

Standing Up the Edge

The reverse proxy and public TLS, before any inspection is added. This is the baseline every public site sits on. Nginx routing a hostname to a backend over public TLS, with a consistent per-site configuration that new services slot into.

Install nginx from the official repository

Open-appsec attaches to specific nginx versions, so nginx has to come from nginx's own repository (not the distribution's package) and then be held at a compatible version. Review [Open-appsec nginx attachment documentation](#) to find the list of compatible nginx versions. Then add the official repo, install that version, and pin it:

```
curl -fsSL https://nginx.org/keys/nginx_signing.key \  
| sudo gpg --dearmor -o /usr/share/keyrings/nginx-archive-keyring.gpg  
  
echo "deb [signed-by=/usr/share/keyrings/nginx-archive-keyring.gpg]  
http://nginx.org/packages/ubuntu noble nginx" \  
| sudo tee /etc/apt/sources.list.d/nginx.org.list  
  
sudo tee /etc/apt/preferences.d/99nginx <<'EOF'  
Package: nginx*  
Pin: origin nginx.org  
Pin-Priority: 900  
EOF  
  
sudo apt update  
sudo apt install nginx<version> certbot python3-certbot-nginx
```

Then hold nginx so an unattended upgrade cannot move it to a version open-appsec does not yet support:

```
sudo apt-mark hold nginx
```

That hold is important because an automatic nginx upgrade can break the open-appsec attachment. Nginx must be updated deliberately, in step with open-appsec's compatibility, not automatically. This is a common way the setup breaks weeks later.

Some versions of nginx handle activating websites differently. Personally, I follow the `sites-available` -> `sites-enabled` method, which requires a change in `nginx.conf`:

```
include /etc/nginx/sites-enabled/*;
```

The per-site pattern

Every public service follows the same server-block shape. Understanding it once means every new site is the same move. Here is the pattern, annotated, with the public name and the internal backend name as specific things that must change per site. Additional settings may need to be modified or added depending on the service hosted.

```
server {
    listen 443 ssl;
    server_name app.example.org;                # the public name

    # Public TLS – the certificate a browser sees (managed by Certbot)
    ssl_certificate      /etc/letsencrypt/live/app.example.org/fullchain.pem;
    ssl_certificate_key  /etc/letsencrypt/live/app.example.org/privkey.pem;
    include /etc/letsencrypt/options-ssl-nginx.conf;

    # Security headers
    add_header X-Frame-Options DENY;            # disallow framing (clickjacking)
    add_header X-Content-Type-Options nosniff;  # no MIME-type sniffing

    client_max_body_size 512m;                  # raise/lower per site; comment out if no
    uploads

    location / {
        limit_req zone=general burst=20 nodelay; # rate limiting (see note below)

        proxy_pass https://app.int.example;      # internal backend, over mTLS

        proxy_http_version 1.1;
        proxy_set_header Connection "";          # comment out if the app needs WebSockets
    }
}
```

```

proxy_set_header Host          $host;
proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
proxy_set_header X-Forwarded-Proto $scheme;

# --- internal mutual TLS to the backend ---
# Verify the backend's server certificate against the internal CA
proxy_ssl_server_name on;
proxy_ssl_name app.int.example;
proxy_ssl_trusted_certificate /etc/ssl/certs/internal-ca.crt;
proxy_ssl_verify on;
proxy_ssl_verify_depth 2;

# Present the WAF's client certificate to the backend
proxy_ssl_certificate      /etc/ssl/certs/waf-to-app.crt;
proxy_ssl_certificate_key /etc/ssl/private/waf-to-app.key;
}
}

```

The important pieces:

- **server_name** is the public name; **proxy_pass** is the internal name. The WAF translates a public request into an internal, mutually-authenticated one. The two names are different domains of trust meeting in one block.
- **The security headers** are set at the edge so every site gets them without each backend having to. **X-Frame-Options DENY** prevents the site being framed; **X-Content-Type-Options nosniff** stops browsers from second-guessing content types.
- **client_max_body_size** is tuned per site: large for a file-sync service, small or default for a forum. Set it to what the app legitimately needs and no larger, since it is also a limit on how much a malicious request can send.
- **limit_req** applies rate limiting per site, which is a basic protection against floods and brute-force attempts. Different services tolerate different rates; a sync-heavy or automation service needs a higher burst than a wiki.
- **The proxy_ssl_* block** is the internal mTLS handoff, covered in the mTLS book. It is what makes the connection inward authenticated rather than plaintext.

Public certificates

Public TLS certificates are obtained and renewed with Certbot on the WAF. Because this host is the only place public certificates live, it is the only place they are renewed:

```
sudo certbot --nginx -d app.example.org
```

To run renewal on a schedule, run `sudo crontab -e` and add the following line. This runs certbot on the 5th day of each month. Certbot will renew a certificate once it is within 30 days of expiration.

```
1 0 5 * * certbot renew
```

Certbot's nginx integration reloads nginx after a successful renewal, so unlike the internal certificates (which need an explicit reload, covered in the mTLS book), the public certificates take care of themselves here.

Revision #2

Created 2026-07-27 17:52:24 UTC by Chris Landis

Updated 2026-07-27 18:06:51 UTC by Chris Landis