I use plesk. My web server is unresponsive. I am aware that I have high traffic. These are the php errors. Can you guide me?

[28-Jan-2025 16:13:32] WARNING: [pool ride-eats.com] seems busy (you may need to increase pm.start\_servers, or pm.min/max\_spare\_servers), spawning 32 children, there are 0 idle, and 181 total children

ChatGPT said:

ChatGPT

The PHP error log indicates that your web server is under heavy load, and your PHP-FPM (FastCGI Process Manager) configuration isn't optimized for the level of traffic you're experiencing. Here’s how you can address this issue and make your server more responsive:

1. Understand the Error

The log shows that your server has reached the maximum number of PHP-FPM children processes (181 total), and there are no idle processes available to handle incoming requests.

This results in requests being queued, leading to unresponsiveness.

2. Optimize PHP-FPM Configuration

You need to adjust the pm settings for your PHP-FPM pool. These settings control how PHP-FPM handles incoming requests.

Open the PHP-FPM configuration for the affected domain (e.g., ride-eats.com):

The configuration file is typically located at /etc/php-fpm.d/[pool-name].conf or /opt/plesk/php/[version]/etc/php-fpm.d/[pool-name].conf.

Replace [pool-name] with the name of your domain or pool.

Look for the following parameters and adjust them:

pm = dynamic

pm.max\_children = 250 ; Increase this to allow more concurrent requests.

pm.start\_servers = 50 ; Increase to handle initial burst of traffic.

pm.min\_spare\_servers = 25 ; Set minimum idle processes.

pm.max\_spare\_servers = 75 ; Set maximum idle processes.

pm.process\_idle\_timeout = 10s ; Ensure idle processes terminate after 10 seconds.

pm.max\_requests = 500 ; Restart PHP-FPM processes after handling 500 requests (prevents memory leaks).

Notes:

Increase pm.max\_children based on your server's resources (CPU and RAM).

Make sure the new settings don't exceed your server’s capacity.

3. Restart PHP-FPM

After modifying the configuration file, restart PHP-FPM to apply the changes:

sudo systemctl restart php-fpm

For specific PHP versions in Plesk:

sudo systemctl restart plesk-phpXX-fpm

Replace XX with your PHP version, e.g., 81 for PHP 8.1.

4. Monitor Resource Usage

Use commands like top, htop, or ps to monitor CPU and memory usage.

Check the number of PHP-FPM processes:

ps aux | grep php-fpm

5. Enable OPcache

If not already enabled, OPcache can significantly improve performance by caching precompiled PHP scripts. In Plesk:

Go to Tools & Settings > PHP Settings > Select your PHP version.

Enable OPcache and configure it.

Recommended OPcache settings:

opcache.enable=1

opcache.memory\_consumption=256

opcache.max\_accelerated\_files=10000

opcache.validate\_timestamps=0

6. Optimize the Web Server

You can also optimize the web server (Apache/Nginx) configuration:

If you’re using Nginx as a reverse proxy in front of Apache, increase the worker\_processes and worker\_connections in the Nginx configuration file (e.g., /etc/nginx/nginx.conf):

worker\_processes auto;

events {

worker\_connections 1024;

}

Restart Nginx:

sudo systemctl restart nginx

7. Implement Caching

To further reduce PHP-FPM load:

Use caching mechanisms like Redis or Memcached.

Enable full-page caching for static content (if supported by your application).

8. Upgrade Server Resources

If your current server cannot handle the load:

Increase your server’s CPU and RAM.

Consider load balancing if traffic continues to grow.

9. Test and Monitor

Test your site under simulated load to ensure the changes have improved performance.

Continuously monitor logs and traffic patterns using tools like Plesk's monitoring panel or external services (e.g., New Relic, Datadog).

Let me know if you encounter any issues while applying these changes!

You said:

Are you not interested what my current config is instead of just giving me random configuration on assumptions?

ChatGPT said:

ChatGPT

Je hebt helemaal gelijk. Het is inderdaad verstandiger om eerst jouw huidige configuratie te begrijpen voordat er zomaar aanpassingen worden voorgesteld. Dit zorgt ervoor dat de adviezen nauwkeurig zijn en afgestemd op jouw specifieke situatie. Kun je jouw huidige PHP-FPM configuratie delen, zodat we samen kunnen bekijken wat er aangepast moet worden?

Hier zijn de specifieke instellingen die nuttig kunnen zijn om te delen:

De huidige configuratie van PHP-FPM voor de specifieke pool: Je kunt dit bestand meestal vinden in een van de volgende paden:

/etc/php-fpm.d/[pool-name].conf

/opt/plesk/php/[versie]/etc/php-fpm.d/[pool-name].conf

Deel de waarden van de volgende parameters:

pm

pm.max\_children

pm.start\_servers

pm.min\_spare\_servers

pm.max\_spare\_servers

pm.max\_requests

De serverinformatie:

Hoeveel RAM en CPU's heeft de server?

Welke versie van PHP gebruik je?

De belasting van de server:

Hoeveel bezoekers heb je gemiddeld op het moment van piekverkeer?

Wat is de huidige CPU- en RAM-belasting tijdens hoge verkeerspieken? (Met tools zoals top of htop kun je dit controleren.)

Andere relevante logs:

Andere foutmeldingen in de PHP of webserver logs (bijvoorbeeld Nginx of Apache logs).

Met deze informatie kan ik een veel nauwkeuriger advies geven. Laat maar weten wat je kunt delen!

You said:

Parameters:

pm = dynamic

pm.max\_children = 777

pm.max\_requests = 500

pm.max\_spare\_servers = 150

pm.min\_spare\_servers = 50

pm.process\_idle\_timeout = 10s

pm.start\_servers = 100

PHP versie: 7.4

Server is op lightsail:

32gb

640 gb ssd

8vcpu

Ik weet niet hoeveel bezoekers ik heb maar ik weet dat er ongeveer 1000 php processes open zijn in htop

CPU is niet echt hoog. Misschien 30% op alle cpu's. Ram is ongeveer 4-5 gb

/etc/apache2/mods-enabled/mpm\_event.conf:

<IfModule mpm\_event\_module>

StartServers 2

MinSpareThreads 100

MaxSpareThreads 400

ThreadLimit 64

ThreadsPerChild 50

MaxRequestWorkers 1400

ServerLimit 400

MaxConnectionsPerChild 1600

</IfModule>

ChatGPT said:

ChatGPT

Bedankt dat je de huidige configuratie deelt. Met deze informatie kunnen we een veel beter beeld krijgen van de situatie en specifieke aanpassingen voorstellen. Je configuratie laat zien dat je server redelijk krachtig is (32GB RAM, 8 vCPU’s), maar het aantal actieve PHP-processen (1000) is uitzonderlijk hoog, terwijl je CPU- en RAM-gebruik relatief laag blijven. Dit suggereert dat er mogelijk een inefficiëntie of een bottleneck in de configuratie of applicatie is.

Hier is wat ik aanraad op basis van jouw situatie:

1. PHP-FPM Optimalisatie

Hoewel je een hoog aantal pm.max\_children hebt (777), lijkt het erop dat het nog steeds niet voldoende is voor de werkdruk. Echter, omdat je CPU en RAM niet volledig benut worden, kun je het aantal processen waarschijnlijk verhogen of efficiënter inzetten.

Aanpassing PHP-FPM configuratie: Overweeg de volgende wijzigingen:

pm.max\_children = 1000 ; Verhoog het aantal processen om beter om te gaan met het huidige verkeer.

pm.start\_servers = 150 ; Start meer processen bij het opstarten.

pm.min\_spare\_servers = 100 ; Zorg dat er altijd voldoende idle processen zijn.

pm.max\_spare\_servers = 300 ; Bied meer ruimte voor fluctuaties in verkeer.

pm.process\_idle\_timeout = 5s ; Verkort de idle timeout, zodat processen sneller worden beëindigd.

pm.max\_requests = 300 ; Verlaag om geheugenlekken te voorkomen bij langdurige processen.

Waarom deze aanpassingen? Je hebt veel RAM beschikbaar, dus je kunt veilig meer processen draaien. Door de pm.process\_idle\_timeout te verkorten en pm.max\_requests te verlagen, worden inefficiënte of vastgelopen processen sneller beëindigd.

Belangrijk: Na deze aanpassingen moet je PHP-FPM herstarten:

sudo systemctl restart php7.4-fpm

2. Apache MPM Configuratie

Je Apache MPM Event configuratie lijkt al redelijk efficiënt, maar het kan worden afgestemd op de PHP-FPM pool:

Aanpassing Apache configuratie: Pas de onderstaande parameters aan in /etc/apache2/mods-enabled/mpm\_event.conf:

StartServers 4

MinSpareThreads 200

MaxSpareThreads 600

ThreadsPerChild 75

MaxRequestWorkers 1800

MaxConnectionsPerChild 2000

Waarom deze wijzigingen?

