



The PHP Company

Varnish & Zend Server

By Gaylord Aulke

<http://100days.de>

Agenda

1. Web Application Caching Grundsätze
2. Zend Server Caching
3. Varnish Cache
4. Varnish deployen, konfigurieren
5. Beispiel - Szenario

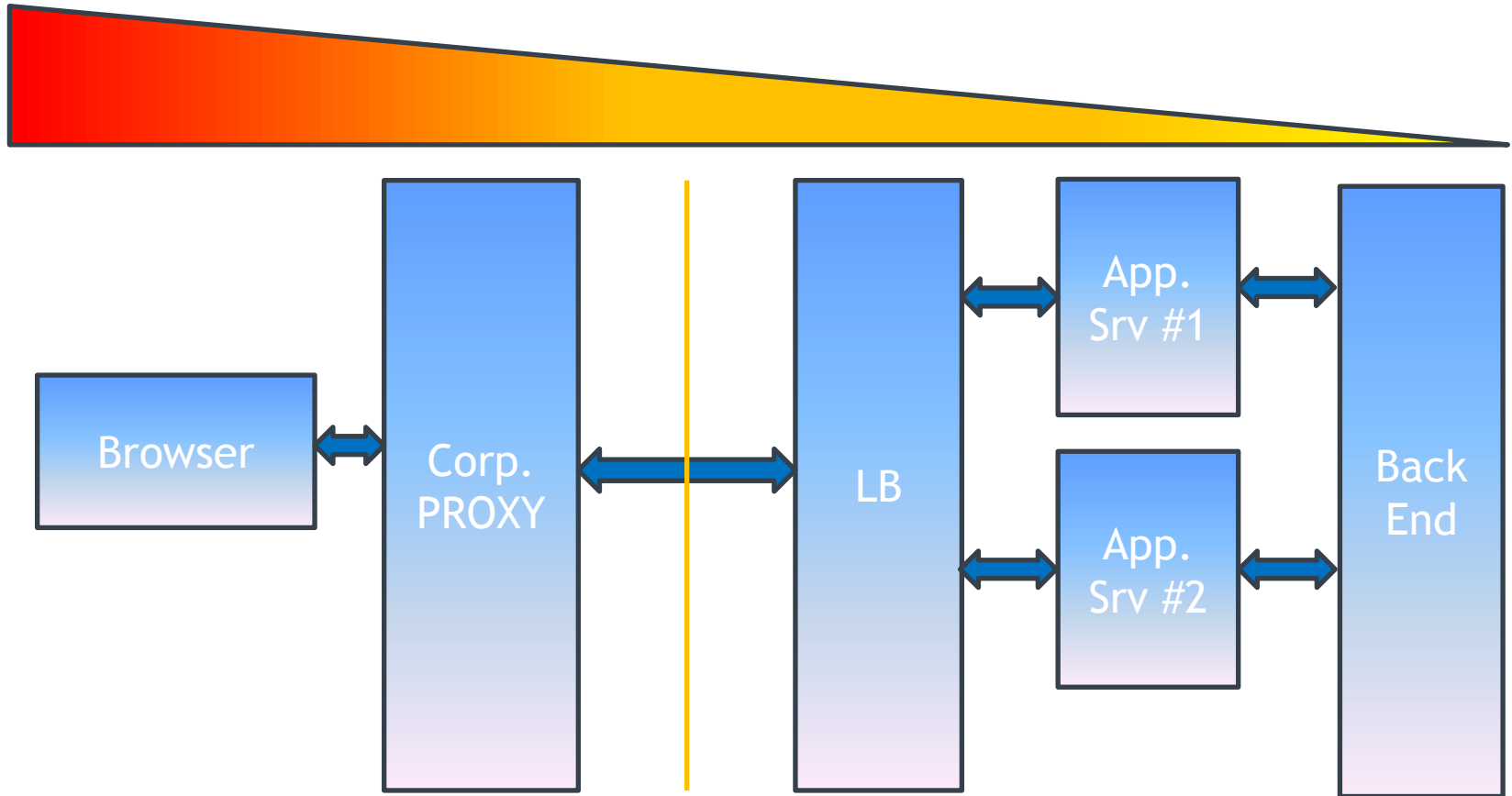
Web Application Caching Grundsätze

- So viel wie möglich cachen
- Anfragen so früh wie möglich cachen
- Cache-Hits sollen mit möglichst wenig Aufwand bearbeitet werden, Cache-Misses sollten den Server nicht umbringen
- Aktive Cache-Invalidierung möglichst vermeiden
- Es darf niemals zu viel gecached werden

