created later.

On your local machine:

```
chmod 400 "${SSH_KEY_PATH}"
ssh -i "${SSH_KEY_PATH}" ec2-user@"${EC2_PUBLIC_IP}"
```

Why:

- You do not SSH directly as the app user.
- `chmod 400` is required or SSH will refuse to use the key.

3.1.3 Point DNS to EC2

At your DNS provider:

- Create an `A` record `api` → your Elastic IP

Verify DNS from your local machine:

```
dig +short "${API_DOMAIN}"
```

3.1.4 Instance sizing notes

- If you still have `sentence-transformers` installed, first boot can be slow (model downloads). Once you remove it, boot is faster.
- If you expect many concurrent users or large RAG sources, plan for:
  - More RAM (vector search + caching)
  - More CPU (chunking + embeddings)

3.1.5 Required ports

- Public: `80`, `443`
- Private only: `8000` (Uvicorn) should NOT be exposed publicly

3.1.6 Server prerequisites (before installing the app)

- SSH access confirmed
- DNS A record for `${API_DOMAIN}` already pointing to the server (or will be set shortly)
- You can run `sudo` commands
- You have all third-party credentials ready (OpenAI, Google OAuth, Slack, Notion, Supabase)

3.2 Install system packages

Below are explicit install steps for the two most common targets.

Option A (recommended): Amazon Linux 2023

```
sudo dnf update -y

# Core tooling
sudo dnf install -y git curl ca-certificates nginx jq unzip

# Python
sudo dnf install -y python3 python3-pip

# Headers/tooling that prevent many pip build failures
sudo dnf install -y python3-devel openssl-devel libffi-devel

# Build tooling (some wheels may still build from source)
sudo dnf install -y gcc gcc-c++ make

# Postgres client for DB connectivity tests
sudo dnf install -y postgresql
```

Enable and start Nginx:

```
sudo systemctl enable --now nginx
sudo systemctl status nginx --no-pager -l
```

Install Certbot (Amazon Linux 2023, recommended):

```
sudo dnf install -y augeas-libs
sudo python3 -m venv /opt/certbot
sudo /opt/certbot/bin/pip install --upgrade pip
sudo /opt/certbot/bin/pip install certbot certbot-nginx
sudo ln -sf /opt/certbot/bin/certbot /usr/bin/certbot
certbot --version
```

Verify versions:

```
python3 --version
nginx -v
```

Firewall (optional, EC2 security group is still the primary firewall):

```
sudo systemctl enable --now firewalld || true
sudo firewall-cmd --permanent --add-service=http || true
sudo firewall-cmd --permanent --add-service=https || true
sudo firewall-cmd --reload || true
sudo firewall-cmd --list-all || true
```

Option B (alternate): Ubuntu 24.04 LTS

```
sudo apt-get update
sudo apt-get -y upgrade
```

```
# Core tooling
sudo apt-get install -y \
  git curl ca-certificates \
  python3 python3-venv python3-pip \
  build-essential \
  nginx

# Optional but commonly needed:
# - libpq-dev: useful if you ever switch from psycopg2-binary to psycopg (source build)
# - unzip/jq: troubleshooting and scripts
sudo apt-get install -y libpq-dev unzip jq

# Needed for Certbot snap on many servers
sudo apt-get install -y snapd

# Enable snapd (needed for certbot via snap)
sudo systemctl enable --now snapd
sudo systemctl status snapd --no-pager
```

Verify versions:

```
python3 --version
nginx -v
```

Optional (recommended) firewall (Ubuntu):

```
sudo ufw allow OpenSSH
sudo ufw allow 80/tcp
sudo ufw allow 443/tcp
sudo ufw enable
sudo ufw status
```

3.2.1 Create a dedicated app user (recommended)

Running as a dedicated user reduces blast radius.

```
sudo useradd --create-home --shell /bin/bash workforce
```

Decide where to deploy the repo. In this guide we use:

- `${APP_DIR}`

Switch into that user:

```
sudo -iu workforce
```

3.2.2 (Optional but recommended) Swap file

Small instances can OOM during dependency installs.