Verhoog de ThreadsPerChild en MaxRequestWorkers om meer gelijktijdige verzoeken aan te kunnen.

Verhoog MaxConnectionsPerChild om processen minder vaak te herstarten.

Herstart Apache na de wijzigingen:

sudo systemctl restart apache2

3. Analyse van de PHP-processen

Het feit dat je 1000 PHP-processen ziet in htop kan wijzen op inefficiënte code of te lange verwerkingstijden. Controleer het volgende:

Langlopende scripts:

Gebruik slowlog in PHP-FPM om langzame scripts te identificeren: Voeg dit toe in de PHP-FPM poolconfiguratie:

request\_slowlog\_timeout = 5s

slowlog = /var/log/php7.4-fpm-slow.log

Herstart PHP-FPM en controleer vervolgens het logbestand:

sudo systemctl restart php7.4-fpm

tail -f /var/log/php7.4-fpm-slow.log

Inefficiënte queries: Als jouw applicatie een database gebruikt, controleer dan of er query’s zijn die lang duren of veel bronnen verbruiken.

4. Caching Implementeren

Als je geen caching gebruikt, kan dit een aanzienlijke belasting op PHP verminderen. Overweeg:

OPcache: Zorg dat OPcache is ingeschakeld met voldoende geheugen (controleer via Plesk of voeg het handmatig toe):

opcache.enable=1

opcache.memory\_consumption=512

opcache.max\_accelerated\_files=20000

opcache.validate\_timestamps=0

Full-page caching: Gebruik caching op applicatieniveau (bijv. Redis of Memcached).

5. Load Balancing Overwegen

Als je verkeer blijft toenemen, kan een enkele server een bottleneck worden. Overweeg het volgende:

Zet een extra Lightsail-instantie in als webserver en gebruik een load balancer om het verkeer te verdelen.

Amazon Lightsail heeft een ingebouwde load balancing-optie.

6. Monitoring

Houd na de wijzigingen je serverprestaties in de gaten:

Gebruik tools zoals htop of atop om de belasting van CPU, RAM en processen te controleren.

Controleer de PHP-FPM logs en Apache logs regelmatig op foutmeldingen of problemen.

7. Eventuele Toepassingsproblemen

Als je serverconfiguratie goed lijkt, maar je toch zoveel processen hebt, kan er een probleem zijn met de applicatie zelf, zoals:

Oneindige loops of inefficiënte algoritmes.

Overbodige API-aanroepen of externe verzoeken.

Zorg ervoor dat je applicatie code wordt geoptimaliseerd als bovenstaande wijzigingen geen significant verschil maken.

Laat me weten hoe de voorgestelde aanpassingen werken en of je nog meer hulp nodig hebt!

You said:

2\. \*\*Apache MPM Configuratie\*\*

Is deze configuratie gelijk met de PHP-FPM configuratie?

ChatGPT said:

ChatGPT

Nee, de configuratie van Apache MPM en PHP-FPM zijn verschillend en hebben elk een aparte rol in het verwerken van verzoeken:

Apache MPM Configuratie

Apache MPM (Multi-Processing Module) beheert hoe Apache verzoeken afhandelt, zoals verbindingen met clients (bijvoorbeeld browsers).

Het regelt threading, processen, en hoeveel gelijktijdige verbindingen de webserver aankan.

Dit bepaalt hoe Apache communiceert met PHP-FPM via FastCGI.

Belangrijke parameters in Apache MPM:

StartServers, MinSpareThreads, MaxSpareThreads: Regelen hoeveel threads of processen beschikbaar zijn om verbindingen te accepteren.

MaxRequestWorkers: Het maximum aantal gelijktijdige verbindingen dat Apache aankan.

MaxConnectionsPerChild: Het maximum aantal verzoeken dat een Apache-proces verwerkt voordat het wordt herstart.

Doel: Apache verwerkt HTTP(S)-verzoeken en stuurt PHP-verzoeken door naar PHP-FPM via FastCGI.

PHP-FPM Configuratie

PHP-FPM (FastCGI Process Manager) beheert hoe PHP-scripts worden uitgevoerd.

Het draait processen die PHP-code uitvoeren en communiceert met Apache via een FastCGI-module.

Belangrijke parameters in PHP-FPM:

pm (process manager): Bepaalt hoe PHP-FPM processen beheert (static, dynamic, of ondemand).

pm.max\_children: Het maximum aantal PHP-processen dat tegelijkertijd kan draaien.

pm.start\_servers, pm.min\_spare\_servers, pm.max\_spare\_servers: Bepalen hoe processen worden opgestart en beheerd.

pm.max\_requests: Bepaalt hoeveel verzoeken een proces verwerkt voordat het opnieuw wordt gestart.

Doel: PHP-FPM voert de PHP-code uit die door Apache wordt doorgestuurd.

Relatie tussen Apache MPM en PHP-FPM

Apache MPM bepaalt hoeveel verbindingen met clients (zoals browsers) het aankan.

Voor PHP-verzoeken stuurt Apache deze door naar PHP-FPM.

PHP-FPM verwerkt de PHP-code en stuurt het resultaat terug naar Apache.

Apache geeft het resultaat door aan de client.

Waarom zijn beide configuraties belangrijk?

Als Apache meer verzoeken toestaat (MaxRequestWorkers) dan PHP-FPM aankan (pm.max\_children), zullen verzoeken in de wachtrij blijven en kan de server traag reageren of time-outs geven.

Beide configuraties moeten goed op elkaar worden afgestemd, afhankelijk van je serverbronnen (CPU en RAM) en verkeersniveau.

Aanbeveling

Zorg ervoor dat de waarde van MaxRequestWorkers in Apache MPM niet hoger is dan het aantal gelijktijdige processen dat PHP-FPM aankan (pm.max\_children).

In jouw geval stel ik voor dat je MaxRequestWorkers (Apache) aanpast aan de hogere waarde van pm.max\_children (PHP-FPM) of iets lager houdt, bijvoorbeeld:

MaxRequestWorkers = 1000

Hiermee zorg je ervoor dat er geen onevenwicht ontstaat tussen Apache en PHP-FPM.

Laat me weten of je meer hulp nodig hebt bij het verder afstemmen van deze configuraties!

You said:

Dus de MPM configuratie wat je aangaf is dus correct?

ChatGPT said:

ChatGPT

De MPM-configuratie die ik eerder voorstelde is in grote lijnen correct, maar omdat jouw PHP-FPM configuratie al goed is ingesteld (met een hoge waarde voor pm.max\_children van 777) en jouw server voldoende krachtig is, wil ik de configuratie van Apache MPM nog iets nauwkeuriger afstemmen op jouw specifieke situatie.

Afgestemde Apache MPM-configuratie

Hier is een aangepaste versie van de MPM-configuratie, specifiek afgestemd op jouw huidige PHP-FPM instellingen en serverbronnen:

<IfModule mpm\_event\_module>

StartServers 4 # Start iets meer servers om pieken op te vangen.

MinSpareThreads 200 # Zorg voor voldoende threads om idle te blijven.

MaxSpareThreads 600 # Voldoende threads om verkeer op te vangen.

ThreadsPerChild 75 # Meer threads per child voor betere schaalbaarheid.

MaxRequestWorkers 1000 # Houd dit op max. 1000, afgestemd op PHP-FPM.

ServerLimit 15 # Bepaalt het maximum aantal child processen.

MaxConnectionsPerChild 2000 # Beperk het herstarten van processen om overhead te verminderen.

</IfModule>

Waarom deze wijzigingen?

ThreadsPerChild: Het verhogen van ThreadsPerChild naar 75 zorgt ervoor dat één Apache-proces meer verzoeken aankan, wat efficiënter is.

MaxRequestWorkers: Deze waarde is nu beperkt tot 1000, wat past bij de limiet van jouw PHP-FPM (pm.max\_children = 777) en geeft nog wat ruimte voor statische verzoeken (zoals CSS en JS).

MaxConnectionsPerChild: Door deze hoger te zetten (2000), worden Apache-processen minder vaak herstart, wat overhead vermindert.

Evenwicht tussen Apache en PHP-FPM: Deze configuratie zorgt ervoor dat beide goed samenwerken zonder dat een van de twee een bottleneck wordt.

Houd dit in gedachten

Als je later merkt dat de server nog steeds ongebruikte bronnen heeft (bijvoorbeeld veel vrije CPU of RAM), kun je de waarden voor MaxRequestWorkers en pm.max\_children verder verhogen.

Herstart Apache na de wijzigingen:

sudo systemctl restart apache2

Volgende stap: Monitoring

Na het aanbrengen van deze wijzigingen:

Controleer regelmatig de belasting met htop of atop.

Bekijk of de wachtrijen voor PHP-processen (in htop) verminderen.

Controleer of er geen Apache time-outs optreden in de error logs:

sudo tail -f /var/log/apache2/error.log

Laat weten of je nog verdere optimalisaties nodig hebt!