Ziel:

So viel Serverkapazität wie möglich für die wirklich aufwändigen Dinge aufbewahren

Cache-Effizienz



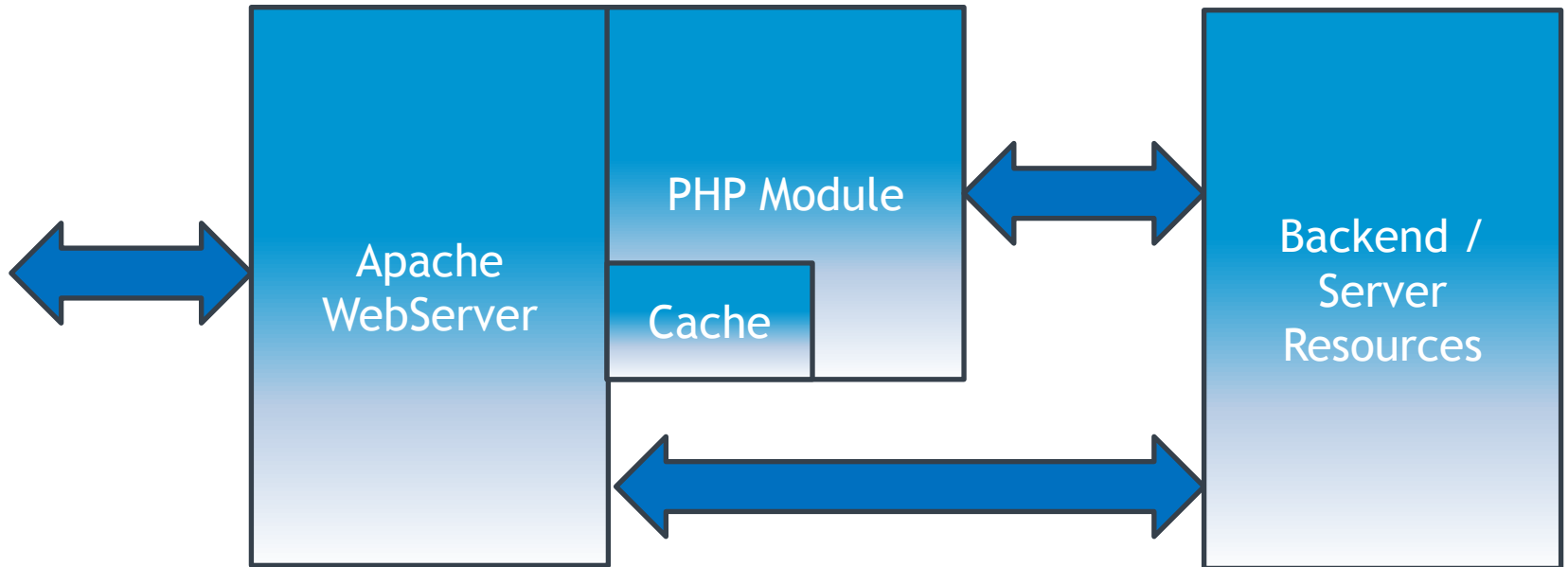
Web Application Caching Grundsätze

- **Caching auf allen Ebenen:**
 - ▶ Page-Caching: Antwort auf komplette Anfragen cachen
 - ▶ Partial Page-Caching: Teile von Antworten cachen
 - ▶ Data Caching: Daten aus Backend Systemen cachen

- **Cache Expiration**
 - ▶ Über Timeout
 - ▶ Über explizites Löschen von Cache-Elementen
 - ▶ Über modifizierte URL / Parameter