2GB swap:

```
sudo fallocate -l 2G /swapfile
sudo chmod 600 /swapfile
sudo mkswap /swapfile
sudo swapon /swapfile
echo '/swapfile none swap sw 0 0' | sudo tee -a /etc/fstab
free -h
```

3.3 Clone repo on EC2

Command (non-destructive if the folder already exists):

```
if [ -d "${APP_DIR}" ]; then
  mv "${APP_DIR}" "${APP_DIR}.bak.$(date +%s)"
fi
git clone "${REPO_URL}" "${APP_DIR}"
```

Why:

- Keeps all paths consistent with systemd + Nginx config below.

3.4 Backend Python environment

```
cd "${APP_DIR}/backend"
python3 -m venv ../.venv
source ../.venv/bin/activate
python -m pip install --upgrade pip
pip install -r requirements.txt
```

If you are on a small instance and installs fail due to low disk/tmp space, use a larger tmp directory:

```
mkdir -p "${APP_HOME}/pip-tmp"
export TMPDIR="${APP_HOME}/pip-tmp"

source "${APP_DIR}/.venv/bin/activate"
pip install --no-cache-dir -r "${APP_DIR}/backend/requirements.txt"
```

If you are using `sentence-transformers` and torch installs are slow/failing, install a CPU-only torch wheel first:

```
source "${APP_DIR}/.venv/bin/activate"
pip install --no-cache-dir torch==2.2.1 --index-url https://download.pytorch.org/whl/cpu
pip install --no-cache-dir -r "${APP_DIR}/backend/requirements.txt"
```

Why:

- Avoids pip exhausting `/tmp` on tiny instances.
- CPU-only torch wheels are smaller and more reliable on non-GPU hosts.
- Uses an isolated venv for deterministic installs.
- `pip install --upgrade pip` reduces install failures on newer wheels.

Verification:

```
python -c "import sys; print(sys.version)"
python -c "import fastapi, uvicorn; print('fastapi', fastapi.__version__, 'uvicorn', uvicorn
```

3.5 Create production `.env` (repo root on EC2)

The backend loads env vars from:

- `${APP_DIR}/.env`

Create it: Generate a strong session secret:

```
python3 -c "import secrets; print(secrets.token_urlsafe(48))"
```

Create a clean `.env` skeleton (no fake tokens):

```
cat > "${APP_DIR}/.env" <<'EOF'
SESSION_SECRET=
COOKIE_DOMAIN=
FRONTEND_BASE_URL=
GOOGLE_OAUTH_REDIRECT_BASE=
CORS_ALLOWED_ORIGINS=

DATABASE_URL=

OPENAI_API_KEY=

SLACK_BOT_TOKEN=
SLACK_APP_TOKEN=
SLACK_APP_ID=
SLACK_CLIENT_ID=
SLACK_CLIENT_SECRET=
SLACK_SIGNING_SECRET=
SLACK_VERIFICATION_TOKEN=
SOCKET_MODE_ENABLED=true
MAX_RECONNECT_ATTEMPTS=10
SLACK_CONVERSATIONS_HISTORY_LIMIT=200
SLACK_MODE=standard

NOTION_TOKEN=
NOTION_PARENT_PAGE_ID=
NOTION_MODE=standard
```

```
GOOGLE_CLIENT_ID=
GOOGLE_CLIENT_SECRET=

API_HOST=0.0.0.0
API_PORT=8000

LOG_LEVEL=INFO
EOF

chmod 600 "${APP_DIR}/.env"
nano "${APP_DIR}/.env"
```

Fill these values (required):

- `SESSION_SECRET`: paste the generated secret
- `COOKIE_DOMAIN=.${DOMAIN}`
- `FRONTEND_BASE_URL=https://${FRONTEND_DOMAIN}`
- `GOOGLE_OAUTH_REDIRECT_BASE=https://${API_DOMAIN}`
- `CORS_ALLOWED_ORIGINS=https://${FRONTEND_DOMAIN},https://${DOMAIN}`
- `DATABASE_URL`: paste Supabase Postgres URL (with SSL)
- `OPENAI_API_KEY`
- Slack tokens + app creds
- Notion token + parent page id
- Google OAuth client id/secret

Why:

- No fake token strings that look real.
- You explicitly paste the real values from each provider.

Notes:

- **Do not** set `VITE_API_BASE_URL` in this backend `.env`. That is a frontend build-time var.
- Keep secrets in EC2 only (or a secret manager).