You said:

\*\*Afgestemde Apache MPM-configuratie\*\*

Ik heb de PHP-FPM configuratie gebruikt die je me gaf. De server moet nu beter functioneren. Dit is wat je daarnet gaf:

pm.max\_children = 1000 ; Verhoog het aantal processen om beter om te gaan met het huidige verkeer.

pm.start\_servers = 150 ; Start meer processen bij het opstarten.

pm.min\_spare\_servers = 100 ; Zorg dat er altijd voldoende idle processen zijn.

pm.max\_spare\_servers = 300 ; Bied meer ruimte voor fluctuaties in verkeer.

pm.process\_idle\_timeout = 5s ; Verkort de idle timeout, zodat processen sneller worden beëindigd.

pm.max\_requests = 300 ; Verlaag om geheugenlekken te voorkomen bij langdurige processen.

ChatGPT said:

ChatGPT

Goed dat je de wijzigingen hebt toegepast! De PHP-FPM configuratie die je hebt gebruikt lijkt een goede stap in de richting van een beter functionerende server, zeker gezien de verkeersniveaus en serverbronnen die je hebt. Om de configuratie nu volledig op elkaar af te stemmen, stel ik voor om de Apache MPM-configuratie die ik eerder heb gedeeld als volgt te gebruiken:

Afgestemde Apache MPM-configuratie (opnieuw)

Hier nogmaals de configuratie met een toelichting waarom deze waarden goed werken samen met jouw PHP-FPM configuratie:

<IfModule mpm\_event\_module>

StartServers 4 # Begin met meer child-processen voor sneller opstarten.

MinSpareThreads 200 # Houd genoeg idle threads voor een vlotte respons.

MaxSpareThreads 600 # Laat voldoende threads beschikbaar voor pieken.

ThreadsPerChild 75 # Elk proces kan meer threads (verzoeken) afhandelen.

MaxRequestWorkers 1000 # Past goed bij jouw `pm.max\_children = 1000`.

ServerLimit 15 # Zorgt dat Apache niet meer processen start dan nodig.

MaxConnectionsPerChild 2000 # Herstart processen minder vaak om overhead te verlagen.

</IfModule>

Waarom werkt dit samen met de PHP-FPM instellingen?

MaxRequestWorkers = 1000:

Deze waarde sluit aan bij de maximale capaciteit van PHP-FPM (pm.max\_children = 1000). Hierdoor kan Apache gelijktijdig maximaal 1000 PHP-verzoeken naar PHP-FPM sturen zonder dat een wachtrij ontstaat.

ThreadsPerChild = 75:

Elk Apache child-proces kan meer verbindingen verwerken, wat handig is bij piekverkeer.

MaxConnectionsPerChild = 2000:

Door processen minder vaak te herstarten, voorkom je onnodige overhead en verlies je geen tijd met het opnieuw opbouwen van verbindingen.

Balans tussen Apache en PHP-FPM:

Beide systemen zijn nu beter in balans, waardoor wachtrijen en time-outs minimaal zouden moeten zijn.

Wat nu?

Herstart Apache en PHP-FPM om de wijzigingen door te voeren:

sudo systemctl restart apache2

sudo systemctl restart php7.4-fpm

Monitor je server:

Gebruik htop of atop om de belasting op CPU, RAM en het aantal processen in de gaten te houden.

Controleer de PHP-FPM logs:

sudo tail -f /var/log/php7.4-fpm.log

Controleer de Apache logs:

sudo tail -f /var/log/apache2/error.log

Langetermijnoptimalisatie:

Blijf kijken of de server alle bronnen optimaal gebruikt. Als je nog veel vrije CPU en RAM hebt, kun je eventueel MaxRequestWorkers of pm.max\_children verder verhogen.

Overweeg cachingmechanismen zoals Redis of OPcache als je die nog niet hebt ingeschakeld.

Laat weten hoe je server zich gedraagt met deze instellingen of als je verdere hulp nodig hebt!

You said:

<IfModule mpm\\_event\\_module> StartServers 4 # Begin met meer child-processen voor sneller opstarten. MinSpareThreads 200 # Houd genoeg idle threads voor een vlotte respons. MaxSpareThreads 600 # Laat voldoende threads beschikbaar voor pieken. ThreadsPerChild 75 # Elk proces kan meer threads (verzoeken) afhandelen. MaxRequestWorkers 1000 # Past goed bij jouw \`pm.max\\_children = 1000\`. ServerLimit 15 # Zorgt dat Apache niet meer processen start dan nodig. MaxConnectionsPerChild 2000 # Herstart processen minder vaak om overhead te verlagen. </IfModule>

Wat about ThreadLimit?

ChatGPT said:

ChatGPT

De ThreadLimit is een belangrijke parameter in de Apache MPM Event-module omdat deze het maximale aantal threads per child-proces instelt. Het fungeert als een harde limiet voor hoeveel threads per child-proces kunnen worden toegewezen.

Wat is het doel van ThreadLimit?

Harde limiet: ThreadLimit bepaalt de maximale waarde van ThreadsPerChild. Als je ThreadsPerChild hoger instelt dan de waarde van ThreadLimit, zal Apache dit negeren en het terugzetten naar de waarde van ThreadLimit.

Het wordt meestal gecombineerd met andere parameters zoals ThreadsPerChild en ServerLimit.

Wat stel je in voor ThreadLimit?

In jouw configuratie gebruik je ThreadsPerChild = 75, dus je moet ervoor zorgen dat ThreadLimit minstens 75 of hoger is. Als je ThreadsPerChild later wilt verhogen, is het verstandig om een iets hogere waarde in te stellen voor flexibiliteit.

Voor jouw server stel ik het volgende voor:

<IfModule mpm\_event\_module>

StartServers 4

MinSpareThreads 200

MaxSpareThreads 600

ThreadsPerChild 75

MaxRequestWorkers 1000

ServerLimit 15

MaxConnectionsPerChild 2000

ThreadLimit 100 # Stel dit in iets boven ThreadsPerChild.

</IfModule>

Waarom ThreadLimit = 100?

Het biedt voldoende ruimte voor toekomstige schaalbaarheid.

Je voorkomt dat je beperkt wordt door een te lage waarde van ThreadLimit als je ThreadsPerChild wilt verhogen.

Het blijft binnen de mogelijkheden van jouw krachtige server.

Hoe werkt het samen met MaxRequestWorkers?

De MaxRequestWorkers-waarde bepaalt het totale aantal gelijktijdige verzoeken dat Apache aankan. Dit wordt berekend als volgt:

MaxRequestWorkers=ThreadsPerChild×ServerLimit

MaxRequestWorkers=ThreadsPerChild×ServerLimit

Met de bovenstaande configuratie:

ThreadsPerChild = 75

ServerLimit = 15

MaxRequestWorkers=75×15=1125

MaxRequestWorkers=75×15=1125

Omdat je MaxRequestWorkers = 1000 hebt ingesteld, gebruik je iets minder dan de maximale capaciteit, wat zorgt voor een goede balans en voorkomt overbelasting.

Wat nu?

Voeg ThreadLimit toe aan je configuratie.

Herstart Apache:

sudo systemctl restart apache2

Controleer na deze aanpassingen of alles soepel werkt, en laat me weten of je meer vragen hebt!

You said:

worker\\_processes auto; events { worker\\_connections 1024; }

worker\_processes 8;

events {

worker\_connections 10240;

}

Is this enough?

ChatGPT said:

ChatGPT

Your current configuration of:

worker\_processes 8;

events {

worker\_connections 10240;

}

is a strong setup, but let's analyze it based on your server's specifications and traffic load.

Breakdown of the Configuration

worker\_processes 8:

This defines the number of worker processes Nginx spawns. Setting it to 8 matches your 8 vCPUs, which is optimal for CPU utilization.

However, you could also set it to auto, which automatically uses the number of available CPU cores:

worker\_processes auto;

This ensures you always use the maximum number of CPU cores available, even if you later upgrade your server.

worker\_connections 10240:

Each worker process can handle up to 10,240 simultaneous connections.

The total number of connections your server can handle is calculated as: \text{Total connections} = \text{worker\_processes} \times \text{worker\_connections} With your current configuration:

8×10240=81,920 simultaneous connections

8×10240=81,920 simultaneous connections This is more than enough for most high-traffic websites. However, keep in mind that each connection includes HTTP keep-alives, static files, and back-end proxy connections, so the real number of clients your server can handle may be lower.

Recommendations

Your configuration is already strong, but here are some additional points to ensure it's optimized:

1. Enable Epoll (Linux-specific)

Epoll is a more efficient event mechanism on Linux and is often enabled by default. To explicitly enable it:

events {

use epoll;

worker\_connections 10240;

}

2. Optimize Keep-Alive Settings

Reduce unnecessary connections by optimizing keep-alive:

keepalive\_timeout 65;

keepalive\_requests 100;

keepalive\_timeout: Reduces idle time for persistent connections.

keepalive\_requests: Limits the number of requests per connection to prevent long-lived connections.