Zend Server Cache

Zend Server Cache



Zend Server Cache

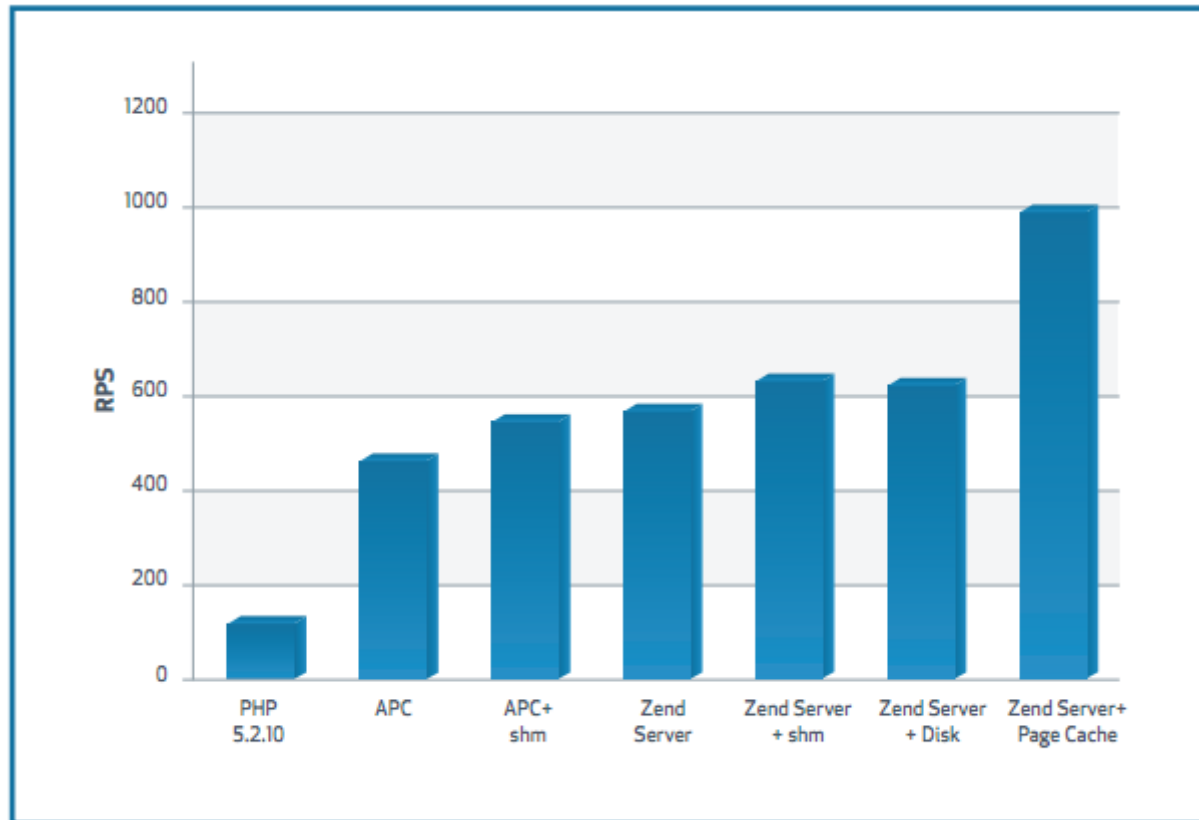
- Clusterweit einstellbar
- Integrierter Bestandteil von Zend Server
- Data Caching (SHM und Disk)
 - ▶ Teile von Antworten (Partial Page Cache)
 - ▶ Daten aus Backend
 - ▶ Config
 - ▶ Ansteuerung über PHP API oder Zend_Cache
 - ▶ Funktion vergleichbar mit Memcached

Zend Server Cache

- Page Caching

- ▶ Speichert ganze Antwortseiten
- ▶ Storage: Disk
- ▶ Cache-Hit Bedingungen einstellbar, Cache-Key einstellbar
- ▶ Zwischenspeicherung der komprimierten Outputs (GZIP)
- ▶ Kann für Cache-Keys und Ausschlussbedingungen auf `$_SERVER` und `$_SESSION` zurückgreifen
- ▶ Arbeitet mit KeepAlive- und SSL- Einstellungen des Webservers
- ▶ Konfiguration über Zend Server GUI oder XML Files

