zurück

Webserver – Apache & Nginx – Grundlagen

Ein Webserver stellt Webseiten, APIs oder Anwendungen bereit und liefert HTTP- bzw. HTTPS-Inhalte an Clients wie Browser oder Apps aus.

Die beiden bekanntesten Webserver sind:

- **Apache HTTP Server**
- **Nginx (sprich: Engine-X)**

Beide sind weit verbreitet, aber unterschiedlich aufgebaut und eingesetzt.

Aufgaben eines Webservers

- Bereitstellen von Webseiten (HTML, CSS, JS)
- Ausliefern von Bildern, Downloads, Dateien
- Ausführen von Webanwendungen (PHP, Python, FastCGI)
- HTTPS-Verschlüsselung via TLS
- Reverse Proxy für Backend-Dienste
- Weiterleitungen, Rewrite-Regeln
- Logging von Zugriffen

Apache – Grundlagen

Apache ist modular aufgebaut und gilt als sehr flexibel.
Ideal, wenn viele Funktionen gebraucht werden.

Merkmale

- Prozessbasiert (jeder Client erzeugt eigenen Prozess/Thread)
- sehr flexibel dank vieler Module
- traditionell für PHP-Anwendungen (z. B. WordPress, Joomla)
- gute .htaccess-Unterstützung

Module (Beispiele):

- `mod_ssl` – TLS/HTTPS
- `mod_rewrite` – URL-Umschreibungen
- `mod_php` – PHP direkt ausführbar

Standardverzeichnisse (Debian/Ubuntu)

```
/etc/apache2/
/var/www/html/
```

Konfigurationen:

- `/etc/apache2/sites-available/*.conf`
- aktivieren via:

```
a2ensite sitename.conf
systemctl reload apache2
```

Beispiel Virtual Host

```
<VirtualHost *:80>
    ServerName example.com
    DocumentRoot /var/www/example
</VirtualHost>
```

Nginx – Grundlagen

Nginx ist moderner, schlanker und extrem performant.
Er ist besonders gut geeignet als:

- Reverse Proxy
- Load Balancer
- Webserver für statische Dateien

Merkmale

- ereignisbasiert (event-driven)
- sehr ressourcenschonend
- extrem leistungsfähig bei vielen Verbindungen
- bevorzugt für Microservices/Docker
- PHP per FastCGI statt direkt

Standardverzeichnisse (Debian/Ubuntu)

```
/etc/nginx/
/var/www/html/
```

Aktivierung von Sites:

```
ln -s /etc/nginx/sites-available/site.conf /etc/nginx/sites-enabled/
nginx -t
systemctl reload nginx
```

Beispiel Serverblock (Virtual Host)

```
server {
    listen 80;
    server_name example.com;
    root /var/www/example;

    location / {
        try_files $uri $uri/ =404;
    }
}
```

Apache vs. Nginx – Vergleich

Kategorie	Apache	Nginx
———————	——————————-	——————————-
Architektur	prozess-/threadbasiert	event-driven
Geschwindigkeit	mittel	sehr hoch
Ressourcenbedarf	höher	sehr gering
PHP-Unterstützung	direkt via mod_php	via PHP-FPM (FastCGI)
.htaccess	unterstützt	nicht unterstützt
guter Einsatzort	klassische Webanwendungen	Reverse Proxy, moderne Setups

Reverse Proxy – Beispiel Nginx

Nginx vor einem Backend (z. B. Docker-Service):

```
location /api/ {
    proxy_pass http://localhost:8080/;
}
```

HTTPS mit TLS (beide Server)

Beispiel (Nginx):

```
server {
    listen 443 ssl;
    ssl_certificate /etc/ssl/certs/fullchain.pem;
    ssl_certificate_key /etc/ssl/private/key.pem;
}
```

Apache:

```
<VirtualHost *:443>
    SSLEngine on
    SSLCertificateFile /etc/ssl/certs/fullchain.pem
    SSLCertificateKeyFile /etc/ssl/private/key.pem
</VirtualHost>
```

Logging

Apache:

```
/var/log/apache2/access.log
/var/log/apache2/error.log
```

Nginx:

```
/var/log/nginx/access.log
/var/log/nginx/error.log
```

Docker & Webserver

Nginx ist in Docker-Umgebungen extrem beliebt wegen:

- wenig RAM
- guter Performance
- Reverse-Proxy-Fähigkeiten

Apache wird oft in LAMP-Stacks genutzt.

Sicherheitshinweise

- TLS erzwingen (Redirect zu HTTPS)
- Prinzip „Least Privilege“
- keine .git- oder .env-Dateien ausliefern
- Rate-Limits setzen (Nginx)
- aktuelle Versionen nutzen
- WAF-Integration möglich (z. B. ModSecurity)

ASCII-Übersicht

```
Browser → HTTP/HTTPS → Webserver (Apache/Nginx) → Anwendung
```

Zusammenfassung

- Apache = flexibel, modular, ideal für klassische Websites
    - Nginx = schnell, modern, ideal für Reverse Proxy & Docker
      * beide liefern Webseiten, Anwendungen und APIs aus
      * TLS-Verschlüsselung ist Pflicht in modernen Umgebungen
      * zentrale Komponenten jeder Infrastruktur