3. Monitor Resource Usage

Ensure your configuration isn’t overloading the server:

CPU Usage: Since worker\_processes is set to 8, you should monitor if all CPU cores are utilized efficiently.

RAM Usage: The number of simultaneous connections affects RAM usage. Monitor with:

free -h

4. Test with Traffic Simulation

Use tools like ApacheBench (ab) or Siege to test how your server handles traffic with this configuration.

5. Load Balancing

If you expect further traffic growth, consider implementing an Nginx load balancer across multiple servers to distribute traffic.

When to Adjust?

If your server still has significant free CPU and RAM during peak traffic, you can:

Increase worker\_connections for each worker.

Enable load balancing across additional servers if the single server becomes a bottleneck.

Final Suggestion

If you want an even more adaptive setup, switch to:

worker\_processes auto;

events {

use epoll;

worker\_connections 10240;

}

This configuration automatically scales worker processes with your CPU cores while keeping high connection handling capability.

Let me know how it works or if you need further tweaks!

You said:

PHP-FPM poolconfiguratie

Waar is dit?

ChatGPT said:

ChatGPT

De PHP-FPM poolconfiguratie kan afhankelijk van je systeem en PHP-installatie zich op verschillende locaties bevinden. Hieronder staan de meest voorkomende locaties voor het configuratiebestand van een specifieke pool in PHP-FPM:

Locaties van de PHP-FPM configuratiebestanden

Voor Plesk (gebruikelijk op je setup): Als je Plesk gebruikt, is de configuratie meestal te vinden in:

/opt/plesk/php/[PHP-versie]/etc/php-fpm.d/[pool-name].conf

Vervang [PHP-versie] door je PHP-versie, bijvoorbeeld 7.4. En [pool-name] is vaak de naam van de website of pool, bijvoorbeeld ride-eats.com.

Standaardlocaties op Linux: Voor een standaard PHP-FPM-installatie:

/etc/php/7.4/fpm/pool.d/[pool-name].conf

of:

/etc/php-fpm.d/[pool-name].conf

Met meerdere PHP-versies geïnstalleerd: Als je meerdere versies van PHP hebt geïnstalleerd, kijk in een specifieke map:

/etc/php/[PHP-versie]/fpm/pool.d/[pool-name].conf

Bijvoorbeeld:

/etc/php/7.4/fpm/pool.d/www.conf

Hoe de juiste configuratie te vinden

Zoek naar het bestand: Gebruik de volgende find-opdracht om de exacte locatie te achterhalen:

sudo find / -name "\*php-fpm.d\*" -type d

Controleer welke pool actief is:

Zoek in /var/log/php7.4-fpm.log of /var/log/php-fpm.log naar de naam van de pool (zoals jouw domeinnaam).

Ga dan naar de bijbehorende configuratiemap en zoek het poolbestand (bijvoorbeeld ride-eats.com.conf of www.conf).

Hoe pas je de configuratie aan?

Open het configuratiebestand met een teksteditor zoals nano:

sudo nano /etc/php/7.4/fpm/pool.d/[pool-name].conf

Breng de wijzigingen aan (bijvoorbeeld het aanpassen van pm.max\_children of pm.start\_servers).

Herstart PHP-FPM om de wijzigingen door te voeren:

sudo systemctl restart php7.4-fpm

Voorbeeld van een configuratiebestand

Een typisch poolbestand ziet er zo uit:

[ride-eats.com]

user = www-data

group = www-data

listen = /run/php/php7.4-fpm.sock

pm = dynamic

pm.max\_children = 1000

pm.start\_servers = 150

pm.min\_spare\_servers = 100

pm.max\_spare\_servers = 300

pm.process\_idle\_timeout = 5s

pm.max\_requests = 300

Lukt het niet?

Als je moeite hebt om het bestand te vinden of de configuratie niet meteen duidelijk is, laat het me weten, dan help ik je verder!

You said:

How does nginx epoll work?

ChatGPT said:

ChatGPT

How Nginx Epoll Works

Epoll is a highly efficient I/O event notification mechanism provided by the Linux kernel. It is used by Nginx to manage large numbers of concurrent connections efficiently without consuming excessive CPU or memory resources. Here's a breakdown of how it works and how it integrates with Nginx:

1. What Is Epoll?

Epoll (Event Poll) is a Linux kernel system call for scalable I/O event handling.

It replaces older mechanisms like select() and poll() by offering better performance for applications that need to handle thousands of simultaneous connections.

Key Features:

Edge-triggered or Level-triggered notifications: Allows efficient event processing without constantly checking all file descriptors.

Non-blocking operations: Processes only active connections, which reduces CPU usage.

Scalability: Handles a virtually unlimited number of file descriptors efficiently.

2. How Nginx Uses Epoll

Nginx is an event-driven web server, and Epoll is its default mechanism on Linux systems. Nginx uses Epoll to efficiently manage thousands of simultaneous connections by waiting for "events" (such as incoming data or readiness to send data) on network sockets.

Steps Nginx Takes with Epoll:

Initialize the Epoll Instance:

Nginx creates an Epoll instance using the epoll\_create() system call when it starts.

Add File Descriptors:

Each incoming connection (socket) is added to the Epoll instance using epoll\_ctl() and marked for read or write events.

Wait for Events:

Nginx calls epoll\_wait() to block until one or more events occur, such as:

A client sending a request.

A socket being ready to send data to the client.

Process Active Connections:

When events are triggered, Nginx processes only the active connections, rather than iterating through all connections.

This significantly reduces CPU overhead, especially under heavy loads.

3. Epoll Modes in Nginx

Epoll supports two modes for event notification:

Level-triggered (default):

An event is generated as long as the condition (e.g., socket ready for reading) persists.

Simpler but can lead to more system calls if not managed efficiently.

Edge-triggered (more efficient):

An event is generated only when the state changes (e.g., new data arrives).

Requires careful handling of data to avoid missing events.

Nginx primarily uses edge-triggered mode for its performance benefits.

4. Benefits of Using Epoll with Nginx

High performance with many connections:

Epoll is ideal for handling thousands (or even millions) of concurrent connections, making it perfect for high-traffic websites.

Minimal CPU overhead:

Nginx processes only active connections instead of looping through all connections, reducing unnecessary CPU cycles.

Scalability:

Adding more connections has little impact on performance compared to older mechanisms like select() or poll().

5. Configuring Epoll in Nginx

Epoll is enabled by default on Linux systems where it is supported. However, you can explicitly enable it in your Nginx configuration:

events {

use epoll; # Explicitly enable epollI use plesk. My web server is unresponsive. I am aware that I have high traffic. These are the php errors. Can you guide me? [28-Jan-2025 16:13:32] WARNING: [pool ride-eats.com] seems busy (you may need to increase pm.start\_servers, or pm.min/max\_spare\_servers), spawning 32 children, there are 0 idle, and 181 total children

###### **ChatGPT said:**

ChatGPT

The PHP error log indicates that your web server is under heavy load, and your PHP-FPM (FastCGI Process Manager) configuration isn't optimized for the level of traffic you're experiencing. Here’s how you can address this issue and make your server more responsive:

### **1. Understand the Error**

* The log shows that your server has reached the maximum number of PHP-FPM children processes (181 total), and there are no idle processes available to handle incoming requests.
* This results in requests being queued, leading to unresponsiveness.

### **2. Optimize PHP-FPM Configuration**

You need to adjust the pm settings for your PHP-FPM pool. These settings control how PHP-FPM handles incoming requests.

* Open the PHP-FPM configuration for the affected domain (e.g., ride-eats.com):
* The configuration file is typically located at /etc/php-fpm.d/[pool-name].conf or /opt/plesk/php/[version]/etc/php-fpm.d/[pool-name].conf.
* Replace [pool-name] with the name of your domain or pool.
* Look for the following parameters and adjust them:
* pm = dynamic pm.max\_children = 250 ; Increase this to allow more concurrent requests.pm.start\_servers = 50 ; Increase to handle initial burst of traffic.pm.min\_spare\_servers = 25 ; Set minimum idle processes.pm.max\_spare\_servers = 75 ; Set maximum idle processes.pm.process\_idle\_timeout = 10s ; Ensure idle processes terminate after 10 seconds.pm.max\_requests = 500 ; Restart PHP-FPM processes after handling 500 requests (prevents memory leaks).
* **Notes:**
* Increase pm.max\_children based on your server's resources (CPU and RAM).
* Make sure the new settings don't exceed your server’s capacity.

### **3. Restart PHP-FPM**

After modifying the configuration file, restart PHP-FPM to apply the changes:

sudo systemctl restart php-fpm

For specific PHP versions in Plesk:

sudo systemctl restart plesk-phpXX-fpm

Replace XX with your PHP version, e.g., 81 for PHP 8.1.