Hardening recommendation:

```
chmod 600 .env
```

3.5.1 Test database connectivity from EC2 (recommended)

This confirms your EC2 host can reach Supabase (including IPv6 cases) *before* you debug the app.

On EC2:

```
set -a
source "${APP_DIR}/.env"
set +a
```

Amazon Linux 2023:

```
psql "${DATABASE_URL}" -c 'select 1;'
```

Ubuntu 24.04:

```
sudo apt-get update
sudo apt-get install -y postgresql-client
psql "${DATABASE_URL}" -c 'select 1;'
```

If you use a secret manager, set env vars in systemd and keep the `.env` file minimal.

3.6 Google OAuth configuration (production)

In Google Cloud Console → OAuth Client:

- **Authorized JavaScript origins**:
  - `https://${FRONTEND_DOMAIN}`
  - (optional) `https://${DOMAIN}`
- **Authorized redirect URIs**:
  - `https://${API_DOMAIN}/auth/google/callback`

Backend endpoints (confirmed in code):

- `GET /auth/google/login`
- `GET /auth/google/callback`

Verification:

- Visiting `https://${API_DOMAIN}/auth/google/login` should redirect to Google.
- After login, you should land on the frontend with a valid cookie.

3.7 Run backend via systemd

Create a systemd service that:

- Uses working directory `${APP_DIR}/backend`
- Uses venv `${APP_DIR}/.venv`
- Runs `uvicorn api.main:app --host 127.0.0.1 --port 8000`

Create the systemd unit file (copy/paste):

```
sudo tee /etc/systemd/system/workforce-backend.service >/dev/null <<EOF
[Unit]
Description=Workforce Backend (Uvicorn)
After=network.target

[Service]
User=${APP_USER}
WorkingDirectory=${APP_DIR}/backend
```

```
Environment=PYTHONUNBUFFERED=1
EnvironmentFile=${APP_DIR}/.env

# Single-process: this app starts background scheduler threads on startup.
ExecStart=${APP_DIR}/.venv/bin/uvicorn api.main:app --host 127.0.0.1 --port 8000

Restart=always
RestartSec=3
NoNewPrivileges=true
PrivateTmp=true

[Install]
WantedBy=multi-user.target
EOF
```

Amazon Linux 2023 (SELinux): allow Nginx to proxy to Uvicorn:

```
if command -v getenforce >/dev/null 2>&1; then
  getenforce
fi

sudo setsebool -P httpd_can_network_connect 1 || true
```

Important details:

- This binds Uvicorn to `127.0.0.1` so it is only reachable through Nginx.
- If you keep `0.0.0.0`, ensure port `8000` is not publicly exposed.

Commands:

```
sudo systemctl daemon-reload
sudo systemctl enable workforce-backend
sudo systemctl restart workforce-backend
sudo journalctl -u workforce-backend -n 200 --no-pager
```

Quick smoke test (from the server):

```
curl -sS http://127.0.0.1:8000/health | head
```

Important architecture note (production constraint):

- The backend starts background scheduler threads on startup.
- Until those are moved to a separate worker process (or guarded by leader election), you should run **a single Uvicorn worker**. Do not switch to multi-worker Gunicorn/Uvicorn without refactoring, or you can get duplicated scheduled jobs and inconsistent in-memory state.

---

4) Nginx reverse proxy (HTTPS + streaming + websockets)

4.1 Why nginx config matters

This app uses:

- **WebSockets** (`/api/chat/ws`)
- **Streaming NDJSON** (`/api/chat/project/${PROJECT_ID}/stream`)

If nginx buffers responses or proxies with HTTP/1.0, **project chat streaming will hang**.

