Nginx & PHP-FPM Performance Tuning on Ubuntu VPS

Executive Summary

Running PHP applications efficiently on a Linux VPS requires systematic validation, tuning, and iterative optimization. This tutorial provides a **practical, engineering-focused guide** to testing and tuning **PHP memory limits, OPcache, Nginx FastCGI caching, kernel parameters, and application-level caching**. It is tailored for **Ubuntu-based VPS environments** commonly used in **engineering consulting, SaaS, and SME production workloads**.

The approach follows a layered optimization strategy:

1. Verify and baseline the stack
2. Tune PHP memory and PHP-FPM process management
3. Enable and optimize OPcache
4. Configure Nginx caching and workers
5. Apply compression, HTTP/2, and TLS optimizations
6. Tune kernel and network parameters
7. Add database and application-level caching
8. Measure, validate, and iterate under load

---

1. Prerequisites and Baseline Setup

1.1 Access and Stack Verification

Log into the VPS using SSH with sudo privileges:

```
ssh user@your-vps-ip
```

Verify installed components:

```
nginx -v
php -v
mysql --version
```

1.2 Install Required Packages (Ubuntu)

```
sudo apt update && sudo apt install -y \
nginx php-fpm php-mysql php-opcache php-redis \
mariadb-server redis-server htop curl
```

1.3 Backup Configuration Files

```
sudo cp -r /etc/nginx /etc/nginx.bak
sudo cp /etc/php/*/fpm/php.ini /etc/php/*/fpm/php.ini.bak
```

1.4 Create a Test PHP File

```
echo "<?php phpinfo(); ?>" | sudo tee /var/www/html/info.php
```

Access via browser:

```
http://your-vps-ip/info.php
```

> **Security note:** Remove this file immediately after testing.

```
sudo rm /var/www/html/info.php
```

1.5 Baseline Monitoring

```
htop
free -h
nginx -V
```

Record baseline CPU, RAM, and request latency before tuning.

---

2. Testing and Tuning PHP Memory Limits

2.1 Identify Current Memory Limit

Check `memory_limit` in `phpinfo()` (default often 128M).

2.2 Stress Test PHP Memory

Create a memory stress test:

```
<?php
$data = [];
for ($i = 0; $i < 2000000; $i++) {
    $data[] = str_repeat('x', 1024);
}
echo memory_get_peak_usage(true)/1024/1024 . " MB";
```

Increase loop count until a fatal error appears:

```
Allowed memory size exhausted
```

2.3 Adjust PHP Memory Limit

Edit:

```
sudo nano /etc/php/*/fpm/php.ini
```

Recommended guidelines:

VPS RAM	memory_limit
2 GB	128–256M
4 GB	256–512M
8 GB+	512M–1G

Example:

```
memory_limit = 512M
```

Restart PHP-FPM:

```
sudo systemctl restart php*-fpm
```

Target: **<70% total RAM usage under peak load**.

---

3. Enabling and Optimizing OPcache

OPcache significantly reduces PHP execution time by caching compiled bytecode.

3.1 OPcache Configuration

Edit `php.ini`:

```
zend_extension=opcache.so
opcache.enable=1
opcache.memory_consumption=256
opcache.interned_strings_buffer=16
opcache.max_accelerated_files=10000
opcache.validate_timestamps=0
opcache.revalidate_freq=0
opcache.save_comments=1
opcache.max_wasted_percentage=5
```

Restart services:

```
sudo systemctl restart php*-fpm nginx
```

3.2 Verification

Check `phpinfo()`:

- OPcache enabled: **Yes**
- Hit rate: **>95%**

3.3 Benchmarking

```
sudo apt install apache2-utils
ab -n 2000 -c 50 http://your-vps-ip/test.php
```

Typical gains: **3–5× throughput improvement**.

