Ubuntu CLI (2026)

- Certbot Wildcards Certifiactes (OVH)
- Setup webserver (NGINX) on Ubuntu
- Set up subdomain routing NGINX
- Enable https NGINX

Certbot Wildcards Certifiactes (OVH)

Wildcard Let’s Encrypt Certificate with OVH DNS (Certbot)

0. Introduction

Goal

Obtain and automatically renew a wildcard TLS certificate (***.qool.ovh**) using Let’s Encrypt, with DNS hosted at OVH and the server hosted elsewhere (e.g. Contabo).

Key Requirements

Wildcard certificates require DNS-01 validation. This means Certbot must be able to create and remove DNS TXT records via the OVH API.

High-Level Overview

The process consists of:

1. Installing Certbot (Snap version)
2. Creating a correct OVH API token
3. Storing OVH credentials securely
4. Verifying API access with Python
5. Running Certbot with the OVH DNS plugin
6. Cleaning up and relying on auto-renewal
7. 1. Install Certbot (Snap)

1. Installing Certbot (Snap version)

Remove old packages

```
sudo apt remove certbot
```

Install Snap and Certbot

```
sudo apt update
sudo apt install snapd
sudo snap install core
sudo snap refresh core
sudo snap install --classic certbot
sudo ln -s /snap/bin/certbot /usr/bin/certbot
```

Install OVH DNS plugin

```
sudo snap set certbot trust-plugin-with-root=ok
sudo snap install certbot-dns-ovh
```

2. Create OVH API Token

Where

Create the token at:

https://api.ovh.com/createToken/

Token settings

Validity: **Unlimited**

Required permissions (exact)

```
GET    /domain/zone
GET    /domain/zone/
GET    /domain/zone/qool.ovh
GET    /domain/zone/qool.ovh/*
POST   /domain/zone/qool.ovh/*
```

```
PUT    /domain/zone/qool.ovh/*
DELETE /domain/zone/qool.ovh/*
```

Note: OVH treats `/domain/zone` and `/domain/zone/` as different paths. Certbot (Lexicon) requires the trailing slash permission.

wise.ovh, these are the API settings

```
GET    /domain/zone/
GET    /domain/zone/*
POST   /domain/zone/*
PUT    /domain/zone/*
DELETE /domain/zone/*
```

When API key is set, this can be executed:

```
sudo certbot certonly \
  --dns-ovh \
  --dns-ovh-credentials ~/.secrets/certbot/ovh.ini \
  -d wise.ovh -d '*.wise.ovh'
```

3. Store OVH Credentials Securely

Create credentials file

```
sudo nano /etc/letsencrypt/ovh.ini
```

File contents

```
dns_ovh_endpoint = ovh-eu
dns_ovh_application_key = YOUR_APPLICATION_KEY
dns_ovh_application_secret = YOUR_APPLICATION_SECRET
dns_ovh_consumer_key = YOUR_CONSUMER_KEY
```

Lock down permissions

```
sudo chmod 600 /etc/letsencrypt/ovh.ini
```

4. Verify OVH API Access (Before Certbot)

Create a small Python test

```
import ovh

client = ovh.Client(
    endpoint='ovh-eu',
    application_key='YOUR_APPLICATION_KEY',
    application_secret='YOUR_APPLICATION_SECRET',
    consumer_key='YOUR_CONSUMER_KEY'
)

print(client.get('/domain/zone'))
```

Expected output

A list of domains including **qool.ovh**. If this works, the API token and permissions are correct.

5. Request the Wildcard Certificate

Run Certbot

```
sudo certbot certonly \
  --dns-ovh \
  --dns-ovh-credentials /etc/letsencrypt/ovh.ini \
  --dns-ovh-propagation-seconds 120 \
  --agree-tos \
  --email admin@qool.ovh \
  -d "*.qool.ovh" \
  -d "qool.ovh"
```

Successful result

Certbot reports that the certificate was issued and stored in:

```
/etc/letsencrypt/live/qool.ovh/
```

6. Verify and Test Renewal

Check certificate files

```
sudo ls -l /etc/letsencrypt/live/qool.ovh/
```

Test auto-renewal

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

Snap installs a systemd timer automatically, so renewals run without manual action.

7. Cleanup (Recommended)

Remove duplicate credential copies

Keep only:

```
/etc/letsencrypt/ovh.ini
```

Remove test artifacts

```
rm -f ~/python/set-ovy.py
rm -rf ~/ovh-test
```

Final Notes

- DNS hosting location matters; web hosting location does not.
- Certbot OVH plugin requires both `/domain/zone` and `/domain/zone/` permissions.
- Protect the OVH API token like a root password.
- Once working, the setup is fully automatic and production-safe.

