anoBytes | TECHNOLOGY SOLUTIONS

nginx as a TLS reverse proxy on AWS

A hardened front door for a private backend, with Let's Encrypt, on Ubuntu 24.04 and Amazon Linux 2023

NGINX · LET'S ENCRYPT · VPC SEGMENTATION · HOW-TO GUIDE

General educational guidance, not affiliated with or endorsed by nginx, Let's Encrypt, or AWS. Published 2026-08-06.

1 internet-facing host

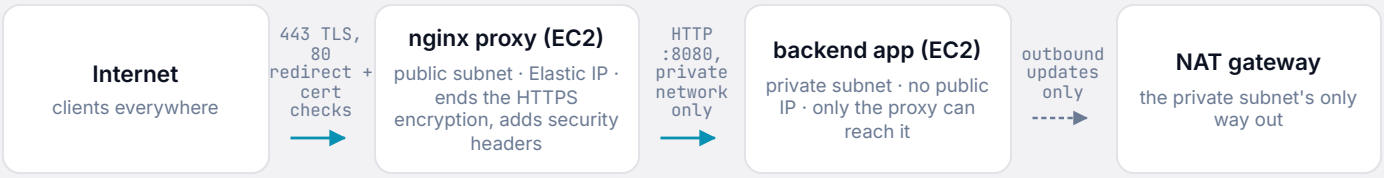
nginx handles the encryption; the backend has no public IP

0 open management ports

Session Manager replaces SSH; port 22 stays closed

01 What you are building

A reverse proxy takes requests from the internet and passes them to an application that never faces the internet itself. You get one host to harden and patch, and you can replace or scale the app behind it.



Cyan solid arrows are inbound requests. The grey dashed arrow is outbound only: the backend reaches the internet for updates through the NAT gateway, and nothing inbound ever uses that path.

Routing decides what a subnet is

A public subnet has a route to an internet gateway. A private subnet does not, so nothing in it can be reached from the internet, and it only reaches out through a NAT gateway.

The security group is the firewall

Security groups attach to instances and are stateful, meaning replies to allowed traffic are let back through automatically. The backend accepts its app port only from the proxy's security group, named by ID instead of by IP range, so the rule keeps working when addresses change.

nginx is the single entry point

It is the only host with a public address, the only place the HTTPS encryption ends, and the only place you manage certificates and security headers.

What this does not do

It is not a web application firewall, it does not patch the backend for you, and it does not authenticate users. Attacks that ride valid HTTPS requests still reach the backend, so keep the app itself maintained.

02 Prerequisites and costs

Have these ready before Step 1, and note that two line items bill by the hour.

You need

- An AWS account and a registered domain whose DNS you can edit. Route 53 is assumed for the DNS-01 path; any DNS provider works for HTTP-01.
- A VPC you can add subnets and route tables to.
- Familiarity with launching EC2 instances and attaching IAM roles.
- A backend app on a known port. This guide uses 8080 on 10.0.20.10.

Budget warning

`PUBLIC IPV4 + NAT GATEWAY BILL HOURLY`

- Since 1 February 2024, every public IPv4 address costs about $0.005 per hour, attached or not. The proxy's Elastic IP and the NAT gateway's Elastic IP each run roughly $3.65 a month.
- The NAT gateway itself also bills per hour and per gigabyte processed.
- Tear the lab down when you are done.

03 Step 1 · Build the network

This is the layered network layout the AWS Well-Architected guidance recommends: one way in, through the internet gateway to the proxy, and one way out, through the NAT gateway. You need one VPC, two subnets, an internet gateway, a NAT gateway, and two route tables. The console's "VPC and more" wizard creates all of it in one pass; the routing is what makes a subnet public or private.

Internet gateway

`ONE PER VPC · THE ONLY WAY IN`

Attach one to the VPC. The public subnet's route table sends 0.0.0.0/0 (all destinations) to it. The nginx proxy and the NAT gateway live in that subnet.

Private subnet + NAT gateway

`0.0.0.0/0 → NAT, NOT THE IGW`

The backend lives here with no public address. All of its outbound traffic (package updates, the Session Manager connection from Step 3) goes out through the NAT gateway; nothing comes in. For IPv6-only outbound, an egress-only internet gateway does the same job without the NAT charge.

Stable public address + DNS

`ELASTIC IP · A RECORD`

Allocate an Elastic IP (AWS's name for a static public IP that stays yours), attach it to the proxy instance, and create a DNS A record pointing example.com (and www if you use it) at that address.