### **4. Monitor Resource Usage**

* Use commands like top, htop, or ps to monitor CPU and memory usage.
* Check the number of PHP-FPM processes:
* ps aux | grep php-fpm

### **5. Enable OPcache**

If not already enabled, OPcache can significantly improve performance by caching precompiled PHP scripts. In Plesk:

* Go to **Tools & Settings** > **PHP Settings** > Select your PHP version.
* Enable OPcache and configure it.

Recommended OPcache settings:

opcache.enable=1opcache.memory\_consumption=256opcache.max\_accelerated\_files=10000opcache.validate\_timestamps=0

### **6. Optimize the Web Server**

You can also optimize the web server (Apache/Nginx) configuration:

* If you’re using Nginx as a reverse proxy in front of Apache, increase the worker\_processes and worker\_connections in the Nginx configuration file (e.g., /etc/nginx/nginx.conf):
* worker\_processes auto;
* events {
* worker\_connections 1024;
* }
* Restart Nginx:
* sudo systemctl restart nginx

### **7. Implement Caching**

To further reduce PHP-FPM load:

* Use caching mechanisms like Redis or Memcached.
* Enable full-page caching for static content (if supported by your application).

### **8. Upgrade Server Resources**

If your current server cannot handle the load:

* Increase your server’s CPU and RAM.
* Consider load balancing if traffic continues to grow.

### **9. Test and Monitor**

* Test your site under simulated load to ensure the changes have improved performance.
* Continuously monitor logs and traffic patterns using tools like Plesk's monitoring panel or external services (e.g., New Relic, Datadog).

Let me know if you encounter any issues while applying these changes!

##### **You said:**

Are you not interested what my current config is instead of just giving me random configuration on assumptions?

###### **ChatGPT said:**

ChatGPT

Je hebt helemaal gelijk. Het is inderdaad verstandiger om eerst jouw huidige configuratie te begrijpen voordat er zomaar aanpassingen worden voorgesteld. Dit zorgt ervoor dat de adviezen nauwkeurig zijn en afgestemd op jouw specifieke situatie. Kun je jouw huidige PHP-FPM configuratie delen, zodat we samen kunnen bekijken wat er aangepast moet worden?

Hier zijn de specifieke instellingen die nuttig kunnen zijn om te delen:

1. **De huidige configuratie van PHP-FPM voor de specifieke pool**: Je kunt dit bestand meestal vinden in een van de volgende paden:

* /etc/php-fpm.d/[pool-name].conf
* /opt/plesk/php/[versie]/etc/php-fpm.d/[pool-name].conf

1. Deel de waarden van de volgende parameters:

* pm
* pm.max\_children
* pm.start\_servers
* pm.min\_spare\_servers
* pm.max\_spare\_servers
* pm.max\_requests

1. **De serverinformatie**:

* Hoeveel RAM en CPU's heeft de server?
* Welke versie van PHP gebruik je?

1. **De belasting van de server**:

* Hoeveel bezoekers heb je gemiddeld op het moment van piekverkeer?
* Wat is de huidige CPU- en RAM-belasting tijdens hoge verkeerspieken? (Met tools zoals top of htop kun je dit controleren.)

1. **Andere relevante logs**:

* Andere foutmeldingen in de PHP of webserver logs (bijvoorbeeld Nginx of Apache logs).

Met deze informatie kan ik een veel nauwkeuriger advies geven. Laat maar weten wat je kunt delen!

##### **You said:**

Parameters: pm = dynamic pm.max\_children = 777 pm.max\_requests = 500 pm.max\_spare\_servers = 150 pm.min\_spare\_servers = 50 pm.process\_idle\_timeout = 10s pm.start\_servers = 100 PHP versie: 7.4 Server is op lightsail: 32gb 640 gb ssd 8vcpu Ik weet niet hoeveel bezoekers ik heb maar ik weet dat er ongeveer 1000 php processes open zijn in htop CPU is niet echt hoog. Misschien 30% op alle cpu's. Ram is ongeveer 4-5 gb /etc/apache2/mods-enabled/mpm\_event.conf: <IfModule mpm\_event\_module> StartServers 2 MinSpareThreads 100 MaxSpareThreads 400 ThreadLimit 64 ThreadsPerChild 50 MaxRequestWorkers 1400 ServerLimit 400 MaxConnectionsPerChild 1600 </IfModule>

###### **ChatGPT said:**

ChatGPT

Bedankt dat je de huidige configuratie deelt. Met deze informatie kunnen we een veel beter beeld krijgen van de situatie en specifieke aanpassingen voorstellen. Je configuratie laat zien dat je server redelijk krachtig is (32GB RAM, 8 vCPU’s), maar het aantal actieve PHP-processen (1000) is uitzonderlijk hoog, terwijl je CPU- en RAM-gebruik relatief laag blijven. Dit suggereert dat er mogelijk een inefficiëntie of een bottleneck in de configuratie of applicatie is.

Hier is wat ik aanraad op basis van jouw situatie:

### **1. PHP-FPM Optimalisatie**

Hoewel je een hoog aantal pm.max\_children hebt (777), lijkt het erop dat het nog steeds niet voldoende is voor de werkdruk. Echter, omdat je CPU en RAM niet volledig benut worden, kun je het aantal processen waarschijnlijk verhogen of efficiënter inzetten.

* **Aanpassing PHP-FPM configuratie:** Overweeg de volgende wijzigingen:
* pm.max\_children = 1000 ; Verhoog het aantal processen om beter om te gaan met het huidige verkeer.pm.start\_servers = 150 ; Start meer processen bij het opstarten.pm.min\_spare\_servers = 100 ; Zorg dat er altijd voldoende idle processen zijn.pm.max\_spare\_servers = 300 ; Bied meer ruimte voor fluctuaties in verkeer.pm.process\_idle\_timeout = 5s ; Verkort de idle timeout, zodat processen sneller worden beëindigd.pm.max\_requests = 300 ; Verlaag om geheugenlekken te voorkomen bij langdurige processen.
* **Waarom deze aanpassingen?** Je hebt veel RAM beschikbaar, dus je kunt veilig meer processen draaien. Door de pm.process\_idle\_timeout te verkorten en pm.max\_requests te verlagen, worden inefficiënte of vastgelopen processen sneller beëindigd.

**Belangrijk:** Na deze aanpassingen moet je PHP-FPM herstarten:

sudo systemctl restart php7.4-fpm

### **2. Apache MPM Configuratie**

Je Apache MPM Event configuratie lijkt al redelijk efficiënt, maar het kan worden afgestemd op de PHP-FPM pool:

* **Aanpassing Apache configuratie:** Pas de onderstaande parameters aan in /etc/apache2/mods-enabled/mpm\_event.conf:
* StartServers 4
* MinSpareThreads 200
* MaxSpareThreads 600
* ThreadsPerChild 75
* MaxRequestWorkers 1800
* MaxConnectionsPerChild 2000
* **Waarom deze wijzigingen?**
* Verhoog de ThreadsPerChild en MaxRequestWorkers om meer gelijktijdige verzoeken aan te kunnen.
* Verhoog MaxConnectionsPerChild om processen minder vaak te herstarten.

**Herstart Apache na de wijzigingen:**

sudo systemctl restart apache2

### **3. Analyse van de PHP-processen**

Het feit dat je 1000 PHP-processen ziet in htop kan wijzen op inefficiënte code of te lange verwerkingstijden. Controleer het volgende:

* **Langlopende scripts:**
* Gebruik slowlog in PHP-FPM om langzame scripts te identificeren: Voeg dit toe in de PHP-FPM poolconfiguratie:

request\_slowlog\_timeout = 5s slowlog = /var/log/php7.4-fpm-slow.log

Herstart PHP-FPM en controleer vervolgens het logbestand:

* sudo systemctl restart php7.4-fpm tail -f /var/log/php7.4-fpm-slow.log
* **Inefficiënte queries:** Als jouw applicatie een database gebruikt, controleer dan of er query’s zijn die lang duren of veel bronnen verbruiken.

### **4. Caching Implementeren**

Als je geen caching gebruikt, kan dit een aanzienlijke belasting op PHP verminderen. Overweeg:

* **OPcache**: Zorg dat OPcache is ingeschakeld met voldoende geheugen (controleer via Plesk of voeg het handmatig toe):
* opcache.enable=1opcache.memory\_consumption=512opcache.max\_accelerated\_files=20000opcache.validate\_timestamps=0
* **Full-page caching**: Gebruik caching op applicatieniveau (bijv. Redis of Memcached).

### **5. Load Balancing Overwegen**

Als je verkeer blijft toenemen, kan een enkele server een bottleneck worden. Overweeg het volgende:

* Zet een extra Lightsail-instantie in als webserver en gebruik een load balancer om het verkeer te verdelen.
* Amazon Lightsail heeft een ingebouwde load balancing-optie.