Setup webserver (NGINX) on Ubuntu

Nginx + PHP + MariaDB + phpMyAdmin + SSH

0. Introduction

Goal

Set up a production-ready Ubuntu server with: **Nginx**, **PHP (PHP-FPM)**, **MariaDB**, **phpMyAdmin**, and **SSH on a custom port**.

Assumptions

- Ubuntu Server (22.04+)
- CLI access
- User `max` with sudo rights

---

1. Secure SSH (Custom Port)

Edit SSH configuration

```
sudo nano /etc/ssh/sshd_config
```

Required settings

```
Port 1611
PermitRootLogin no
PasswordAuthentication yes
```

Apply changes

```
sudo ufw allow 1611/tcp
sudo systemctl restart ssh
```

Verify

```
sudo ss -tlnp | grep ssh
```

---

2. Install Nginx

Install

```
sudo apt update
sudo apt install nginx -y
```

Enable and start

```
sudo systemctl enable nginx
sudo systemctl start nginx
```

Allow firewall

```
sudo ufw allow 'Nginx Full'
```

---

3. Install PHP (PHP-FPM)

Install PHP and extensions

```
sudo apt install php-fpm php-cli php-mysql php-curl php-gd php-mbstring php-xml php-zip -y
```

Verify PHP

```
php -v
systemctl status php8.3-fpm
```

4. Configure Nginx + PHP

Default site config

```
sudo nano /etc/nginx/sites-available/default
```

Minimal PHP configuration

```
server {
    listen 80;
    server_name _;
    root /var/www/html;
    index index.php index.html;

    location / {
        try_files $uri $uri/ /index.php?$query_string;
    }

    location ~ \.php$ {
        include snippets/fastcgi-php.conf;
        fastcgi_pass unix:/run/php/php8.3-fpm.sock;
    }
}
```

Reload

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

5. Install MariaDB

Install

```
sudo apt install mariadb-server mariadb-client -y
```

Secure

```
sudo mysql_secure_installation
```

Create admin user

```
sudo mariadb
```

```
CREATE USER 'max'@'localhost' IDENTIFIED BY 'STRONG_PASSWORD';
GRANT ALL PRIVILEGES ON *.* TO 'max'@'localhost' WITH GRANT OPTION;
FLUSH PRIVILEGES;
EXIT;
```

---

6. Install phpMyAdmin

Install package

```
sudo apt install phpmyadmin -y
```

When prompted:

- Select no webserver
- Choose **Yes** for dbconfig-common

Nginx configuration

```
sudo nano /etc/nginx/snippets/phpmyadmin.conf
```

```
location /phpmyadmin {
    root /usr/share/;
    index index.php;

    location ~ \.php$ {
        include snippets/fastcgi-php.conf;
        fastcgi_pass unix:/run/php/php8.3-fpm.sock;
    }
}
```

Enable

```
sudo nano /etc/nginx/sites-available/default
```

Add:

```
include snippets/phpmyadmin.conf;
```

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

---

7. File Ownership & Permissions

Recommended structure

```
/var/www/qool/test
```

Ownership

```
sudo chown -R max:www-data /var/www/qool/test
```

Permissions

```
find /var/www/qool/test -type d -exec chmod 750 {} \;
find /var/www/qool/test -type f -exec chmod 640 {} \;
```

Final Notes

- Nginx handles HTTP, PHP-FPM executes PHP
- MariaDB uses socket authentication for root
- phpMyAdmin should be IP-restricted in production
- SSH on a custom port reduces automated attacks
- This setup is production-ready and extensible

Set up subdomain routing NGINX

Dynamic Subdomain Routing for *.qool.ovh (Nginx)

0. Introduction

Goal

Automatically route requests for `<subdomain>.qool.ovh` to the corresponding directory `/var/www/qool/<subdomain>`, without creating a new DNS record or Nginx config for each site.

Use case

- `test.qool.ovh` → `/var/www/qool/test`
- `blog.qool.ovh` → `/var/www/qool/blog`
- `demo.qool.ovh` → `/var/www/qool/demo`

---

1. DNS Configuration (Wildcard)

Create wildcard DNS record

At your DNS provider (OVH), create the following record:

```
Type: A
Name: *.qool.ovh
Value: <SERVER_IP>
TTL: Auto
```

This ensures that **all subdomains** of `qool.ovh` resolve to your server.