04 Step 2 · Write the security groups

Two groups enforce the segmentation. You only write the inbound rules; because security groups are stateful, the replies flow back automatically.

GROUP	INBOUND	SOURCE	WHY
Proxy SG `nginx instance`	**443**	**0.0.0.0/0, ::/0**	Public HTTPS
Proxy SG	**80**	**0.0.0.0/0, ::/0**	HTTP→HTTPS redirect and Let's Encrypt HTTP-01
Proxy SG	**22**	**omit**	Use Session Manager; no public SSH port
Backend SG `app instance`	**8080**	**the proxy security-group ID (sg-…)**	Only the proxy may reach the app
Backend SG	**22**	**omit**	No management port exposed

- **Set the source to the security-group ID, not an IP range**
  AWS lets one security group name another as its source (in the same VPC, across VPC peering, or, for inbound rules where it is turned on, across a transit gateway). With the backend's rule pointing at the proxy's group ID, nothing outside that group can open the app port, and the rule still works after instances are replaced or IPs change. Outbound stays default-allow on both groups.

05 Step 3 · Launch the instances

Proxy into the public subnet with the Elastic IP; backend into the private subnet with no public IP. Set two things on both instances.

Require IMDSv2

`HTTPTOKENS=REQUIRED · SET AT LAUNCH`

Every EC2 instance answers a special local address, 169.254.169.254, that hands out its credentials. A reverse proxy is a classic target for server-side request forgery (SSRF), where an attacker tricks the proxy into reading that address for them. IMDSv2 blocks the trick: every read needs a token minted by a PUT request first, and the token cannot leave the instance.

Attach an IAM role for Session Manager

`AMAZONSSMMANAGEDINSTANCECORE · NO BASTION, NO SSH KEY, NO INBOUND PORT`

- The SSM Agent comes preinstalled on the standard Ubuntu 24.04 LTS and Amazon Linux 2023 AMIs. Confirm it with `systemctl status amazon-ssm-agent` (Ubuntu: `snap.amazon-ssm-agent…`).
- Attach an IAM role with the AWS managed policy AmazonSSMManagedInstanceCore to both instances.
- The instances need a path to the SSM endpoints: through the NAT gateway, or through ssm, ssmmessages, and ec2messages VPC interface endpoints (plus an S3 gateway endpoint) if the backend should need no NAT at all; current agents use ssmmessages, and AWS keeps ec2messages on the list for older ones.
- The command below runs from your workstation and needs the Session Manager plugin, a separate download from the AWS CLI itself.

```
aws ssm start-session --target i-0123456789abcdef0
```

Prefer a normal SSH terminal? Two private alternatives

`TAILSCALE SSH · EC2 INSTANCE CONNECT ENDPOINT`

- Session Manager is the default here because it needs nothing extra on AWS: access is controlled through IAM, sessions can be logged to CloudWatch or S3, and no port listens anywhere.
- If you run many servers and want a plain ssh workflow, a mesh VPN such as Tailscale works well: install its agent on each instance, enable Tailscale SSH, and every box is one ssh command away over WireGuard, with port 22 still closed to the internet. The trade-offs are an agent on every host, a third-party service in the login path, and session recording that is plan-dependent and needs a recorder node you run.
- AWS's EC2 Instance Connect Endpoint is a middle option: standard SSH into private instances through a VPC endpoint, with no public IP and no bastion (jump host) needed.

06 Step 4 · Install nginx on the proxy

The package and paths differ by distribution. On both, check config changes with nginx -t and apply them with systemctl reload nginx.

Ubuntu 24.04

`NGINX 1.24.0 · RUNS AS WWW-DATA · WEB ROOT /VAR/WWW/HTML`

Server blocks live in /etc/nginx/sites-available/ and are enabled by symlink into /etc/nginx/sites-enabled/.

```
sudo apt update && sudo apt install nginx
sudo systemctl enable --now nginx
```

Amazon Linux 2023

NGINX 1.30.4 · RUNS AS NGINX · WEB ROOT /USR/SHARE/NGINX/HTML

AL2023 is the current Amazon Linux (AL2 reached end of life on 30 June 2026). nginx is in the default repositories, and server blocks live in /etc/nginx/conf.d/*.conf. If you switch AL2023 to SELinux enforcing mode, note the SELinux warning in Step 5.

```
sudo dnf install nginx
sudo systemctl enable --now nginx
```