4.2 Nginx (HTTP first), then TLS via Certbot, then verify streaming

Important:

- Do NOT write an HTTPS server block that references `/etc/letsencrypt/...` before the certificate exists.
- The flow is:
  - Put Nginx in front of Uvicorn (HTTP)
  - Obtain certs (Certbot)
  - Ensure the final HTTPS server block includes the streaming + WebSocket locations

4.2.1 Install an HTTP Nginx reverse proxy (copy/paste)

```
sudo tee /etc/nginx/conf.d/workforce-backend.conf >/dev/null <<EOF
map \$http_upgrade \$connection_upgrade {
  default upgrade;
  '' close;
}

server {
  listen 80;
  listen [::]:80;
  server_name ${API_DOMAIN};

  location /api/chat/ws {
    proxy_pass http://127.0.0.1:8000;
    proxy_http_version 1.1;
    proxy_set_header Upgrade \$http_upgrade;
    proxy_set_header Connection \$connection_upgrade;
    proxy_set_header Host \$host;
    proxy_set_header X-Real-IP \$remote_addr;
    proxy_set_header X-Forwarded-For \$proxy_add_x_forwarded_for;
    proxy_set_header X-Forwarded-Proto \$scheme;
    proxy_read_timeout 7d;
    proxy_send_timeout 7d;
    proxy_buffering off;
```

```
  }

  location /api/chat/project/ {
    proxy_pass http://127.0.0.1:8000;
    proxy_http_version 1.1;
    proxy_set_header Connection "";
    proxy_set_header Host \$host;
    proxy_set_header X-Real-IP \$remote_addr;
    proxy_set_header X-Forwarded-For \$proxy_add_x_forwarded_for;
    proxy_set_header X-Forwarded-Proto \$scheme;
    proxy_buffering off;
    proxy_request_buffering off;
    gzip off;
    proxy_cache off;
    proxy_read_timeout 600s;
    proxy_send_timeout 600s;
  }

  location / {
    proxy_pass http://127.0.0.1:8000;
    proxy_http_version 1.1;
    proxy_set_header Connection "";
    proxy_set_header Host \$host;
    proxy_set_header X-Real-IP \$remote_addr;
    proxy_set_header X-Forwarded-For \$proxy_add_x_forwarded_for;
    proxy_set_header X-Forwarded-Proto \$scheme;
    proxy_connect_timeout 10s;
    proxy_send_timeout 120s;
    proxy_read_timeout 120s;
  }
}
EOF
```

Why:

- Guarantees you don't miss a required directive.
- Ensures streaming routes are not buffered and WebSockets upgrade correctly.

Reload:

```
sudo nginx -t
sudo systemctl reload nginx
```

Verify HTTP is reachable before TLS:

```
curl -i "http://${API_DOMAIN}/health"
```

If you keep the `/.well-known/acme-challenge/` webroot stanza, create the

directory:

```
sudo mkdir -p /var/www/certbot
sudo chown -R www-data:www-data /var/www/certbot || true
```

Verification:

```
curl -i http://127.0.0.1:8000/health
curl -i "http://${API_DOMAIN}/health"
```

4.3 TLS certificates (Let's Encrypt)

Install Certbot based on your host OS:

- Amazon Linux 2023: Certbot via `/opt/certbot` virtualenv (installed in section 3.2)
- Ubuntu 24.04: Certbot via **Snap**

Amazon Linux 2023 verification:

```
certbot --version
```

4.3.1 Install Certbot (Ubuntu 24.04 via Snap) If you have an OS-package certbot installed, remove it first to avoid conflicts.

Commands:

```
sudo apt-get remove -y certbot || true
sudo snap install --classic certbot
sudo ln -s /snap/bin/certbot /usr/bin/certbot
certbot --version
```

If `sudo snap install ...` fails, make sure `snapd` is installed and running:

```
sudo systemctl enable --now snapd
sudo systemctl status snapd --no-pager
```

4.3.2 Issue certificate for the API domain You must have:

- DNS A record for `${API_DOMAIN}` pointing to this server
- Port 80 reachable (for HTTP-01 challenge)

Run:

```
sudo certbot --nginx -d "${API_DOMAIN}"
```

Important:

- Certbot will generate certificates and may also generate an initial HTTPS server block.
- You must ensure the final HTTPS server block contains the **WebSocket + streaming** locations.