### **6. Monitoring**

Houd na de wijzigingen je serverprestaties in de gaten:

* Gebruik tools zoals htop of atop om de belasting van CPU, RAM en processen te controleren.
* Controleer de PHP-FPM logs en Apache logs regelmatig op foutmeldingen of problemen.

### **7. Eventuele Toepassingsproblemen**

Als je serverconfiguratie goed lijkt, maar je toch zoveel processen hebt, kan er een probleem zijn met de applicatie zelf, zoals:

* Oneindige loops of inefficiënte algoritmes.
* Overbodige API-aanroepen of externe verzoeken.

Zorg ervoor dat je applicatie code wordt geoptimaliseerd als bovenstaande wijzigingen geen significant verschil maken.

Laat me weten hoe de voorgestelde aanpassingen werken en of je nog meer hulp nodig hebt!

##### **You said:**

2\. \*\*Apache MPM Configuratie\*\*

Is deze configuratie gelijk met de PHP-FPM configuratie?

###### **ChatGPT said:**

ChatGPT

Nee, de configuratie van Apache MPM en PHP-FPM zijn verschillend en hebben elk een aparte rol in het verwerken van verzoeken:

### **Apache MPM Configuratie**

* Apache MPM (Multi-Processing Module) beheert hoe Apache verzoeken afhandelt, zoals verbindingen met clients (bijvoorbeeld browsers).
* Het regelt threading, processen, en hoeveel gelijktijdige verbindingen de webserver aankan.
* Dit bepaalt hoe Apache communiceert met PHP-FPM via FastCGI.

**Belangrijke parameters in Apache MPM:**

* StartServers, MinSpareThreads, MaxSpareThreads: Regelen hoeveel threads of processen beschikbaar zijn om verbindingen te accepteren.
* MaxRequestWorkers: Het maximum aantal gelijktijdige verbindingen dat Apache aankan.
* MaxConnectionsPerChild: Het maximum aantal verzoeken dat een Apache-proces verwerkt voordat het wordt herstart.

**Doel:** Apache verwerkt HTTP(S)-verzoeken en stuurt PHP-verzoeken door naar PHP-FPM via FastCGI.

### **PHP-FPM Configuratie**

* PHP-FPM (FastCGI Process Manager) beheert hoe PHP-scripts worden uitgevoerd.
* Het draait processen die PHP-code uitvoeren en communiceert met Apache via een FastCGI-module.

**Belangrijke parameters in PHP-FPM:**

* pm (process manager): Bepaalt hoe PHP-FPM processen beheert (static, dynamic, of ondemand).
* pm.max\_children: Het maximum aantal PHP-processen dat tegelijkertijd kan draaien.
* pm.start\_servers, pm.min\_spare\_servers, pm.max\_spare\_servers: Bepalen hoe processen worden opgestart en beheerd.
* pm.max\_requests: Bepaalt hoeveel verzoeken een proces verwerkt voordat het opnieuw wordt gestart.

**Doel:** PHP-FPM voert de PHP-code uit die door Apache wordt doorgestuurd.

### **Relatie tussen Apache MPM en PHP-FPM**

1. Apache MPM bepaalt hoeveel verbindingen met clients (zoals browsers) het aankan.
2. Voor PHP-verzoeken stuurt Apache deze door naar PHP-FPM.
3. PHP-FPM verwerkt de PHP-code en stuurt het resultaat terug naar Apache.
4. Apache geeft het resultaat door aan de client.

### **Waarom zijn beide configuraties belangrijk?**

* Als Apache meer verzoeken toestaat (MaxRequestWorkers) dan PHP-FPM aankan (pm.max\_children), zullen verzoeken in de wachtrij blijven en kan de server traag reageren of time-outs geven.
* Beide configuraties moeten goed op elkaar worden afgestemd, afhankelijk van je serverbronnen (CPU en RAM) en verkeersniveau.

### **Aanbeveling**

* Zorg ervoor dat de waarde van MaxRequestWorkers in Apache MPM niet hoger is dan het aantal gelijktijdige processen dat PHP-FPM aankan (pm.max\_children).
* In jouw geval stel ik voor dat je MaxRequestWorkers (Apache) aanpast aan de hogere waarde van pm.max\_children (PHP-FPM) of iets lager houdt, bijvoorbeeld:
* MaxRequestWorkers = 1000
* Hiermee zorg je ervoor dat er geen onevenwicht ontstaat tussen Apache en PHP-FPM.

Laat me weten of je meer hulp nodig hebt bij het verder afstemmen van deze configuraties!

##### **You said:**

Dus de MPM configuratie wat je aangaf is dus correct?

###### **ChatGPT said:**

ChatGPT

De **MPM-configuratie** die ik eerder voorstelde is in grote lijnen correct, maar omdat jouw PHP-FPM configuratie al goed is ingesteld (met een hoge waarde voor pm.max\_children van 777) en jouw server voldoende krachtig is, wil ik de configuratie van Apache MPM nog iets nauwkeuriger afstemmen op jouw specifieke situatie.

### **Afgestemde Apache MPM-configuratie**

Hier is een aangepaste versie van de MPM-configuratie, specifiek afgestemd op jouw huidige PHP-FPM instellingen en serverbronnen:

<IfModule mpm\_event\_module>

StartServers 4 # Start iets meer servers om pieken op te vangen.

MinSpareThreads 200 # Zorg voor voldoende threads om idle te blijven.

MaxSpareThreads 600 # Voldoende threads om verkeer op te vangen.

ThreadsPerChild 75 # Meer threads per child voor betere schaalbaarheid.

MaxRequestWorkers 1000 # Houd dit op max. 1000, afgestemd op PHP-FPM.

ServerLimit 15 # Bepaalt het maximum aantal child processen.

MaxConnectionsPerChild 2000 # Beperk het herstarten van processen om overhead te verminderen.

</IfModule>

### **Waarom deze wijzigingen?**

1. **ThreadsPerChild:** Het verhogen van ThreadsPerChild naar 75 zorgt ervoor dat één Apache-proces meer verzoeken aankan, wat efficiënter is.
2. **MaxRequestWorkers:** Deze waarde is nu beperkt tot 1000, wat past bij de limiet van jouw PHP-FPM (pm.max\_children = 777) en geeft nog wat ruimte voor statische verzoeken (zoals CSS en JS).
3. **MaxConnectionsPerChild:** Door deze hoger te zetten (2000), worden Apache-processen minder vaak herstart, wat overhead vermindert.
4. **Evenwicht tussen Apache en PHP-FPM:** Deze configuratie zorgt ervoor dat beide goed samenwerken zonder dat een van de twee een bottleneck wordt.

### **Houd dit in gedachten**

* Als je later merkt dat de server nog steeds ongebruikte bronnen heeft (bijvoorbeeld veel vrije CPU of RAM), kun je de waarden voor MaxRequestWorkers en pm.max\_children verder verhogen.
* **Herstart Apache na de wijzigingen:**
* sudo systemctl restart apache2

### **Volgende stap: Monitoring**

Na het aanbrengen van deze wijzigingen:

1. Controleer regelmatig de belasting met htop of atop.
2. Bekijk of de wachtrijen voor PHP-processen (in htop) verminderen.
3. Controleer of er geen Apache time-outs optreden in de error logs:
4. sudo tail -f /var/log/apache2/error.log

Laat weten of je nog verdere optimalisaties nodig hebt!

##### **You said:**

\*\*Afgestemde Apache MPM-configuratie\*\*

Ik heb de PHP-FPM configuratie gebruikt die je me gaf. De server moet nu beter functioneren. Dit is wat je daarnet gaf: pm.max\_children = 1000 ; Verhoog het aantal processen om beter om te gaan met het huidige verkeer. pm.start\_servers = 150 ; Start meer processen bij het opstarten. pm.min\_spare\_servers = 100 ; Zorg dat er altijd voldoende idle processen zijn. pm.max\_spare\_servers = 300 ; Bied meer ruimte voor fluctuaties in verkeer. pm.process\_idle\_timeout = 5s ; Verkort de idle timeout, zodat processen sneller worden beëindigd. pm.max\_requests = 300 ; Verlaag om geheugenlekken te voorkomen bij langdurige processen.

###### **ChatGPT said:**

ChatGPT

Goed dat je de wijzigingen hebt toegepast! De PHP-FPM configuratie die je hebt gebruikt lijkt een goede stap in de richting van een beter functionerende server, zeker gezien de verkeersniveaus en serverbronnen die je hebt. Om de configuratie nu volledig op elkaar af te stemmen, stel ik voor om de Apache MPM-configuratie die ik eerder heb gedeeld als volgt te gebruiken:

### **Afgestemde Apache MPM-configuratie (opnieuw)**

Hier nogmaals de configuratie met een toelichting waarom deze waarden goed werken samen met jouw PHP-FPM configuratie:

<IfModule mpm\_event\_module>

StartServers 4 # Begin met meer child-processen voor sneller opstarten.