07 Step 5 · Configure the reverse proxy, HTTP first

Get proxying working over plain HTTP, then add the certificate in Step 6. Doing it in two stages keeps a TLS problem from hiding a proxy problem. Create the config on Ubuntu at /etc/nginx/sites-available/example.com (symlinked into sites-enabled/), on AL2023 at /etc/nginx/conf.d/example.com.conf.

```
upstream backend {
    server 10.0.20.10:8080;   # the backend's private IP and port
    keepalive 16;
}

server {
    listen 80;
    listen [::]:80;
    server_name example.com www.example.com;

    location / {
        proxy_pass http://backend;
        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_set_header X-Forwarded-Host  $host;
    }
}
```

What each line does

- `keepalive 16` holds up to 16 idle connections to the backend open for reuse, instead of opening a new one per request. It only works when the next two lines are also set.
- `proxy_http_version 1.1` and `Connection ""` are what nginx requires (on every version before 1.29.7) for that reuse to happen. Without the cleared Connection header, nginx tells the backend to close after every request and the keepalive pool sits unused.
- `Host $host` sends the backend the hostname the client asked for. Without it, nginx sends the upstream's own name.
- `X-Forwarded-For` adds the client IP to the request, so the backend logs the real client instead of the proxy.
- `X-Forwarded-Proto` tells the backend the original request was HTTPS, so the app builds https:// URLs and sets Secure cookies.
- `X-Real-IP` repeats the client IP as a single value, and `X-Forwarded-Host` repeats the requested hostname. Some apps read these names instead of the X-Forwarded pair, and sending both styles is harmless.

WebSocket backends

On that location, replace the empty Connection header with the upgrade pair: `proxy_set_header Upgrade $http_upgrade;` and `proxy_set_header Connection "upgrade";`

Check it before adding TLS

`sudo nginx -t && sudo systemctl reload nginx` then `curl -I http://example.com/` should reach the backend.

- **AL2023 with SELinux enforcing: expect 502s**

  AL2023 ships SELinux in permissive mode, so a fresh box proxies fine. Harden it to enforcing and nginx can no longer open connections to the backend: every request returns 502 with a permission-denied entry in the error log. Allow it with one of two switches:

  - `sudo setsebool -P httpd_can_network_relay 1` allows only a fixed set of web and proxy ports: 80, 81, 443, 488, 8008, 8009, 8443, and 9000, plus cache and proxy ports including 8080, 8118, and 3128. That covers this guide's backend on 8080, so prefer it here; it is the least-privilege choice, since it opens only the ports this backend needs.
  - `sudo setsebool -P httpd_can_network_connect 1` allows any outbound port. Use it when the backend listens outside that set, on 3000 or 5000 for example.

  Ubuntu does not confine nginx with AppArmor by default, so there is no equivalent step there.

Let’s Encrypt, hardening, and renewal

Issue the certificate, harden the TLS configuration, and make renewal automatic

08 Step 6 · Get a Let’s Encrypt certificate

Certbot does the work here. It speaks ACME, the protocol Let's Encrypt uses to check that you control a domain and then issue the certificate, and it automates the whole issue-and-renew loop. Two recent changes at Let's Encrypt affect how you run it.

No more expiration emails

REMINDER EMAILS ENDED 4 JUNE 2025

Let's Encrypt no longer warns you before a certificate expires. Set up the automated renewal in Step 8 and add your own monitoring.

Certificate lifetimes are getting shorter

PROFILES: CLASSIC 90 DAYS · TLSSERVER 45 DAYS · SHORTLIVED 6 DAYS

- The default classic profile still issues 90-day certificates.
- The tlsserver profile issues 45-day certificates (since 13 May 2026), and the shortlived profile issues certificates valid for 160 hours, which Let's Encrypt calls six-day certificates.
- Certbot picks a profile with --preferred-profile. Leave it unset and you get the 90-day default, which is fine for a first deployment.

Rate limits during testing

- 50 certificates per registered domain per rolling 7 days.
- 5 duplicate certificates (the same exact set of names) per rolling 7 days.
- 5 failed validations per hostname, per account, per hour.
- Use --staging while you work out the flow so a mistake does not use up a limit.