2. Directory Structure

Base webroot layout

```
/var/www/qool/
 ├─ test/
 │   └─ index.php
 ├─ blog/
 │   └─ index.php
 └─ demo/
     └─ index.php
```

Ownership and permissions

```
sudo chown -R max:www-data /var/www/qool
find /var/www/qool -type d -exec chmod 750 {} \;
find /var/www/qool -type f -exec chmod 640 {} \;
```

3. Nginx Wildcard Configuration

Create Nginx site file

```
sudo nano /etc/nginx/sites-available/qool.ovh
```

Wildcard server block

```
server {
    listen 80;

    # Capture subdomain name dynamically
    server_name ~^(?<site>[a-z0-9-]+)\.qool\.ovh$;

    root /var/www/qool/$site;
    index index.php index.html;

    # Reject non-existing site directories
    if (!-d /var/www/qool/$site) {
        return 404;
    }
```

```
    location / {
        try_files $uri $uri/ /index.php?$query_string;
    }

    location ~ \.php$ {
        include snippets/fastcgi-php.conf;
        fastcgi_pass unix:/run/php/php8.3-fpm.sock;
    }
}
```

Enable the site

```
sudo ln -s /etc/nginx/sites-available/qool.ovh /etc/nginx/sites-enabled/
sudo nginx -t
sudo systemctl reload nginx
```

---

4. How the Routing Works

Nginx logic

- Nginx matches `<subdomain>.qool.ovh` using a regex
- The subdomain is captured into variable `$site`
- The document root is set to `/var/www/qool/$site`
- If the directory does not exist, a **404** is returned

Example

Request:

```
GET http://test.qool.ovh
```

Results in:

```
root = /var/www/qool/test
```

---

5. Testing

Create a test file

```
nano /var/www/qool/test/index.php
```

```
<?php
echo "Site: test.qool.ovh";
```

Test locally

```
curl -H "Host: test.qool.ovh" http://localhost
```

Test in browser

Open:

```
http://test.qool.ovh
```

---

6. Notes & Best Practices

- No per-site Nginx configuration required
- Adding a new site = create a directory only
- Works with PHP, static files, and frameworks
- Compatible with wildcard TLS certificates
- Production-safe when directory existence is validated

Final Summary

- Wildcard DNS resolves all subdomains
- Nginx regex captures subdomain name
- Filesystem routing maps subdomain → directory
- Scales to unlimited sites with zero config changes

Enable https NGINX

Enable HTTPS on Nginx (Let’s Encrypt)

0. Introduction

Goal

Enable HTTPS on Nginx using an existing Let’s Encrypt certificate, open the HTTPS port, and (optionally) redirect all HTTP traffic to HTTPS.

---

1. Prerequisites

- Nginx installed and running
- Valid TLS certificate from Let’s Encrypt
- Certificate files available under `/etc/letsencrypt/live/<domain>/`

Required files:

```
/etc/letsencrypt/live/qool.ovh/fullchain.pem
/etc/letsencrypt/live/qool.ovh/privkey.pem
```

---

2. Allow HTTPS Port (443)

Add firewall rule

```
sudo ufw allow 443/tcp
```

Enable firewall (if not active)

```
sudo ufw enable
```

Verify

```
sudo ufw status
```

---

3. Configure Nginx for HTTPS

Edit Nginx site configuration

```
sudo nano /etc/nginx/sites-available/qool.ovh
```

HTTPS server block

```
server {
    listen 443 ssl http2;
    server_name *.qool.ovh;

    root /var/www/qool;
    index index.php index.html;

    ssl_certificate     /etc/letsencrypt/live/qool.ovh/fullchain.pem;
    ssl_certificate_key /etc/letsencrypt/live/qool.ovh/privkey.pem;

    ssl_protocols TLSv1.2 TLSv1.3;

    location / {
        try_files $uri $uri/ /index.php?$query_string;
    }

    location ~ \.php$ {
        include snippets/fastcgi-php.conf;
        fastcgi_pass unix:/run/php/php8.3-fpm.sock;
    }
}
```

---

4. Redirect HTTP to HTTPS (Recommended)

HTTP redirect block

```
server {
    listen 80;
    server_name *.qool.ovh;
    return 301 https://$host$request_uri;
}
```

---

5. Apply Configuration

Test configuration

```
sudo nginx -t
```

Reload Nginx

```
sudo systemctl reload nginx
```

---

6. Verify HTTPS

Check listening ports

```
sudo ss -tlnp | grep 443
```

Test in browser

```
https://test.qool.ovh
```

Test via CLI

```
curl -I https://test.qool.ovh
```

Final Notes

- Port 443 must be open in the firewall
- Nginx must reference the correct certificate paths
- Certbot renews certificates automatically
- No manual changes are required after renewal
- This setup supports wildcard subdomains