Zend Server Cache



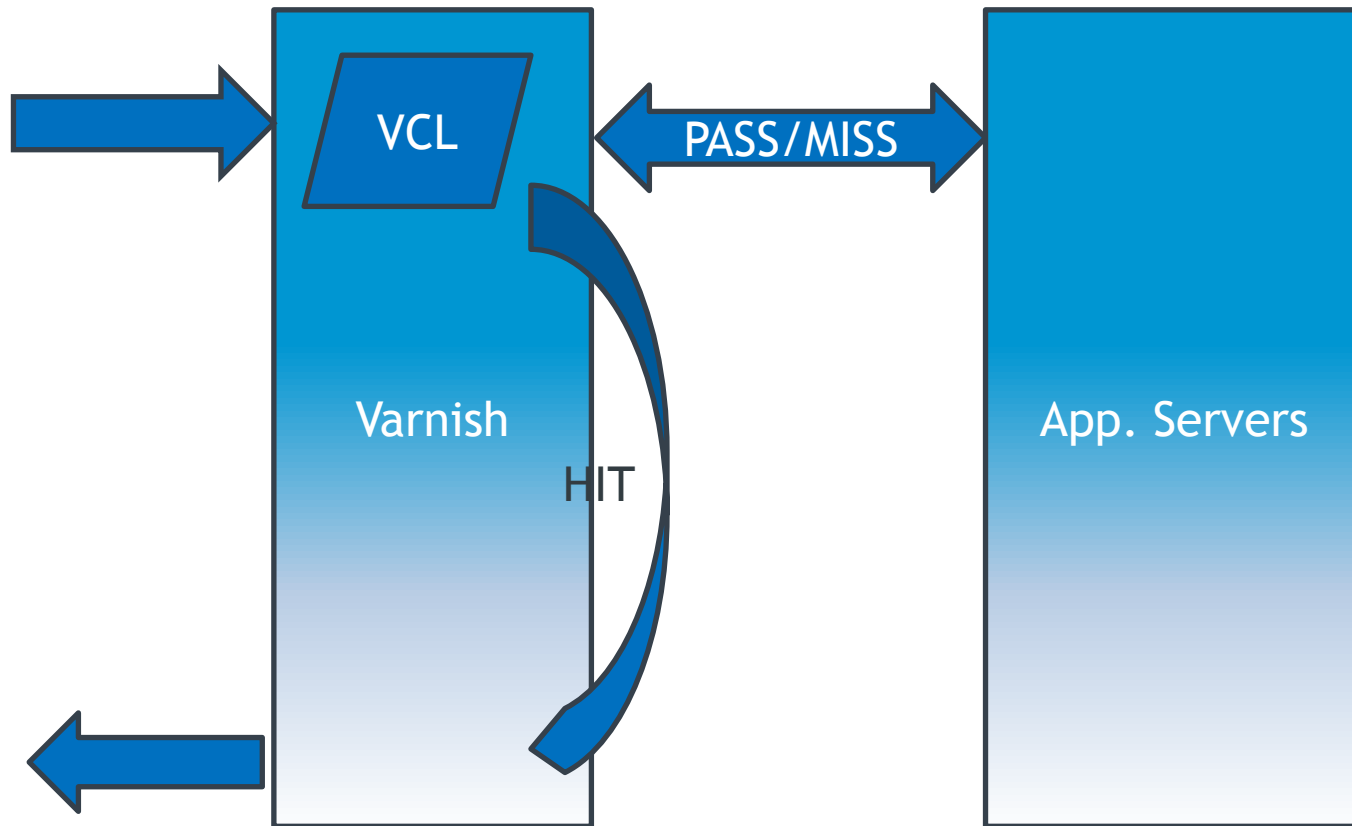
Quelle: <http://static.zend.com/topics/Optimizing-Drupal-Performance-Zend-Acquia-Whitepaper-Feb2010v2.pdf>

Zend Server Cache Limits

- Arbeitet innerhalb PHP
- Daher nur für „dynamische Contents“ sinnvoll
- Apache Prefork produziert viele Prozesse