INSTALL CERTBOT · UBUNTU 24.04 (EFF RECOMMENDS THE SNAP; THE APT PACKAGE IS 2.9.0 AND LAGS)

```
sudo apt remove certbot 2>/dev/null    # drop any apt version first
sudo snap install core && sudo snap refresh core
sudo snap install --classic certbot
sudo ln -s /snap/bin/certbot /usr/bin/certbot
```

INSTALL CERTBOT · AMAZON LINUX 2023 (NO EPEL, NO SNAPD; DNF HAS ONLY CERTBOT 2.6.0: PIP INTO A VENV)

```
sudo dnf install -y python3.11 python3.11-pip augeas-libs
sudo python3.11 -m venv /opt/certbot/
sudo /opt/certbot/bin/pip install --upgrade pip
sudo /opt/certbot/bin/pip install certbot certbot-nginx
sudo ln -s /opt/certbot/bin/certbot /usr/bin/certbot
```

The venv is a self-contained Python environment under /opt/certbot. It is built with python3.11 because AL2023's system Python is 3.9 and current Certbot needs 3.10 or newer. AL2023's repositories do carry certbot with the nginx and Route 53 plugins, but frozen at 2.6.0, older than the profiles and renewal behavior this guide uses; the venv installs the current release. The pip install creates no renewal timer; Step 8 adds one by hand.

Path A · HTTP-01: the simplest, needs port 80

```
sudo certbot --nginx -d example.com -d www.example.com
```

- The proxy already answers on port 80, so the nginx plugin proves you control the name, adds the listen 443 ssl config, and offers the HTTP→HTTPS redirect.
- HTTP-01 cannot issue wildcard certificates.
- To keep your nginx config untouched, use certbot certonly --nginx instead: it authenticates, reverts its temporary changes, and leaves the TLS block for you to write in Step 7.

Path B · DNS-01 via Route 53: no port 80, supports wildcards

```
sudo certbot certonly --dns-route53 -d example.com -d '*.example.com'
```

- DNS-01 proves control by writing a TXT record in your DNS, so it works with port 80 closed, and it is the only way to get a wildcard (one certificate that covers every subdomain).
- Install the plugin. AL2023: pip install certbot-dns-route53 into the venv. Ubuntu snap: snap set certbot trust-plugin-with-root=ok, then snap install certbot-dns-route53.
- Quote the wildcard so the shell does not expand it.
- Use the instance role, not static access keys: certbot picks up the role's credentials automatically, so no keys sit on disk.

IAM POLICY FOR THE DNS-01 ROLE (MATCHES THE CERTBOT PLUGIN'S DOCUMENTED POLICY)

```
{
  "Version": "2012-10-17",
  "Statement": [
    { "Effect": "Allow",
      "Action": ["route53:ListHostedZones", "route53:GetChange"],
      "Resource": ["*"] },
    { "Effect": "Allow",
      "Action": ["route53:ChangeResourceRecordSets"],
      "Resource": ["arn:aws:route53:::hostedzone/YOURHOSTEDZONEID"] }
  ]
}
```

The hosted zone is your domain's DNS zone in Route 53; its ID is on the zone's details page. The two read actions stay on *: ListHostedZones supports no resource scoping, and a GetChange change ID cannot be known in advance, so certbot's own documented policy uses * for both; the write action is limited to your one zone. Add the policy to the proxy's instance role from Step 3: an instance carries one role, so these permissions go on the same role as the Session Manager policy.

09 Step 7 · Harden the TLS configuration

certbot --nginx writes a working but generic TLS block. Replace the TLS and header settings with the Mozilla intermediate profile, a published set of TLS settings that balances strong security with broad client compatibility, plus a small set of response headers. Mozilla archived its SSL Configuration Generator in May 2026 and handed maintenance to