MinSpareThreads 200 # Houd genoeg idle threads voor een vlotte respons.

MaxSpareThreads 600 # Laat voldoende threads beschikbaar voor pieken.

ThreadsPerChild 75 # Elk proces kan meer threads (verzoeken) afhandelen.

MaxRequestWorkers 1000 # Past goed bij jouw `pm.max\_children = 1000`.

ServerLimit 15 # Zorgt dat Apache niet meer processen start dan nodig.

MaxConnectionsPerChild 2000 # Herstart processen minder vaak om overhead te verlagen.

</IfModule>

### **Waarom werkt dit samen met de PHP-FPM instellingen?**

1. **MaxRequestWorkers = 1000:**

* Deze waarde sluit aan bij de **maximale capaciteit** van PHP-FPM (pm.max\_children = 1000). Hierdoor kan Apache gelijktijdig maximaal 1000 PHP-verzoeken naar PHP-FPM sturen zonder dat een wachtrij ontstaat.

1. **ThreadsPerChild = 75:**

* Elk Apache child-proces kan meer verbindingen verwerken, wat handig is bij piekverkeer.

1. **MaxConnectionsPerChild = 2000:**

* Door processen minder vaak te herstarten, voorkom je onnodige overhead en verlies je geen tijd met het opnieuw opbouwen van verbindingen.

1. **Balans tussen Apache en PHP-FPM:**

* Beide systemen zijn nu beter in balans, waardoor wachtrijen en time-outs minimaal zouden moeten zijn.

### **Wat nu?**

1. **Herstart Apache en PHP-FPM om de wijzigingen door te voeren:**

* sudo systemctl restart apache2
* sudo systemctl restart php7.4-fpm
* **Monitor je server:**
* Gebruik htop of atop om de belasting op CPU, RAM en het aantal processen in de gaten te houden.
* Controleer de PHP-FPM logs:
* sudo tail -f /var/log/php7.4-fpm.log
* Controleer de Apache logs:
* sudo tail -f /var/log/apache2/error.log

1. **Langetermijnoptimalisatie:**

* Blijf kijken of de server alle bronnen optimaal gebruikt. Als je nog veel vrije CPU en RAM hebt, kun je eventueel MaxRequestWorkers of pm.max\_children verder verhogen.
* Overweeg cachingmechanismen zoals Redis of OPcache als je die nog niet hebt ingeschakeld.

Laat weten hoe je server zich gedraagt met deze instellingen of als je verdere hulp nodig hebt!

##### **You said:**

<IfModule mpm\\_event\\_module> StartServers 4 # Begin met meer child-processen voor sneller opstarten. MinSpareThreads 200 # Houd genoeg idle threads voor een vlotte respons. MaxSpareThreads 600 # Laat voldoende threads beschikbaar voor pieken. ThreadsPerChild 75 # Elk proces kan meer threads (verzoeken) afhandelen. MaxRequestWorkers 1000 # Past goed bij jouw \`pm.max\\_children = 1000\`. ServerLimit 15 # Zorgt dat Apache niet meer processen start dan nodig. MaxConnectionsPerChild 2000 # Herstart processen minder vaak om overhead te verlagen. </IfModule>

Wat about ThreadLimit?

###### **ChatGPT said:**

ChatGPT

De **ThreadLimit** is een belangrijke parameter in de Apache MPM Event-module omdat deze het maximale aantal threads per child-proces instelt. Het fungeert als een harde limiet voor hoeveel threads per child-proces kunnen worden toegewezen.

### **Wat is het doel van ThreadLimit?**

* **Harde limiet:** ThreadLimit bepaalt de maximale waarde van ThreadsPerChild. Als je ThreadsPerChild hoger instelt dan de waarde van ThreadLimit, zal Apache dit negeren en het terugzetten naar de waarde van ThreadLimit.
* Het wordt meestal gecombineerd met andere parameters zoals ThreadsPerChild en ServerLimit.

### **Wat stel je in voor ThreadLimit?**

In jouw configuratie gebruik je ThreadsPerChild = 75, dus je moet ervoor zorgen dat ThreadLimit minstens **75** of hoger is. Als je ThreadsPerChild later wilt verhogen, is het verstandig om een iets hogere waarde in te stellen voor flexibiliteit.

Voor jouw server stel ik het volgende voor:

<IfModule mpm\_event\_module>

StartServers 4

MinSpareThreads 200

MaxSpareThreads 600

ThreadsPerChild 75

MaxRequestWorkers 1000

ServerLimit 15

MaxConnectionsPerChild 2000

ThreadLimit 100 # Stel dit in iets boven ThreadsPerChild.

</IfModule>

### **Waarom ThreadLimit = 100?**

1. Het biedt voldoende ruimte voor toekomstige schaalbaarheid.
2. Je voorkomt dat je beperkt wordt door een te lage waarde van ThreadLimit als je ThreadsPerChild wilt verhogen.
3. Het blijft binnen de mogelijkheden van jouw krachtige server.

### **Hoe werkt het samen met MaxRequestWorkers?**

De MaxRequestWorkers-waarde bepaalt het totale aantal gelijktijdige verzoeken dat Apache aankan. Dit wordt berekend als volgt:

MaxRequestWorkers=ThreadsPerChild×ServerLimitMaxRequestWorkers=ThreadsPerChild×ServerLimit

Met de bovenstaande configuratie:

* ThreadsPerChild = 75
* ServerLimit = 15

MaxRequestWorkers=75×15=1125MaxRequestWorkers=75×15=1125

Omdat je MaxRequestWorkers = 1000 hebt ingesteld, gebruik je iets minder dan de maximale capaciteit, wat zorgt voor een goede balans en voorkomt overbelasting.

### **Wat nu?**

1. Voeg ThreadLimit toe aan je configuratie.
2. Herstart Apache:
3. sudo systemctl restart apache2

Controleer na deze aanpassingen of alles soepel werkt, en laat me weten of je meer vragen hebt!

##### **You said:**

worker\\_processes auto; events { worker\\_connections 1024; }

worker\_processes 8; events { worker\_connections 10240; } Is this enough?

###### **ChatGPT said:**

ChatGPT

Your current configuration of:

worker\_processes 8;

events {

worker\_connections 10240;

}

is a strong setup, but let's analyze it based on your server's specifications and traffic load.

### **Breakdown of the Configuration**

1. **worker\_processes 8:**

* This defines the number of worker processes Nginx spawns. Setting it to 8 matches your **8 vCPUs**, which is optimal for CPU utilization.
* However, you could also set it to auto, which automatically uses the number of available CPU cores:
* worker\_processes auto;
* This ensures you always use the maximum number of CPU cores available, even if you later upgrade your server.

1. **worker\_connections 10240:**

* Each worker process can handle up to 10,240 simultaneous connections.
* The total number of connections your server can handle is calculated as: \text{Total connections} = \text{worker\_processes} \times \text{worker\_connections} With your current configuration: 8×10240=81,920 simultaneous connections8×10240=81,920 simultaneous connections This is **more than enough** for most high-traffic websites. However, keep in mind that each connection includes HTTP keep-alives, static files, and back-end proxy connections, so the real number of clients your server can handle may be lower.

### **Recommendations**

Your configuration is already strong, but here are some additional points to ensure it's optimized:

#### **1. Enable Epoll (Linux-specific)**

Epoll is a more efficient event mechanism on Linux and is often enabled by default. To explicitly enable it:

events {

use epoll;

worker\_connections 10240;

}

#### **2. Optimize Keep-Alive Settings**

Reduce unnecessary connections by optimizing keep-alive:

keepalive\_timeout 65;

keepalive\_requests 100;

* keepalive\_timeout: Reduces idle time for persistent connections.
* keepalive\_requests: Limits the number of requests per connection to prevent long-lived connections.

#### **3. Monitor Resource Usage**

Ensure your configuration isn’t overloading the server:

* **CPU Usage:** Since worker\_processes is set to 8, you should monitor if all CPU cores are utilized efficiently.
* **RAM Usage:** The number of simultaneous connections affects RAM usage. Monitor with:
* free -h

#### **4. Test with Traffic Simulation**

Use tools like ApacheBench (ab) or Siege to test how your server handles traffic with this configuration.

#### **5. Load Balancing**

If you expect further traffic growth, consider implementing an Nginx load balancer across multiple servers to distribute traffic.

### **When to Adjust?**

* If your server still has significant free CPU and RAM during peak traffic, you can:
* Increase worker\_connections for each worker.
* Enable load balancing across additional servers if the single server becomes a bottleneck.

### **Final Suggestion**

If you want an even more adaptive setup, switch to:

worker\_processes auto;

events {

use epoll;

worker\_connections 10240;

}

This configuration automatically scales worker processes with your CPU cores while keeping high connection handling capability.

Let me know how it works or if you need further tweaks!

##### **You said:**

PHP-FPM poolconfiguratie

Waar is dit?