Varnish Cache

Varnish Cache

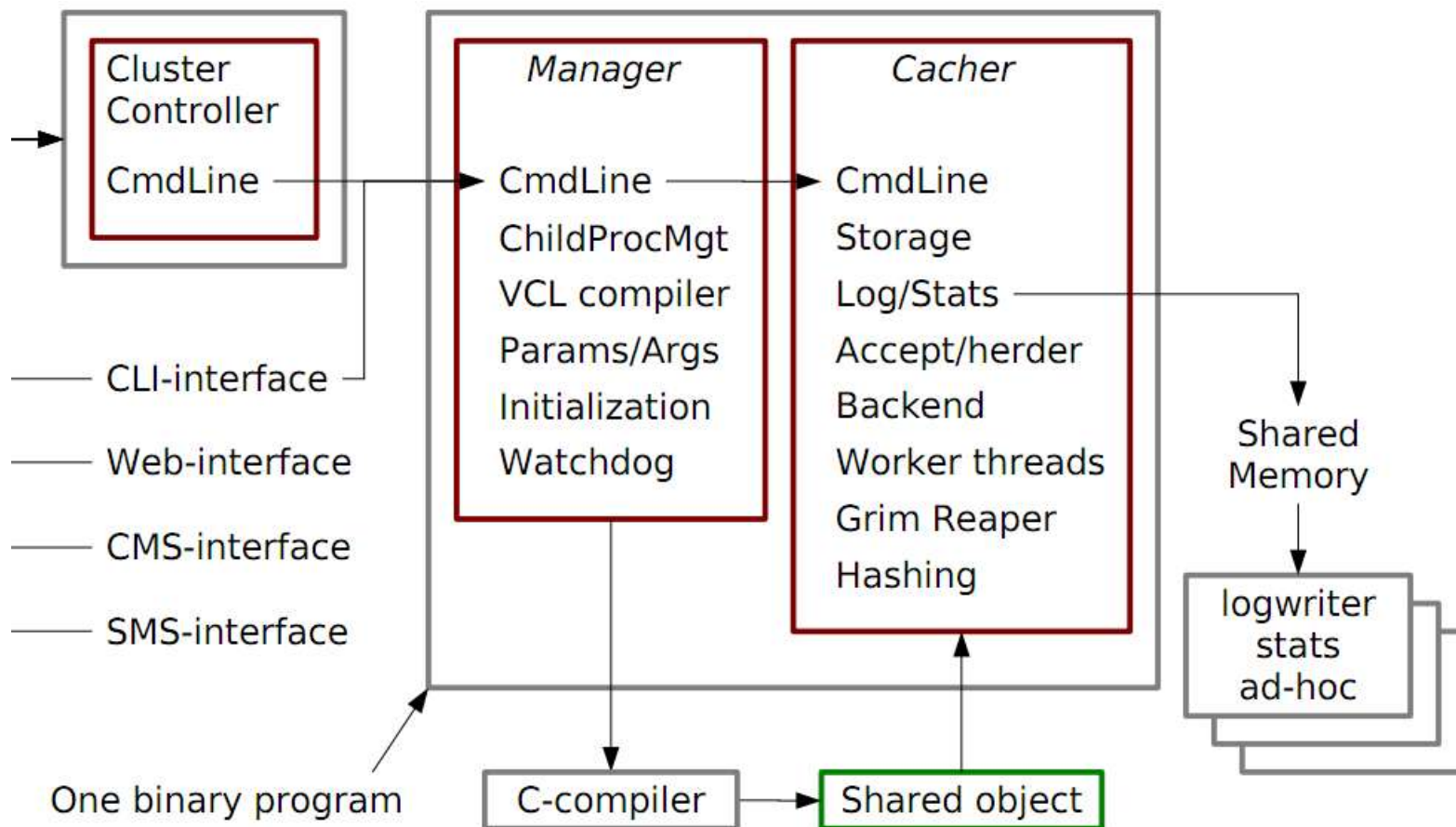


Varnish Cache

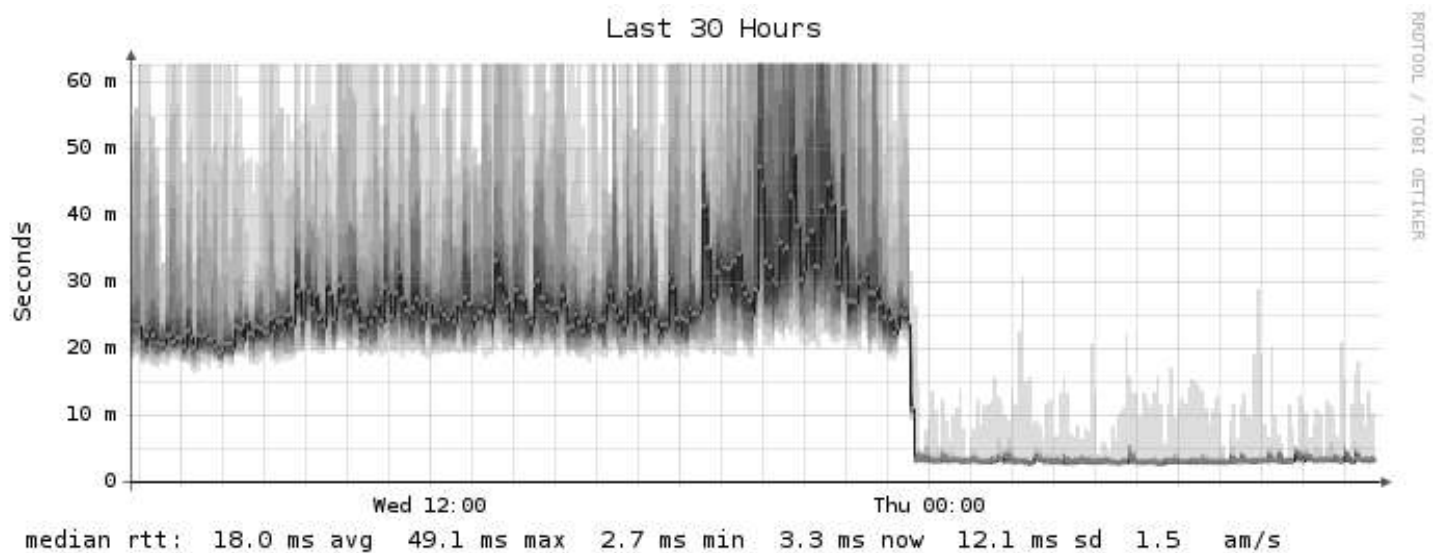
- Reverse Proxy, Ähnlich wie Squid
- Sehr einfach zu installieren / konfigurieren
- Extrem effizient
- Über VCL (Domain Specific Language) steuerbar
- Cache in Memory oder auf Disk
- Logging in Memory Buffer
- Unterstützung von ESI (Edge Side Include)
- Sehr robust
- Kann als Lastverteiler und Fassade für mehrere Backend-Systeme verwendet werden

Varnish Cache

Quelle: http://phk.freebsd.dk/pubs/varnish_tech.pdf



Varnish Cache



From 14 servers running squid to two servers running Varnish

Konfiguration von Varnish

Varnish installieren

```
curl http://repo.varnish-cache.org/debian/GPG-key.txt | apt-key add -  
echo "deb http://repo.varnish-cache.org/debian/ $(lsb_release -s -c)  
varnish-2.1" >> /etc/apt/sources.list  
  
apt-get update  
  
apt-get install varnish
```

Varnish konfigurieren

- In `/etc/default/varnish`:

```
DAEMON_OPTS="-a :80 \  
             -T localhost:6082 \  
             -f /etc/varnish/default.vcl \  
             -S /etc/varnish/secret \  
             -s file,/var/lib/varnish/$INSTANCE/varnish_storage.bin,1G"
```

`-a`: Listen Address, Listen Port

`-T`: Admin Address, Admin Port

`-f`: detail config for Backend and VCL

`-S`: Varnish Admin Secret File

`-s`: Cache Storage configuration (for memory: `-s malloc,4G`)

Varnish konfigurieren

- In `/etc/varnish/default.vcl`:

```
# Default backend definition.  Set this to point to your content
# server.
#
backend default {
    .host = "127.0.0.1";
    .port = "8080";
}

# a lot of additional logic can be added here in VCL
```

Varnish VCL hooks

Function	Description	Possible keywords
vcl_recv	Called after receiving a request. Decides how to serve the request	error, pass, pipe
vcl_pipe	Called after entering pipe mode. Creates a direct connection between the client and the backend, bypassing Varnish all together.	error, pipe
vcl_pass	Called after entering pass mode. Unlike pipe mode, only the current request bypasses Varnish. Subsequent requests for the same connection are handled normally.	error, pass
vcl_hash	Called when computing the hash key for an object.	hash
vcl_hit	Called after a cache hit.	error, pass, deliver
vcl_miss	Called after a cache miss.	error, pass, fetch
vcl_fetch	Called after a successful retrieval from the backend. An 'insert' will add the retrieved object in the cache and then continue to vcl_deliver	error, pass, insert
vcl_deliver	Called before the cached object is delivered to the client.	error, deliver
vcl_timeout	Called by the reaper thread shortly before an object expires in the cache 'discard' will discard the object and 'fetch' will retrieve a fresh copy	discard, fetch
vcl_discard	Called by the reaper thread when a cached object is about to be discarded due to expiration or space is running low	discard, keep