a community working group at configurator.tlsref.org; the profile names and settings continue there, and the settings below match the current guidelines.

```
server {
    listen 443 ssl;
    listen [::]:443 ssl;
    http2 on;                       # see the version note below
    server_name example.com www.example.com;

    ssl_certificate     /etc/letsencrypt/live/example.com/fullchain.pem;
    ssl_certificate_key /etc/letsencrypt/live/example.com/privkey.pem;

    # Mozilla intermediate profile; no dhparam file needed
    ssl_protocols TLSv1.2 TLSv1.3;
    ssl_ciphers ECDHE-ECDSA-AES128-GCM-SHA256:ECDHE-RSA-AES128-GCM-SHA256:ECDHE-ECDSA-AES256-GCM-
SHA384:ECDHE-RSA-AES256-GCM-SHA384:ECDHE-ECDSA-CHACHA20-POLY1305:ECDHE-RSA-CHACHA20-POLY1305;
    ssl_prefer_server_ciphers off;
    ssl_session_cache shared:MozSSL:10m;
    ssl_session_timeout 1d;

    add_header Strict-Transport-Security "max-age=63072000; includeSubDomains" always;
    add_header X-Content-Type-Options "nosniff" always;
    add_header X-Frame-Options "SAMEORIGIN" always;
    add_header Content-Security-Policy "frame-ancestors 'self';" always;
    add_header Referrer-Policy "strict-origin-when-cross-origin" always;

    server_tokens off;
    proxy_hide_header X-Powered-By;

    location / {
        proxy_pass http://backend;
        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_set_header X-Forwarded-Host  $host;
    }
}

server {
    listen 80;
    listen [::]:80;
    server_name example.com www.example.com;
    return 301 https://$host$request_uri;
}
```

- **There is no OCSP stapling in this config**
  Let's Encrypt stopped putting an OCSP URL in its certificates in May 2025 and shut its OCSP responders off in August 2025. ssl_stapling has nothing to fetch, so it is left out. Browsers now get revocation data from certificate revocation lists (CRLs) instead.

- **There is no dhparam file to generate**
  The current cipher list no longer includes the DHE ciphers that needed a Diffie-Hellman parameter file; the guidelines dropped them in version 5.8, so ssl_dhparam is not used.

- **Session tickets and the ECDH curve line**
  - ssl_session_tickets off is also gone. Since nginx 1.23.2, ticket encryption keys rotate automatically when the shared session cache (which this config sets) is in use, so the generator's current output no longer emits the line for modern nginx.
  - The generator's newest output does add an ssl_ecdh_curve list led by the post-quantum group X25519MLKEM768. That group needs nginx built against OpenSSL 3.5 or newer. Amazon Linux 2023 has shipped OpenSSL 3.5 since late March 2026, so current AL2023 supports it; stock Ubuntu 24.04 (OpenSSL 3.0) does not, which is why this shared config leaves the line out. Add it wherever your OpenSSL supports the group.

- **http2 on; needs nginx 1.25.1 or newer**

  Amazon Linux 2023 ships nginx 1.30.4, so it works there as written. Ubuntu 24.04 is on nginx 1.24.0, which predates the directive: on stock Ubuntu use the older form listen 443 ssl http2; instead, or add the official nginx.org repository to get a newer nginx.

- **Notes on the response headers**
  - HSTS: max-age=63072000 means HTTPS-only for two years. Add ; preload and submit to hstspreload.org only once every subdomain serves HTTPS, because preload is hard to undo.
  - X-XSS-Protection is deliberately absent: it is deprecated and can create vulnerabilities. CSP is the replacement.
  - proxy_hide_header X-Powered-By stops the backend's technology banner (PHP, Express, ASP.NET versions) from reaching clients. nginx already replaces the backend's Server header with its own, and server_tokens off trims that to just nginx.
  - An add_header inside a location block replaces the headers set in the parent server block instead of adding to them. If you add headers in a location, repeat the ones you still want.

Reload and test: sudo nginx -t && sudo systemctl reload nginx, then curl -sI https://example.com/ and check the headers. For an external grade, run Qualys SSL Labs or testssl.sh.

10 Step 8 · Automate renewal

Certbot only renews once a certificate has less than a third of its lifetime left (about 30 days on a 90-day certificate). Until then, renew runs do nothing, so a twice-daily schedule is safe.

Ubuntu, snap or apt

RENEWAL IS PRECONFIGURED AND RUNS TWICE A DAY

Add a reload hook so nginx picks up the new certificate, then test with a dry run. Certbot saves a hook only from a successful real issuance or renewal, never from a dry run, so attach it with reconfigure, which validates the change and writes it into the certificate's renewal config, where every future renewal reuses it; --run-deploy-hooks runs the reload once during that check, so a mistyped hook fails now instead of at the first real renewal. (Plain certbot --nginx from Path A already reloads nginx on renewal through its recorded installer; the hook matters on the certonly paths, and it does no harm to set it everywhere.)

```
sudo certbot reconfigure --cert-name example.com --deploy-hook "systemctl reload nginx" --run-deploy-hooks
sudo certbot renew --dry-run
```