4.3.3 Write the final HTTPS Nginx config (copy/paste) After certs exist at `/etc/letsencrypt/live/${API_DOMAIN}/...`, overwrite the Nginx config with the final version:

```
sudo tee /etc/nginx/conf.d/workforce-backend.conf >/dev/null <<EOF
map \$http_upgrade \$connection_upgrade {
  default upgrade;
  '' close;
}

server {
  listen 80;
  listen [::]:80;
  server_name ${API_DOMAIN};
  return 301 https://\$host\$request_uri;
}

server {
  listen 443 ssl http2;
  listen [::]:443 ssl http2;
  server_name ${API_DOMAIN};

  ssl_certificate /etc/letsencrypt/live/${API_DOMAIN}/fullchain.pem;
  ssl_certificate_key /etc/letsencrypt/live/${API_DOMAIN}/privkey.pem;

  client_max_body_size 25m;

  location /api/chat/ws {
    proxy_pass http://127.0.0.1:8000;
    proxy_http_version 1.1;
    proxy_set_header Upgrade \$http_upgrade;
    proxy_set_header Connection \$connection_upgrade;
    proxy_set_header Host \$host;
    proxy_set_header X-Real-IP \$remote_addr;
    proxy_set_header X-Forwarded-For \$proxy_add_x_forwarded_for;
    proxy_set_header X-Forwarded-Proto \$scheme;
    proxy_read_timeout 7d;
    proxy_send_timeout 7d;
    proxy_buffering off;
  }

  location /api/chat/project/ {
    proxy_pass http://127.0.0.1:8000;
    proxy_http_version 1.1;
    proxy_set_header Connection "";
    proxy_set_header Host \$host;
```

```
    proxy_set_header X-Real-IP \$remote_addr;
    proxy_set_header X-Forwarded-For \$proxy_add_x_forwarded_for;
    proxy_set_header X-Forwarded-Proto \$scheme;
    proxy_buffering off;
    proxy_request_buffering off;
    gzip off;
    proxy_cache off;
    proxy_read_timeout 600s;
    proxy_send_timeout 600s;
  }

  location / {
    proxy_pass http://127.0.0.1:8000;
    proxy_http_version 1.1;
    proxy_set_header Connection "";
    proxy_set_header Host \$host;
    proxy_set_header X-Real-IP \$remote_addr;
    proxy_set_header X-Forwarded-For \$proxy_add_x_forwarded_for;
    proxy_set_header X-Forwarded-Proto \$scheme;
    proxy_connect_timeout 10s;
    proxy_send_timeout 120s;
    proxy_read_timeout 120s;
  }
}
EOF
```

After writing the final config:

```
sudo nginx -t
sudo systemctl reload nginx
curl -i "https://${API_DOMAIN}/health"
```

4.3.4 Verify renewal

```
sudo certbot renew --dry-run
```

Then verify:

- `https://${API_DOMAIN}/auth/me` returns 401/200 (depending on cookie)

---

5) Frontend on Vercel

5.1 Create Vercel project

- Import your repo
- Set **Root Directory** to `frontend/`
- Build command: `npm run build`

- Output directory: `dist`

5.2 Frontend environment variables (Vercel)

Set this in Vercel Project → Settings → Environment Variables:

- `VITE_API_BASE_URL=https://${API_DOMAIN}`

Verification:

- Ensure requests from the frontend go to `https://${API_DOMAIN}/...`.
- If you change `VITE_API_BASE_URL`, you must trigger a new Vercel deploy.

(You can set it for Production + Preview.)

5.3 Set Vercel domains

In Vercel → Domains:

- Add `${FRONTEND_DOMAIN}`
- Optional: add `${DOMAIN}` and redirect it to `www` (recommended)

5.4 After Vercel deploy: set `FRONTEND_BASE_URL` on EC2 and restart

Your backend CORS allowlist is driven by `FRONTEND_BASE_URL` (see `backend/api/main.py`). After your Vercel production domain is live, update the backend `.env` to match it exactly and restart the backend.

On EC2:

```
nano "${APP_DIR}/.env"
```

Set:

- `FRONTEND_BASE_URL=https://${FRONTEND_DOMAIN}`

Then restart:

```
sudo systemctl restart workforce-backend
sudo journalctl -u workforce-backend -n 200 --no-pager
```

Why:

- If `FRONTEND_BASE_URL` does not match your deployed frontend origin exactly, you will see CORS failures and cookie/auth issues.