Beispiel-Szenario

Beispiel-Szenario

- 2 Frontend Server, 1 DB-Server (mit cold-standby)
- Typo3 Website und Zend Framework Customer Portal
- Ziel:
 - ▶ 30.000 concurrent users
 - ▶ Davon 800 im Portal, der Rest auf öffentlichen Seiten
- Messung: Jmeter, 2 Lastgeneratoren, 5 Testpläne
- Ramp-Up in 100 Sekunden, danach Testwiederholung

- Constraint: Anwendung soll nicht geändert werden!

Lasttest ohne Caching (nur typo3 cache)

- 5.000 User generieren Load 60 und steigend
- Es kommen mehr neue Requests rein als abgearbeitet werden
- Memory Kritisch, CPU ausgelastet
- Response-Time geht hoch
- Fehleranzahl ebenfalls
- -> Test-Abbruch

Lasttest mit Zend Server Page Cache

- Caching nur auf Typo3-Content, ALLGET
- 15.000 User mit 20% CPU idle
- Hohe Last auf NFS
- Load 20-30, stabil
- Einige sporadische Fehler
- Antwortzeiten ca. 2,5 Sekunden für Seite plus statische Ressourcen Grafik, css, js etc.

Lasttest mit Zend Server und Varnish

- Caching Typo3-Content, statische Elemente, einige Portalseiten
- 30.000 User mit 40% CPU idle
- Kaum Last auf NFS
- Load 4-6, stabil
- Keine Fehler
- Antwortzeiten < 1 Sekunde für Seite plus statische Ressourcen Grafik, CSS, JS etc.

Erfahrungswerte

Erfahrungswerte

- Varnish cached nicht bei Anwesenheit von Cookies
 - ▶ Entfernen der Cookies mit VCL
- Log kann per Log-Daemon geschrieben werden, kostet aber signifikant Performance
- Version in Standard Debian Repository ist zu alt
- Zum Kontrollieren der Cache-Lifetime von Statischen Objekten: mod_expire in Apache einbauen
- Client Caching mit YSlow oder Chrome tools testen
- Mehrere Instanzen funktionieren problemlos auf einem Server

Ausblick

- Management der Varnish Config Cluster-Weit automatisieren
- ESI für dynamische Teile verwenden
- Pre-Fetch ausprobieren
- Behandlung von Backend-Ausfällen mit Varnish abdecken
- CDN-Ähnliche Content-Verteilung mit Hilfe von Varnish

FAZIT

- Varnish reduziert Last auf Apache/PHP
- Antwortzeiten sind dramatisch kürzer als mit Apache/PHP
- Ressourcenverbrauch bei vielen Verbindungen ist geringer
- Backend Last wird reduziert
- Zend Server für SSL Inhalte und komplexere Caching-Entscheidungen (z.B. basierend auf Rollen in Session)

FAZIT

- Mit Varnish und Zend Server hatten wir mehr Probleme Last zu generieren, als sie zu verarbeiten.

Hilfreiche Links

Zend Server:

<http://www.zend.com/de/products/server/>

Zend Server Caching Benchmarks für Drupal:

<http://static.zend.com/topics/Optimizing-Drupal-Performance-Zend-Acquia-Whitepaper-Feb2010v2.pdf>

Zend Server Data Caching:

<http://static.zend.com/topics/Zend-Server-Data-Caching-Whitepaper-0106-T-WP-R1-EN.pdf>

Varnish Dokumentation: <http://www.varnish-cache.org/docs/2.1/>

Quickstart: <http://www.varnish-cache.org/static/getting-started.html>

Technik: http://phk.freebsd.dk/pubs/varnish_tech.pdf

The End.

More about my adventures: <http://100days.de/serendipity>