Amazon Linux 2023, pip venv

THE PIP INSTALL CREATED NO TIMER · AL2023 HAS NO CRON EITHER: INSTALL CRONIE FIRST

AL2023 does not install cron by default, so the entry below does nothing until cronie is installed and crond is running. The random sleep spreads renewals out instead of having every server hit Let's Encrypt at exactly midnight and noon; it is part of certbot's own documented cron line. Confirm the renewal path works with a dry run before you rely on it.

```
sudo dnf install -y cronie && sudo systemctl enable --now crond
echo '0 0,12 * * * root /opt/certbot/bin/python -c "import random,time; time.sleep(random.random()*3600)" &&
/opt/certbot/bin/certbot renew --deploy-hook "systemctl reload nginx" -q' \
  | sudo tee /etc/cron.d/certbot-renew
sudo certbot renew --dry-run
```

- **Monitor expiry from outside the box**
  Let's Encrypt no longer emails you before expiry, so add monitoring that alerts on an approaching expiry from somewhere other than the machine doing the renewing.

11 Optional · Re-encrypt to the backend

Ending TLS at the proxy and sending plain HTTP over the private subnet is common, and the backend security group protects it here. If policy requires encryption all the way to the app (regulated data, zero trust, a shared network), have nginx open a TLS connection to the backend. Be aware that nginx does not verify the backend certificate unless you turn that on.

```
location / {
    proxy_pass https://backend;            # backend now listens on TLS, such as :8443
    proxy_ssl_verify              on;      # default is off; you must enable it
    proxy_ssl_trusted_certificate /etc/nginx/backend-ca.pem;
    proxy_ssl_name                backend.internal;
    proxy_ssl_server_name         on;      # send the server name (SNI); default is off
    proxy_ssl_protocols           TLSv1.3; # add TLSv1.2 only if the backend requires it
    # ... the same proxy_set_header lines as before ...
}
```

Update the upstream block's port to the backend's TLS port (8443 in this example). The backend now needs its own certificate and a CA the proxy trusts, issued by an internal CA or AWS Private CA. Public Let's Encrypt certificates are for the public name on the proxy, not for private backends. The protocol line is TLS 1.3 only because you control both ends of this connection; if the backend's stack cannot do TLS 1.3 yet, add TLSv1.2 and remove it once the backend can.

12 What you achieved

One hardened, internet-facing nginx proxy ends the HTTPS encryption with a current Mozilla intermediate profile and a Let's Encrypt certificate that renews itself.

The app runs in a private subnet with no public IP, reachable only from the proxy's security group, and managed through Session Manager with no bastion and no open SSH port.

The backend can be swapped out behind the proxy without touching the public entry point, and you can re-encrypt to the backend when policy calls for it.

13 Troubleshooting

These are the common failures and where to look first.

- **502 Bad Gateway**

  nginx cannot reach the backend. Check that the backend security group allows the app port from the proxy's security-group ID, that the app listens on the private IP (not only 127.0.0.1), and, on AL2023 in enforcing mode, that one of the SELinux booleans from Step 5 is set.

- **Certbot HTTP-01 fails to validate**

  Port 80 must be open to the internet on the proxy security group and reach nginx. Confirm the DNS A record points at the proxy's Elastic IP and has propagated.

- **Certbot DNS-01 fails with an access error**

  The IAM role is missing or lacks the Route 53 permissions from Step 6. Recheck the three actions and that the write action names your hosted-zone ID.

- **Certbot on AL2023 refuses to install or runs an old version**

  The venv was built with system Python 3.9. Rebuild it with python3.11 (Step 6).

- **The certificate is not renewing**

  On AL2023, confirm cronie is installed, crond is running, and the cron entry exists; on Ubuntu, check systemctl list-timers | grep certbot. Run sudo certbot renew --dry-run to see the actual failure.

- **Cannot reach the backend to administer it**

  It has no public IP by design. Use aws ssm start-session --target <instance-id>; if that fails, the instance is missing the AmazonSSMManagedInstanceCore role or has no network path to the SSM endpoints (NAT gateway or VPC endpoints).

- **http2 on; is rejected by nginx -t**

  Your nginx is older than 1.25.1 (stock Ubuntu 24.04 is 1.24.0). Use listen 443 ssl http2;, or install nginx from nginx.org. Amazon Linux 2023 (nginx 1.30.4) accepts http2 on; as written.