---

6) DNS (connect domains)

6.1 Backend DNS

Create an `A` record:

- `api` → your EC2 Elastic IP

Verify:

```
dig +short "${API_DOMAIN}"
```

6.2 Frontend DNS

Follow Vercel's domain instructions. Typically:

- `CNAME` for `www` → `cname.vercel-dns.com`
- Apex `${DOMAIN}` → Vercel-recommended A/ALIAS target

---

7) End-to-end verification checklist

7.1 Basic health

- Open `https://${FRONTEND_DOMAIN}`
- Confirm API base is correct:
  - In browser devtools, requests go to `https://${API_DOMAIN}/...`

Also verify from your laptop:

```
curl -i "https://${API_DOMAIN}/health"
```

Expected:

- HTTP 200
- JSON with `status: ok`

7.2 Auth

- Click "Login" → it should redirect to Google and back
- Confirm callback:
  - Google redirects to `https://${API_DOMAIN}/auth/google/callback`
  - Then you end up at `https://${FRONTEND_DOMAIN}`

Backend verification endpoints:

- `GET https://${API_DOMAIN}/auth/me`
  - Should return 401 when logged out
  - Should return a JSON user profile when logged in

7.3 Cookies

Backend sets session cookie (`wf_session`). For cross-subdomain:

- `COOKIE_DOMAIN=.${DOMAIN}` on backend
- CORS allows your frontend origins
- Frontend requests must use `credentials: 'include'` (already done in code)

7.4 WebSocket (main chat)

- Ensure `/api/chat/ws` connects (no 1012 disconnect loops)

If you want a CLI test from your laptop, you can use a WebSocket client and confirm you get messages back after auth.

7.5 Project chat streaming

- In Projects tab, send a message
- In backend logs you should see:
  - project chunk retrieval
  - OpenAI chat completions call
- In browser Network tab, the `/api/chat/project/${PROJECT_ID}/stream` response should stream lines continuously.

CLI verification (after you have a valid auth cookie):

```
curl -N --no-buffer \
  -H "Accept: application/x-ndjson" \
  "https://${API_DOMAIN}/api/chat/project/${PROJECT_ID}/stream"
```

If it hangs:

- Verify nginx has:
  - `proxy_http_version 1.1;`
  - `proxy_buffering off;` for `/api/chat/project/`

7.6 Connector status

The UI uses a connector health endpoint:

```
curl -i "https://${API_DOMAIN}/api/chat/connectors/status"
```

If this returns 401, auth is failing. If it returns degraded/disconnected, check `.env` and OAuth scopes.

---

8) Troubleshooting

8.1 Project chat "hangs" on EC2 but works locally

Almost always one of:

- nginx `proxy_buffering on` (streaming gets buffered)
- nginx proxying as HTTP/1.0 (missing `proxy_http_version 1.1`)
- timeouts too short

Fix by creating a dedicated nginx location:

- `location /api/chat/project/ { proxy_buffering off; proxy_http_version 1.1; ... }`

If you are using Cloudflare proxying for the API domain:

- Temporarily set `api.${DOMAIN}` DNS to "DNS only" (no proxy) to confirm whether Cloudflare buffering is involved.

8.2 CORS / cookies not sent

- Ensure backend `.env` includes:
  - `FRONTEND_BASE_URL=https://${FRONTEND_DOMAIN}`
  - `COOKIE_DOMAIN=.${DOMAIN}`
  - `CORS_ALLOWED_ORIGINS=https://${FRONTEND_DOMAIN},https://${DOMAIN}`
- Ensure frontend uses `credentials: 'include'`

Extra production checks:

- Verify the cookie is `Secure` in production (HTTPS).
- If you are using a cross-domain setup (not same root domain), browsers may block cookies even with `SameSite=None`.
- This backend includes an ITP workaround endpoint:
  - `POST /auth/session-exchange`
  - This lets the frontend set cookies via a same-site request.

Verify CORS preflight explicitly:

```
curl -I -X OPTIONS "https://${API_DOMAIN}/api/chat/connectors/status" \
  -H "Origin: https://${FRONTEND_DOMAIN}" \
  -H "Access-Control-Request-Method: GET"
```

Expected:

- `Access-Control-Allow-Origin: https://${FRONTEND_DOMAIN}`
- `Access-Control-Allow-Credentials: true`

8.3 Google OAuth "unauthorized_client"

- Your Google OAuth client must include the exact redirect:
  - `https://${API_DOMAIN}/auth/google/callback`
- Ensure you are using the correct OAuth client ID/secret for production.