---

4. Nginx FastCGI and Static Caching

4.1 FastCGI Cache Configuration

```
fastcgi_cache_path /var/cache/nginx \
levels=1:2 keys_zone=PHP:100m max_size=2g inactive=60m use_temp_path=off;

location ~ \.php$ {
    fastcgi_cache PHP;
    fastcgi_cache_valid 200 302 60m;
    fastcgi_cache_bypass $http_cookie $arg_nocache;
    fastcgi_cache_use_stale error timeout invalid_header updating http_500;
    fastcgi_pass unix:/run/php/php8.2-fpm.sock;
    include fastcgi_params;
    fastcgi_param SCRIPT_FILENAME $document_root$fastcgi_script_name;
}
```

4.2 Static Asset Caching

```
location ~* \.(jpg|jpeg|png|gif|ico|css|js|svg|webp)$ {
    expires 30d;
    add_header Cache-Control "public, immutable";
    access_log off;
}
```

4.3 Validation

```
curl -I http://your-vps-ip/index.php
```

Look for:

```
X-Cache-Status: HIT
```

---

5. Core Nginx Worker Optimization

Edit `/etc/nginx/nginx.conf`:

```
worker_processes auto;
worker_rlimit_nofile 65535;

events {
    worker_connections 4096;
    multi_accept on;
    use epoll;
}

http {
    keepalive_timeout 30;
    keepalive_requests 1000;
    sendfile on;
    tcp_nopush on;
    tcp_nodelay on;
    client_max_body_size 100m;
    server_tokens off;
}
```

Apply:

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

Result: **~2× concurrency improvement** on multi-core VPS.

---

6. Compression, TLS, and HTTP/2

6.1 Gzip Compression

```
gzip on;
gzip_vary on;
gzip_proxied any;
gzip_comp_level 6;
gzip_types text/plain text/css application/javascript application/json
image/svg+xml;
```

6.2 TLS and HTTP/2

```
listen 443 ssl http2;
ssl_protocols TLSv1.2 TLSv1.3;
ssl_ciphers ECDHE-AES256-GCM-SHA384:ECDHE-AES128-GCM-SHA256;
```

6.3 Let’s Encrypt

```
sudo apt install certbot python3-certbot-nginx
sudo certbot --nginx
```

Bandwidth savings: **50–70%**.

---

7. Kernel and Network Optimization

Edit `/etc/sysctl.conf`:

```
net.core.somaxconn=65535
net.ipv4.tcp_max_syn_backlog=8192
net.ipv4.tcp_fin_timeout=15
net.core.default_qdisc=fq
net.ipv4.tcp_congestion_control=bbr
vm.swappiness=10
```

Apply:

```
sudo sysctl -p
```

Enable BBR:

```
sudo modprobe tcp_bbr
```

Expected throughput improvement: **20–50%**.

---

8. PHP-FPM Pool Tuning

Edit `/etc/php/*/fpm/pool.d/www.conf`:

```
pm = dynamic
pm.max_children = 20
pm.start_servers = 5
pm.min_spare_servers = 5
pm.max_spare_servers = 10
```

Rule of thumb:

```
pm.max_children × memory_limit < 80% of system RAM
```

---

9. Database and Application Caching

9.1 Redis Sessions and Object Cache

```
session.save_handler = redis
session.save_path = "tcp:127.0.0.1:6379"
```

9.2 MariaDB Optimization

```
innodb_buffer_pool_size = 2G
query_cache_type = 1
query_cache_size = 64M
```

Restart services:

```
sudo systemctl restart mariadb redis-server php*-fpm
```

---

10. Monitoring, Load Testing, and KPIs

10.1 Monitoring Tools

```
tail -f /var/log/nginx/access.log
htop
glances
```

10.2 Load Testing

```
wrk -t12 -c200 -d60s http://your-vps-ip/
```

10.3 Target Metrics

Metric	Target
P95 Latency	<150 ms
CPU Usage	<70%
RAM Usage	<70%
OPcache Hit Ratio	>98%

Metric	Target
FastCGI Cache HIT	>90%

Iterate using controlled A/B testing and configuration changes.

Conclusion

By combining **PHP memory tuning, OPcache optimization, Nginx caching, kernel tuning, and application-level caching**, Ubuntu-based VPS environments can reliably support **high-performance PHP workloads** used in engineering consulting, SaaS, and SME production systems. This layered methodology ensures predictable scaling, efficient resource usage, and measurable performance gains.

For global traffic and static offloading, integrate a CDN such as **Cloudflare** to further reduce origin load and latency.