14 Framework alignment

How this design maps to two published security frameworks, checked against their current text.

AWS Well-Architected Framework, Security pillar

- SEC03-BP02 Grant least privilege access: the backend admits only the proxy's security group on the app port, and the DNS-01 role is scoped to one hosted zone.
- SEC05-BP01 Create network layers: the proxy is the internet-facing layer in a public subnet; the backend sits in a private subnet with no route to the internet gateway.
- SEC05-BP02 Control traffic flow within your network layers: traffic enters only at the proxy, leaves the private subnet only through the NAT gateway, and the backend flow is point-to-point from the proxy's security group.
- SEC09-BP02 Enforce encryption in transit: TLS ends at the proxy on the Mozilla intermediate profile, with the option to re-encrypt to the backend.
- Also: IMDSv2 closes the SSRF path to instance credentials, and administration runs through Session Manager instead of an open SSH port.

THIS GUIDE SETS	OWASP ASVS 5.0.0
HSTS, max-age two years with includeSubDomains	**3.4.1**
Content-Security-Policy with frame-ancestors 'self'	**3.4.6**
X-Content-Type-Options: nosniff	**3.4.4**
Referrer-Policy	**3.4.5**
TLS 1.2 and 1.3 with recommended cipher suites	**12.1.1, 12.1.2**
Publicly trusted TLS to external clients (Let's Encrypt)	**12.2.1, 12.2.2**

ASVS 3.4.6 treats X-Frame-Options as obsolete and relies on the CSP frame-ancestors directive, which this guide sets; the X-Frame-Options line is kept only for older browsers. ASVS 3.4.3 asks for a full Content-Security-Policy (object-src 'none', base-uri 'none', and an allowlist, nonces, or hashes); that belongs to the application behind the proxy, not to this frame-ancestors baseline.

REF References

nginx: ngx_http_proxy_module · nginx.org/en/docs/http/ngx_http_proxy_module.html

nginx: ngx_http_upstream_module (keepalive) · nginx.org/en/docs/http/ngx_http_upstream_module.html

nginx: ngx_http_v2_module (the http2 directive) · nginx.org/en/docs/http/ngx_http_v2_module.html

nginx official Linux packages · nginx.org/en/linux_packages.html

Mozilla SSL Configuration Generator (archived May 2026; maintained at configurator.tlsref.org) · configurator.tlsref.org

Certbot user guide · eff-certbot.readthedocs.io/en/stable/using.html

certbot-dns-route53 plugin · certbot-dns-route53.readthedocs.io/en/stable/

Let's Encrypt: certificate profiles · letsencrypt.org/docs/profiles/

Let's Encrypt: OCSP end of life · letsencrypt.org/2025/08/06/ocsp-service-has-reached-end-of-life/

Let's Encrypt: expiration notifications ended · letsencrypt.org/2025/06/26/expiration-notification-service-has-ended/

Let's Encrypt: rate limits · letsencrypt.org/docs/rate-limits/

AWS: subnets and routing · docs.aws.amazon.com/vpc/latest/userguide/configure-subnets.html

AWS: NAT gateways · docs.aws.amazon.com/vpc/latest/userguide/vpc-nat-gateway.html

AWS: security group rules · docs.aws.amazon.com/vpc/latest/userguide/security-group-rules.html

AWS: Session Manager · docs.aws.amazon.com/systems-manager/latest/userguide/session-manager.html

AWS: AMIs with the SSM Agent preinstalled · docs.aws.amazon.com/systems-manager/latest/userguide/ami-preinstalled-agent.html

AWS: EC2 Instance Connect Endpoint · docs.aws.amazon.com/AWSEC2/latest/UserGuide/connect-with-ec2-instance-connect-endpoint.html

AWS: use IMDSv2 · docs.aws.amazon.com/AWSEC2/latest/UserGuide/configuring-instance-metadata-service.html

AWS: public IPv4 address charge · aws.amazon.com/blogs/aws/new-aws-public-ipv4-address-charge-public-ip-insights/

Tailscale SSH · tailscale.com/kb/1193/tailscale-ssh

MDN: HTTP headers reference · developer.mozilla.org/en-US/docs/Web/HTTP/Reference/Headers

AWS Well-Architected, Security pillar · docs.aws.amazon.com/wellarchitected/latest/security-pillar/welcome.html

OWASP ASVS · owasp.org/www-project-application-security-verification-standard/