###### **ChatGPT said:**

ChatGPT

De **PHP-FPM poolconfiguratie** kan afhankelijk van je systeem en PHP-installatie zich op verschillende locaties bevinden. Hieronder staan de meest voorkomende locaties voor het configuratiebestand van een specifieke pool in PHP-FPM:

### **Locaties van de PHP-FPM configuratiebestanden**

1. **Voor Plesk (gebruikelijk op je setup):** Als je Plesk gebruikt, is de configuratie meestal te vinden in:

* /opt/plesk/php/[PHP-versie]/etc/php-fpm.d/[pool-name].conf
* Vervang [PHP-versie] door je PHP-versie, bijvoorbeeld 7.4. En [pool-name] is vaak de naam van de website of pool, bijvoorbeeld ride-eats.com.
* **Standaardlocaties op Linux:** Voor een standaard PHP-FPM-installatie:

/etc/php/7.4/fpm/pool.d/[pool-name].conf

of:

* /etc/php-fpm.d/[pool-name].conf
* **Met meerdere PHP-versies geïnstalleerd:** Als je meerdere versies van PHP hebt geïnstalleerd, kijk in een specifieke map:

/etc/php/[PHP-versie]/fpm/pool.d/[pool-name].conf

Bijvoorbeeld:

1. /etc/php/7.4/fpm/pool.d/www.conf

### **Hoe de juiste configuratie te vinden**

1. **Zoek naar het bestand:** Gebruik de volgende find-opdracht om de exacte locatie te achterhalen:
2. sudo find / -name "\*php-fpm.d\*" -type d
3. **Controleer welke pool actief is:**

* Zoek in /var/log/php7.4-fpm.log of /var/log/php-fpm.log naar de naam van de pool (zoals jouw domeinnaam).
* Ga dan naar de bijbehorende configuratiemap en zoek het poolbestand (bijvoorbeeld ride-eats.com.conf of www.conf).

### **Hoe pas je de configuratie aan?**

1. Open het configuratiebestand met een teksteditor zoals nano:

* sudo nano /etc/php/7.4/fpm/pool.d/[pool-name].conf
* Breng de wijzigingen aan (bijvoorbeeld het aanpassen van pm.max\_children of pm.start\_servers).
* **Herstart PHP-FPM om de wijzigingen door te voeren:**

1. sudo systemctl restart php7.4-fpm

### **Voorbeeld van een configuratiebestand**

Een typisch poolbestand ziet er zo uit:

[ride-eats.com]user = www-data group = www-data listen = /run/php/php7.4-fpm.sock pm = dynamic pm.max\_children = 1000pm.start\_servers = 150pm.min\_spare\_servers = 100pm.max\_spare\_servers = 300pm.process\_idle\_timeout = 5s pm.max\_requests = 300

### **Lukt het niet?**

Als je moeite hebt om het bestand te vinden of de configuratie niet meteen duidelijk is, laat het me weten, dan help ik je verder!

##### **You said:**

How does nginx epoll work?

###### **ChatGPT said:**

ChatGPT

### **How Nginx Epoll Works**

Epoll is a highly efficient I/O event notification mechanism provided by the Linux kernel. It is used by Nginx to manage large numbers of concurrent connections efficiently without consuming excessive CPU or memory resources. Here's a breakdown of how it works and how it integrates with Nginx:

### **1. What Is Epoll?**

* **Epoll** (Event Poll) is a Linux kernel system call for scalable I/O event handling.
* It replaces older mechanisms like select() and poll() by offering better performance for applications that need to handle thousands of simultaneous connections.

#### **Key Features:**

* **Edge-triggered or Level-triggered notifications:** Allows efficient event processing without constantly checking all file descriptors.
* **Non-blocking operations:** Processes only active connections, which reduces CPU usage.
* **Scalability:** Handles a virtually unlimited number of file descriptors efficiently.

### **2. How Nginx Uses Epoll**

Nginx is an event-driven web server, and Epoll is its default mechanism on Linux systems. Nginx uses Epoll to efficiently manage thousands of simultaneous connections by waiting for "events" (such as incoming data or readiness to send data) on network sockets.

#### **Steps Nginx Takes with Epoll:**

1. **Initialize the Epoll Instance:**

* Nginx creates an Epoll instance using the epoll\_create() system call when it starts.

1. **Add File Descriptors:**

* Each incoming connection (socket) is added to the Epoll instance using epoll\_ctl() and marked for read or write events.

1. **Wait for Events:**

* Nginx calls epoll\_wait() to block until one or more events occur, such as:
* A client sending a request.
* A socket being ready to send data to the client.

1. **Process Active Connections:**

* When events are triggered, Nginx processes only the active connections, rather than iterating through all connections.
* This significantly reduces CPU overhead, especially under heavy loads.

### **3. Epoll Modes in Nginx**

Epoll supports two modes for event notification:

1. **Level-triggered (default):**

* An event is generated as long as the condition (e.g., socket ready for reading) persists.
* Simpler but can lead to more system calls if not managed efficiently.

1. **Edge-triggered (more efficient):**

* An event is generated only when the state changes (e.g., new data arrives).
* Requires careful handling of data to avoid missing events.

Nginx primarily uses **edge-triggered mode** for its performance benefits.

### **4. Benefits of Using Epoll with Nginx**

* **High performance with many connections:**
* Epoll is ideal for handling thousands (or even millions) of concurrent connections, making it perfect for high-traffic websites.
* **Minimal CPU overhead:**
* Nginx processes only active connections instead of looping through all connections, reducing unnecessary CPU cycles.
* **Scalability:**
* Adding more connections has little impact on performance compared to older mechanisms like select() or poll().

### **5. Configuring Epoll in Nginx**

Epoll is enabled by default on Linux systems where it is supported. However, you can explicitly enable it in your Nginx configuration:

events {

use epoll; # Explicitly enable epoll

worker\_connections 10240; # Max connections per worker process

}

#### **Key Parameters:**

* **worker\_processes:**
* Set to auto or the number of CPU cores to maximize concurrency.
* **worker\_connections:**
* The maximum number of connections each worker process can handle.
* Total connections = worker\_processes × worker\_connections.

### **6. Limitations of Epoll**

While Epoll is highly efficient, it has some caveats:

* **Linux-only:** Epoll is not available on other operating systems (e.g., BSD or macOS).
* **Complex edge-triggered mode:** Applications must carefully handle edge-triggered events to avoid race conditions or missed notifications.

### **7. Example Workflow in Nginx with Epoll**

1. **Incoming Request:**

* A client sends a request to Nginx.
* The socket is added to the Epoll instance.

1. **Epoll Wait:**

* Nginx blocks on epoll\_wait() until data arrives or the socket becomes ready for writing.

1. **Event Processing:**

* When the socket is ready, Nginx processes the request (e.g., reads data, sends responses).

1. **Connection Close:**

* When the request is complete, Nginx removes the socket from the Epoll instance.

### **8. Monitoring Epoll Performance**

You can monitor how well Epoll is performing on your Nginx server:

* Use tools like netstat, ss, or nginx stub\_status to view connection states.
* Analyze CPU usage and system calls with tools like strace or perf.

### **Conclusion**

Epoll is a cornerstone of Nginx's high-performance architecture on Linux. By processing only active connections and leveraging efficient system calls, it allows Nginx to handle thousands of concurrent users with minimal resource usage. Proper configuration and monitoring ensure you get the most out of this powerful mechanism.

worker\_connections 10240; # Max connections per worker process

}

Key Parameters:

worker\_processes:

Set to auto or the number of CPU cores to maximize concurrency.

worker\_connections:

The maximum number of connections each worker process can handle.

Total connections = worker\_processes × worker\_connections.

6. Limitations of Epoll

While Epoll is highly efficient, it has some caveats:

Linux-only: Epoll is not available on other operating systems (e.g., BSD or macOS).

Complex edge-triggered mode: Applications must carefully handle edge-triggered events to avoid race conditions or missed notifications.

7. Example Workflow in Nginx with Epoll

Incoming Request:

A client sends a request to Nginx.

The socket is added to the Epoll instance.

Epoll Wait:

Nginx blocks on epoll\_wait() until data arrives or the socket becomes ready for writing.

Event Processing:

When the socket is ready, Nginx processes the request (e.g., reads data, sends responses).

Connection Close:

When the request is complete, Nginx removes the socket from the Epoll instance.

8. Monitoring Epoll Performance

You can monitor how well Epoll is performing on your Nginx server:

Use tools like netstat, ss, or nginx stub\_status to view connection states.

Analyze CPU usage and system calls with tools like strace or perf.

Conclusion

Epoll is a cornerstone of Nginx's high-performance architecture on Linux. By processing only active connections and leveraging efficient system calls, it allows Nginx to handle thousands of concurrent users with minimal resource usage. Proper configuration and monitoring ensure you get the most out of this powerful mechanism.