8.4 Updating deployments

Backend:

```
cd "${APP_DIR}"
git pull
sudo systemctl restart workforce-backend
sudo journalctl -u workforce-backend -n 200 --no-pager
```

8.5 SSL errors only on some WiFi networks

From your laptop:

```
curl -vI "https://${API_DOMAIN}/health"

echo | openssl s_client -servername "${API_DOMAIN}" -connect "${API_DOMAIN}:443" 2>/dev/null

curl --resolve "${API_DOMAIN}:443:$(dig +short "${API_DOMAIN}" @8.8.8.8)" -I "https://${API_
```

Why:

- If it fails only on one WiFi and works on mobile data, it's usually network SSL inspection/filters, not your server.

If you suspect your certificate is stale or misinstalled on EC2:

```
sudo certbot certificates
sudo certbot renew --force-renewal
sudo systemctl restart nginx
curl -i "https://${API_DOMAIN}/health"
```

Frontend:

- Push to GitHub → Vercel auto-deploys.

---

9) Operations (updates, logs, backups, rotations)

9.1 Logs

- Backend logs:

```
sudo journalctl -u workforce-backend -f
```

- Nginx logs:
  - `/var/log/nginx/access.log`
  - `/var/log/nginx/error.log`

9.2 Safe restart checklist

- `sudo nginx -t`
- `sudo systemctl reload nginx`
- `sudo systemctl restart workforce-backend`
- `curl -i "https://${API_DOMAIN}/health"`

9.2.1 Rollback plan (simple)

- Keep the previous git commit hash.
- If a deploy breaks production:

```
cd "${APP_DIR}"
git log -n 10 --oneline
git checkout "${KNOWN_GOOD_COMMIT}"
```

```
sudo systemctl restart workforce-backend
curl -i "https://${API_DOMAIN}/health"
```

9.2.2 Inspect and edit the systemd service safely

View the active unit:

```
sudo systemctl status workforce-backend.service --no-pager -l
sudo systemctl cat workforce-backend.service --no-pager
```

Edit the full unit:

```
sudo systemctl edit --full workforce-backend.service
```

Apply changes:

```
sudo systemctl daemon-reload
sudo systemctl restart workforce-backend.service
sudo systemctl status workforce-backend.service --no-pager -l
```

Verify what is actually running:

```
ps auxww | grep uvicorn | grep -v grep
ss -lntp | grep :8000
```

9.3 Backups

- Supabase: configure scheduled backups / PITR according to your plan.
- Store OAuth client secrets and Slack tokens in a secure vault.

9.4 Key rotation playbook

- Rotate if exposed:
  - `OPENAI_API_KEY`
  - Slack tokens
  - Google OAuth client secret
  - Notion token

After rotation:

- Update `${APP_DIR}/.env`
- `sudo systemctl restart workforce-backend`

---

10) Cost control (AWS) and teardown

10.1 Keep AWS costs predictable

In AWS Billing & Cost Management:

- Create a **monthly cost budget**.
- Add an **80% alert** and a **100% alert** to your email.

Why:

- Prevents surprise spend while iterating on hosting.

10.2 Teardown (start over cleanly)

Do this order to avoid ongoing charges:

1. Remove DNS records pointing at the backend (the `api` A record).
2. Disassociate and **release** the Elastic IP.
3. Terminate the EC2 instance.
4. Delete any custom security groups/key pairs created only for this deployment.

Why:

- Elastic IPs can accrue charges if left allocated but unused.

10.3 Free tier / billing alerts

In AWS Billing:

- Enable Free Tier usage alerts
- Enable billing alerts

Why:

- You get early warnings if you accidentally exceed free-tier allowances.

---

11) Security hardening checklist

- Disable password SSH login (keys only)
- Restrict SSH (22) to your IP
- Keep 80/443 open
- Do not expose 8000
- Keep `.env` permissions `600`
- Consider `fail2ban` and automatic security upgrades

Must-do:

- Never commit `.env`.
- If you accidentally shared keys/tokens, rotate them:
  - OpenAI key
  - Slack tokens
  - Google client secret
  - Notion token
